

L4Re Operating System Framework

Generated on Sun Aug 31 2025 14:51:08 for L4Re Operating System Framework by Doxygen
1.15.0

Sun Aug 31 2025 14:51:08

Chapter 1

Overview

Welcome to the documentation of the L4Re Operating System Framework, or L4Re for short. There are two parts to this documentation: a user manual, which provides a birds eye view of L4Re and its environment, and a reference section which documents the complete programming API.

User Manual

1. [Introduction](#) shortly explains the concept of microkernels and introduces the basic terminology.
2. [Tutorial](#) helps you getting started with setting up the development environment and writing your own first L4Re application.
3. [Programming for L4Re](#) explains in detail the most important programming concepts.
4. [L4Re Servers](#) provides a quick overview over standard services running on the L4Re operating system.

Reference

The second part provides the complete reference of all classes and functions of the L4Re Operating System Framework as well as a list of example code.

Chapter 2

Introduction

The intention of this section is to provide a short overview about the [L4Re](#) Operating System Framework. The general structure of a microkernel-based system will be introduced and the principal functionality of the servers in the basic environment outlined.

2.1 L4Re Microkernel

The [L4Re](#) Microkernel is the lowest-level component of software running in an L4Re-based system. The microkernel is the only component that runs in privileged processor mode. It does not include complex services such as program loading, device drivers, or file systems; those are implemented in user-level programs on top of it (a basic set of these services and abstractions is provided by the [L4](#) Runtime Environment).

Microkernel services are implemented in kernel objects. Tasks hold references to kernel objects in their respective "*object space*", which is a kernel-protected table. These references are called *capabilities*. System calls to the microkernel are function invocations on kernel objects through the corresponding capabilities. These can be thought of as function invocations on object references in an object-oriented programming environment. Furthermore, if a task owns a capability, it may grant other tasks the same (or fewer) rights on this object by passing the capability from its own to the other task's object space.

From a design perspective, capabilities are a concept that enables flexibility in the system structure. A thread that invokes an object through a capability does not need to care about where this object is implemented. In fact, it is possible to implement all objects either in the kernel or in a user-level server and replace one implementation with the other transparently for clients.

2.1.1 Communication

The basic communication mechanism in L4-based systems is called "*Inter Process Communication (IPC)*". It is always synchronous, i.e. both communication partners need to actively rendezvous for IPC. In addition to transmitting arbitrary data between threads, IPC is also used to resolve hardware exceptions, faults and for virtual memory management.

2.1.2 Kernel Objects

The following list gives a short overview of the kernel objects provided by the [L4Re](#) Microkernel:

- **Task** A task comprises a memory address space (represented by the task's page table), an object space (holding the kernel protected capabilities), and on x86 an IO-port address space.
- **Thread** A thread is bound to a task and executes code. Multiple threads can coexist in one task and are scheduled by the microkernel's scheduler.
- **Factory** A factory is used by applications to create new kernel objects. Access to a factory is required to create any new kernel object. Factories can control and restrict object creation.
- **IPC Gate** An IPC gate is used to create a secure communication channel between different tasks. It embeds a label (kernel protected payload) that securely identifies the gate through which a message is received. The gate label is not visible to and cannot be altered by the sender.
- **IRQ** IRQ objects provide access to hardware interrupts. Additionally, programs can create new virtual interrupt objects and trigger them. This allows to implement a signaling mechanism. The receiver cannot decide whether the interrupt is a physical or virtual one.
- **Vcon** Provides access to the in-kernel debugging console (input and output). There is only one such object in the kernel and it is only available, if the kernel is built with debugging enabled. This object is typically interposed through a user-level service or without debugging in the kernel can be completely based on user-level services.
- **Scheduler** Implements scheduling policy and assignment of threads to CPUs, including CPU statistics.

2.2 L4Re System Structure

The system has a multi-tier architecture consisting of the following layers depicted in the figure below:

- **Microkernel** The microkernel is the component at the lowest level of the software stack. It is the only piece of software that is running in the privileged mode of the processor.
- **Tasks** Tasks are the basic containers (address spaces) in which system services and applications are executed. They run in the processor's deprivileged user mode.

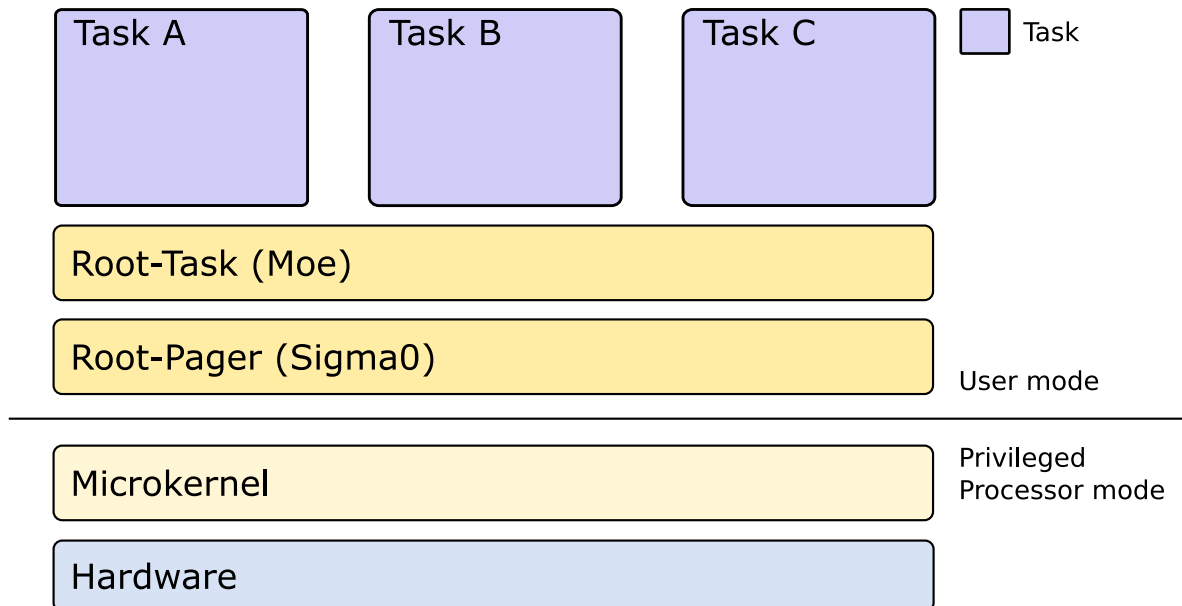


Figure 2.1 Basic Structure of an L4Re based system

In terms of functionality, the system is structured as follows:

- **Microkernel** The kernel provides primitives to execute programs in tasks, to enforce isolation among them, and to provide means of secure communication in order to let them cooperate. As the kernel is the most privileged, security-critical software component in the system, it is a general design goal to make it as small as possible in order to reduce its attack surface. It provides only a minimal set of mechanisms that are necessary to support applications.
- **Runtime Environment** The small kernel offers a concise set of interfaces, but these are not necessarily suited for building applications directly on top of it. The [L4Re](#) Runtime Environment aims at providing more convenient abstractions for application development. It comprises low-level software components that interface directly with the microkernel. The root pager *sigma0* and the root task *Moe* are the most basic components of the [L4Re](#) Runtime Environment. Other services (e.g., for device enumeration) use interfaces provided by them.
- **Applications** Applications run on top of the system and use services provided by the runtime environment – or by other applications. There may be several types of applications in the system and even virtual machine monitors and device drivers are considered applications in the terminology used in this document. They are running alongside other applications on the system.

Lending terminology from the distributed systems area, applications offering services to other applications are usually called *servers*, whereas applications using those services are named *clients*. Being in both roles is also common, for instance, a file system server may be viewed as a server with respect to clients using the file system, while the server itself may also act as a client of a hard disk driver.

2.3 L4Re Runtime Environment

The [L4Re](#) Runtime Environment provides a basic set of services and abstractions, which are useful to implement and run user-level applications on top of the [L4Re](#) Microkernel. They form the [L4Re](#) Operating System Framework.

The [L4Re](#) Operating System Framework consists of a set of libraries and servers. [L4Re](#) follows an object-oriented design. Server interfaces are object-oriented, and the implementation is also object-oriented.

A minimal L4Re-based application needs 3 components to be booted beforehand: the [L4Re](#) Microkernel, the root pager (Sigma0), and the root task (Moe). The Sigma0 root pager initially owns all system resources, but is usually used only to resolve page faults for the Moe root task. Moe provides the essential services to normal user applications such as an initial program loader, a region-map service for virtual memory management, and a memory (data space) allocator.

Chapter 3

Tutorial

This tutorial assumes that the reader is familiar with the basic L4 concepts that were discussed in the [Introduction](#) section.

Here you can find the first steps to boot a very simple setup. The setup consists of the following components:

- [L4Re](#) Microkernel
- Sigma0 — Root Pager
- Moe — Root Task
- Ned — Init Process
- hello — The classical 'Hello World' Application

The guide assumes that you already compiled the base components and describes how to generate an ISO image, with GRUB as a boot loader, that can for example be booted within QEMU.

First you need a `modules.list` file that contains an entry for the scenario.

```
modaddr 0x002000000

entry hello
  kernel fiasco -serial_esc
  roottask moe rom/hello.cfg
  module l4re
  module ned
  module hello.cfg
  module hello
```

This file describes all the binaries and scripts to put into the ISO image, and also describes the GRUB `menu.lst` contents. What you need to do is to set the `make` variable `MODULE_SEARCH_PATH` to contain the path to your [L4Re](#) Microkernel's build directory and the directory containing your `hello.cfg` script.

The `hello.cfg` script should look like the following. A ready to use version can be found in `I4/conf/examples`.

```
local L4 = require("L4");
L4.default_loader:start({}, "rom/hello");
```

The first line of this script ensures that the [L4](#) package is available for the script. The second line uses the default loader object defined in that package and starts the binary `rom/hello`.

Note

All modules defined in `modules.list` are available as data spaces ([L4Re::Dataspace](#)) and registered in a name space ([L4Re::Namespace](#)). This name space is in turn available as 'rom' to the init process ([Ned](#)).

Now you can go to your [L4Re](#) build directory and run the following command.

Note

The example assumes that you have created the `modules.list` and `hello.cfg` files in the `/tmp` directory. Adapt if you created them somewhere else.

```
make grub2iso E=hello MODULES_LIST=/tmp/modules.list MODULE_SEARCH_PATH=/tmp:<path_to_fiasco_builddir>
```

Now you should be able to boot the image in QEMU by running:

```
qemu-system-x86_64 -m 1024 -cdrom images/hello.iso -serial stdio
```

If you press `<ESC>` in the terminal that shows you the serial output you enter the microkernel's debugger... Have fun.

3.1 Customizations

A basic set of bootable entries can be found in `l4/conf/modules.list`. This file is the default for any image creation as shown above. It is recommended that local modification regarding image creation are done in `conf/Makeconf.boot`. Initially you may copy `Makeconf.boot.example` to `Makeconf.boot`. You can overwrite `MODULES_LIST` to set your own modules-list file. Set `MODULE_SEARCH_PATH` to your setup according to the examples given in the file. When configured a `make` call is reduced to:

```
make grub2iso E=hello
```

All other local configuration can be done in a `Makeconf.local` file located in the `l4` directory.

Chapter 4

Programming for L4Re

This part of the documentation discusses the concept of microkernel-based programming in more detail. You should already have a basic understanding of the [L4Re](#) programming environment from the tutorial.

- [L4 Inter-Process Communication \(IPC\)](#)
- [Kernel ABI](#)
- [Capabilities and Naming](#)
- [Spaces and Mappings](#)
- [Initial Environment and Application Bootstrapping](#)
- [Memory management - Data Spaces and the Region Map](#)
- [Program Input and Output](#)
- [Initial Memory Allocator and Factory](#)
- [Application and Server Building Blocks](#)
- [Pthread Support](#)
- tasks and threads
- communication channels
- server loops
- [Interface Definition Language](#)
- hardware access
- [L4Re Build System](#)
- [Kernel Factory](#)

4.1 L4 Inter-Process Communication (IPC)

Inter-process communication (IPC) is the fundamental communication mechanism in the [L4Re](#) Microkernel.

Basically IPC invokes a subroutine in a different context using input and output parameters. It is used to communicate between userland threads and, as a special case, to communicate between a userland thread and a kernel object. IPC provides the only (non-debugging) way of doing system calls.

4.1.1 IPC mechanism

When using this API, an IPC operation can be conducted using the `l4_ipc()` function (or one of its related [helpers](#)). In general it causes a method to be invoked on the called kernel object. An IPC operation consists of a send and receive phase, but either of them can be skipped, that is, an IPC operation can consist of only either a send or a receive phase. IPC is always synchronous and blocking and can be given a timeout. Timeouts can be specified separately for each phase. Invoking the IPC syscall without any phase is deprecated.

On the lowest abstraction level, IPC operations need the following arguments:

- [flags](#) describing the IPC mode,
- the [capability selector](#) of the communication partner,
- a [label](#),
- a [message tag](#), and
- a pair of [timeout](#) values.

During an IPC operation the kernel accesses the UTCB of the current thread to read parameters which are not passed as direct arguments.

As result of an IPC operation the kernel returns a message tag and a label and the kernel also changes UTCB content. For the detailed call signature, refer to `l4_ipc()`. Furthermore, depending on the IPC parameters, the kernel might have transferred the FPU state and capabilities (memory, I/O ports, and/or object capabilities) from the sender to the receiving thread.

The transition between the IPC send phase and the IPC receive phase is atomic, that is, as soon as the send phase has finished, the thread receive phase starts. A relative receive timeout does not start before the send phase has finished (see also below) and a thread which received an IPC call from another thread can assume that the other thread is ready to receive the reply message and the replying thread can therefore reply with a timeout of zero, see [IPC Timeouts](#).

For performance optimization and under certain conditions, the kernel may perform a context switch from the IPC sender to the IPC receiver without consulting the scheduler after the send phase finished. For instance, a client performing an IPC call to a server has to wait for the server anyway. Hence, after the client request was sent to the server, the kernel switches directly to the server thread. This behavior can be disabled by setting the `L4_MSGTAG_SCHEDULE` flag in the sender message tag (see below).

4.1.1.1 IPC Flags

The *flags* defined in `l4_syscall_flags_t` are used by the invoking thread to define the intended IPC operation. The variants of `l4_ipc()` (see [Object Invocation](#)) use the flags

- to request the IPC phases (send-only IPC, receive-only IPC, IPC with send and receive phase), and
- to decide, if the reply capability (see [below](#)) should be used instead of the capability of a dedicated kernel object as target for the send phase (*reply*), and
- to decide, if receiving should wait for an incoming message from any possible sender (*open wait*) instead of a message from a dedicated sender (*closed wait*).

4.1.1.2 Partner capability selector

The *partner capability selector* defines a kernel object as partner of the IPC operation. Some kernel objects forward IPC to a userland thread.

Basically an object capability is represented by `l4_cap_idx_t` where the bits starting from `L4_CAP_SHIFT` are used as index into the local capability table of the current address space.

Specifying `L4_INVALID_CAP` as target for an IPC operation is equivalent to specifying a thread capability of the current thread with full permissions. As a result, the userland thread either communicates with its corresponding kernel thread object (if `L4_PROTO_THREAD` is specified as protocol value, see the description of the message tag below) or the IPC target is the current userland thread. In the latter case, no IPC will be performed and the send phase and the receive phase will block until the corresponding timeout has expired, see below.

A special partner is defined by the *reply capability*. Since it would be impractical (and a security flaw) to always pass an explicit object capability to reply to for each IPC, the kernel generates an implicit one that can be used for just that purpose if the IPC contains an **open wait** phase. The reply capability is valid after a receive phase and points to the kernel object that sent the IPC just received. It can be used only once. The reply capability is selected by setting the `L4_SYSF_REPLY` flag, see above.

4.1.1.3 IPC Label

The IPC label is a machine word which is transferred unchanged from the IPC sender to the IPC receiver when directly sending to a userland thread. However, the primary purpose of the label is to create a relationship between an `L4::Rcv_endpoint` (`L4::lpc_gate` or `L4::lrq`) and the bound thread.

During `L4::Rcv_endpoint::bind_thread()`, a label is specified. If the thread receives an IPC message through the endpoint, that label is delivered to the receiving thread as output parameter of `l4_ipc()` instead of the label specified during the corresponding IPC send operation (see the detailed description of `L4::lpc_gate` for more details on the label with IPC gates). The label is usually used by the receiving thread to invoke the object which is responsible for handling the IPC request of the corresponding endpoint. This mechanism is used by the `L4::Epiface` mechanism for server loops.

4.1.1.4 IPC Message Tag

The *message tag* (`l4_msgtag_t`) describes the payload of the IPC and can also be used to enable certain features. It contains:

- a *protocol value* (cf. `l4_msgtag_t::label()`, also called the tag's *label*),
- the number of items in *UTCB message registers* to transfer (cg. `l4_msgtag_t::words()` and `l4_msgtag_t::items()`), and
- flags (cf. `l4_msgtag_t::flags()` and `L4_msgtag_flags`, may be 0).

The information from the message tag is required by the kernel to transfer the message. The IPC system call returns a message tag as result of the IPC operation. On success, a copy of the message tag specified by the sender is returned if there is a receive phase. Without receive phase, a successful IPC syscall returns the message tag specified as input parameter.

Failures during IPC are signalled using the `L4_MSGTAG_ERROR` flag in the message tag. If this bit is set by the kernel, the content of the returned message tag apart from that bit is undefined and the kernel wrote the actual error code into the `l4_thread_regs_t::error` register of the UTCB (see also *IPC Thread Control Registers*). When an IPC error occurs after the rendezvous of the IPC partners, both partners observe the same error information. If, for

instance, the IPC was aborted using `L4::Thread::ex_regs()`, the sender gets an `L4_IPC_SECANCELED` error while the receiver gets an `L4_IPC_RECANCELED` error. The function `L4Re::chkipc()` can be used to verify that an IPC operation finished successfully: It throws an error if the IPC failed.

The *protocol value* is usually used to distinguish between different protocols of the same interface. Certain protocol IDs are pre-defined when talking to kernel objects, see `L4_msgtag_protocol`. From IPC point of view, the protocol value is just payload that is transferred from sender to receiver and hence doesn't have a dedicated meaning.

By convention, during IPC calls, the protocol value is used for return values, where negative values signify errors. See the [section](#) about L4 RPC return types for further information. The function `L4Re::chksys()` can be used to verify that an RPC call using IPC was successful: It throws an error if the IPC failed or if the returned protocol value is negative.

4.1.1.5 IPC Timeouts

As written above, IPC *timeouts* are specified separately for the send phase (IPC send timeout) and the receive phase (IPC receive timeout). The timeout of one phase is encapsulated by `l4_timeout_s`. Both timeouts are combined into a single `l4_timeout_t` parameter. Timeouts are either relative to the current time of the invoking thread or absolute. In the latter case, the absolute time of the deadline of the respective phase is written into a UTCB buffer register (see `l4_timeout_abs()`). The relative timeout of the receive phase starts immediately after the send phase has finished.

Two specific timeout values are sufficient for most IPC operations:

- `L4_IPC_TIMEOUT_NEVER` is used if the IPC partner might not yet be ready. Usually a client trusting a server will perform an IPC call with an infinite timeout for both phases. The send phase will take some time if the IPC receiver is currently not ready. The receive phase will take some time as the server needs time to serve the request. Also, a server will usually wait with an infinite receive timeout for the next request (see below for a possible exception).
- `L4_IPC_TIMEOUT_0` is used when it is either certain that the IPC receiver is currently ready or if the IPC sender doesn't want to wait if the IPC receiver is not ready. The former case applies to a thread which was called with an IPC call, for example a server got a client request. The reply to the IPC call should use a timeout of zero to ensure that a client doesn't block a server (server could not deliver the result of the request). See also `L4::ipc_srv::Default_timeout`. Another case is an IPC send operation for waking up another thread from an IPC receive operation. If the other thread is not ready to receive, then it might be already woken up and it does not make sense to wait any longer. Also triggering an IRQ is usually done with a send timeout of 0, see `L4::Triggerable::trigger()`.

In certain cases it also makes sense to specify an IPC timeout different from "never" or "zero":

- A server might leave the server loop after some time to perform idle work (see `L4::ipc_srv::Server_iface::add_timeout()`).
- A thread does not want to wait for an infinite time if the partner is not ready. This could be also some safety measure.
- A thread wants to block for a certain amount of time without consuming CPU time. The `l4_ipc_sleep()` function specifies the `L4_INVALID_CAP` as target for an IPC receive operation and specifies the intended relative waiting period as IPC timeout. Waiting for an absolute timeout would be possible with similar code.

Note

The kernel IPC path is optimized for the two special cases using `L4_IPC_TIMEOUT_NEVER` and `L4_IPC_TIMEOUT_0`. Specifying a different timeout causes more maintenance effort for the kernel.

Special care is required if a finite timeout for the receive phase of an IPC call is specified: The IPC receive operation could abort before the partner was able to send the reply message. Under certain circumstances the partner may still have the temporary reply capability to the calling thread and may use this capability to reply to the caller at a later, unexpected time specifying an arbitrary IPC label. This case is relevant for servers which call another, possibly untrusted, server while serving a client request.

4.1.1.6 User-level Thread Control Block

The **UTCB** is located on a power-of-2-sized and power-of-2-aligned memory area shared between userland and the kernel (`L4::Task::add_ku_mem()`). The UTCB is bound to a thread during the `L4::Thread::control()` operation with the `L4::Thread::Attr` parameters set up using `L4::Thread::Attr::bind()`. The UTCB is used for IPC-related data exchange and is set up before invoking `l4_ipc()`. To access certain parts of the UTCB, the corresponding functions have to be used (there is no data type `L4_utcb` or similar). The UTCB consists of:

- the **message registers** `MR[0]`, `MR[1]`, ..., `MR[n-1]` with $n = \text{L4_UTCB_GENERIC_DATA_SIZE}$ (access using `l4_utcb_mr()`),
- the **buffer descriptor register** `BDR` (access using `l4_utcb_br()`, see `l4_buf_regs_t::bdr`),
- the **buffer registers** `BR[0]`, `BR[1]`, ..., `BR[m-1]` with $m = \text{L4_UTCB_GENERIC_BUFFERS_SIZE}$ (access using `l4_utcb_br()`),
- the **thread control registers** (access using `l4_utcb_tcr()`, includes the IPC error code), and
- in case of an exception IPC, the register state of the thread which triggered the exception (access using `l4_utcb_exc()`).

IPC to a kernel object requires the setup of the UTCB of the corresponding userlevel thread. IPC between userlevel threads requires the setup of UTCBs of both partners.

The kernel changes only the following UTCB content:

- The message registers of the UTCB of the receiver of an IPC, and
- the IPC error field of the thread invoking `l4_ipc()` if there was an error during IPC.

4.1.1.6.1 IPC Message registers

The *message registers* contain *untyped items* and *typed items*. The sender's typed items are interpreted by the kernel in conjunction with the receiver's *receive items*. Each typed send item occupies two message registers. The untyped items, on the other hand, are free to be used by the communication partners to exchange data: The content of all message registers used for untyped items (`l4_msgtag_t::words()`) is copied from the UTCB of the IPC sender to the UTCB of the IPC receiver.

A typed send item consists of a *flexpage* (see `l4_fpage()`, `l4_obj_fpage()`, and `l4_iofpage()`) of the to be transferred capabilities (*flexpage word*) and a *message word*. There are two types of send items: *map items* and *void items*. For a void item, the message word is all zero. For a map item, the message word contains:

- the *compound bit* allowing to use the same receive buffer for the subsequent typed send item (scatter-gather behavior, see `L4_ITEM_CONT` of `l4_msg_item_consts_t`),
- the *type bit* defining this typed send item as a *map item*,
- the *grant flag* for delegating the access to the flexpage from the sender to the receiver and atomically removing the rights from the sender (see `l4_msg_item_consts_t`),
- *attributes* with semantics depending on the item type; for memory mappings, they contain cacheability information (see `l4_fpage_cacheability_opt_t`); for objects, they contain additional rights (see `L4_obj_fpage_ctl`),
- the *send base* (also called *hot spot*) defining what is actually mapped when send and receive flexpages have a different size.

A typed send item can be created using `l4_sndfpage_add()`. This function sets the compound bit unconditionally. Alternatively, the functions `l4_map_control()` and `l4_map_obj_control()` can be used to set up the message word of a map item for a memory flexpage respective for objects.

See [below](#) for a description how typed items are transferred.

4.1.1.6.2 IPC Buffer Registers

The *buffer registers* and *buffer descriptor register* are interpreted by the kernel during the receive phase (if any). [Buffer](#) registers define *receive items* which are required to receive typed send items (memory, I/O ports or object capabilities). To specify a receive item, up to three buffer registers are required:

- A *small receive item* ([L4::ipc::Small_buf](#)) occupying one buffer register is sufficient to receive one object capability.
- A *receive item* ([L4::ipc::Rcv_fpage](#)) occupying two or three buffer registers (*message word*, a *flexpage*, and an optional destination task capability index) is required to receive memory flexpages, I/O ports, or multiple object capabilities.

4.1.1.6.3 IPC Buffer Descriptor Register

The buffer descriptor register defines indices of buffer registers used to receive dedicated types of send items and also contains a flag:

- 5 bits starting at [L4_BDR_MEM_SHIFT](#) define the index of the first receive item used for memory flexpages.
- 5 bits starting at [L4_BDR_IO_SHIFT](#) define the index of the first receive item used for I/O flexpages.
- 5 bits starting at [L4_BDR_OBJ_SHIFT](#) define the index of the first receive item used for object flexpages.
- The [L4_UTCB_INHERIT_FPU](#) can be set using [l4_utcb_inherit_fpu\(\)](#) and allows to receive the FPU state from the IPC sender. This is only relevant if the sender set [L4_MSGTAG_TRANSFER_FPU](#) in the message tag.

For most use cases, a BDR value of zero is sufficient. In that case, if `BR[0]` contains a void item, no capabilities are received. Otherwise, only one type of capabilities (memory, I/O ports or objects) can be received even if there are several receive items. For more complex setups that require receiving different types of capabilities in one receive operation, other BDR values are necessary.

The BDR of the receiving thread is only used by the kernel if at least one typed item is transferred during the IPC or if [L4_MSGTAG_TRANSFER_FPU](#) is set in the UTCB of the sending thread.

4.1.1.6.4 IPC Thread Control Registers

The [l4_thread_regs_t::error](#) register contains the IPC error code in case the [L4_MSGTAG_ERROR](#) flag is set in the message tag returned by an IPC syscall. Otherwise this register is not touched by the kernel. See [l4_ipc_tcr_error_t](#) for a detailed enumeration of all possible error codes during an IPC operation.

The [l4_thread_regs_t::free_marker](#) is set by the kernel to zero immediately before a thread is destroyed. This value indicates that the kernel does not longer use the UTCB and it can be re-used by other threads.

The other members of [l4_thread_regs_t](#) are never touched by the kernel.

4.1.1.7 Transfer of Typed Send Items

The kernel interprets all typed items in the sender UTCB ([l4_msgtag_t::items\(\)](#)) and performs the following operations while modifying the corresponding typed items in the receiver UTCB:

- If the message item of the sender is void, this item is ignored and the message word of the corresponding typed item in the receiver UTCB is set to zero (making it a void item). The flexpage word of this item is not changed.
- Otherwise, if the type bit of the message item of the sender is not set, the IPC is aborted with [L4_IPC_SEMSGCUT](#) / [L4_IPC_REMSGCUT](#).
- Otherwise, if there is a receive item corresponding to the flexpage type of the send item available (see [IPC Buffer Descriptor Register](#)), information described by the flexpage is transferred to the receiver.

In the last case, the message word of the typed item in the receiver UTCB is modified to contain the send base, the type and the size of the transferred flexpage, as well as a copy of the compound bit and the type bit of the send item. If the receiver ordered a local ID in the corresponding receive item, the kernel attempts to apply special rules, see [L4_RCV_ITEM_LOCAL_ID](#). Otherwise, regular mappings as described by the flexpage of the send item are created in the receiver space.

A receive item forms a *receive window* of a specific address and size in the receiver space. Each typed send item that is a map item requires a corresponding receive item. By default, there is a one-to-one mapping (one receive item per typed send item) but it is also possible to use one receive item to receive several typed send items: The compound bit (see [l4_msg_item_consts_t](#)) of a send item defines if the following typed send item shall use the same receive item as the current one for receiving the flexpage. If the compound bit is set, proper values of the send base shall be used to prevent overlapping of addresses in the receiver space.

The send base is relevant when the size of the receive flexpage differs from the size of the transferred resource. As a typical example, triggering a memory page fault opens a receive window covering the entire memory address space of the faulting thread. The pager will usually reply a memory flexpage smaller than the entire address space of the faulting thread. Hence, the pager has to specify a proper base which is used as offset of the sent object in the receive flexpage in the *receiver space*. If the sender flexpage is bigger than the receive window, a flexpage of the size of the receive window starting at the send base in the *sender space* is transferred to the receiver.

The kernel will stop transmission of typed items before [l4_msgtag_t::items\(\)](#) is reached either if it finds a void item as receive buffer or if the flexpage type of the send item does not match the flexpage type of the corresponding receive item. Under both conditions, the IPC is aborted with [L4_IPC_SEMSGCUT](#) / [L4_IPC_REMSGCUT](#).

Note

The kernel ignores the flexpage access rights of the receive items. The actual rights for a transferred resource in the target address space are defined by the access rights to that resource of the IPC sender address space and the flexpage access rights in the typed send item. Additionally, when transferring object capabilities, the transferred rights also depend on the sender's rights on the capability invoked for IPC.

The kernel does not unmap capabilities in the receive window when there is no capability present at the corresponding index at the sender. Further, the receiver cannot reliably detect at which capability indices it received capabilities in its receive windows. Therefore, before receiving from an untrusted source, all receive windows should be cleared. Otherwise the receiver may erroneously associate a capability in one of its receive windows with his last IPC partner although it was actually received in an earlier IPC.

However, the kernel indicates if at least one object capability was received or not, see [L4::lpc::Snd_fpage::cap_received\(\)](#).

4.1.2 Examples

A number of examples show the interplay of the concepts introduced so far.

4.1.2.1 User Thread to Kernel Object

The `L4::Scheduler` kernel object has a method `L4::Scheduler.idle_time()`. It takes a set of CPUs to query, represented by two machine words.

In user space, the function `L4::Cap<L4::Scheduler>->idle_time()` is called, which does the following:

- Fill `MR[0]` with a constant identifying the method being called (`L4_SCHEDULER_IDLE_TIME_OP`).
- Fill `MR[1]` and `MR[2]` with the two words making up the CPU set.
- Set up the message tag with the protocol value (`L4_PROTO_SCHEDULER`), the number of untyped and typed items (3 and 0), and some flags.
- Call `l4_ipc()` with the partner capability ID, the tag, the pointer to the filled UTCB, infinite timeouts, and with flags indicating a send and receive phase. (The label does not matter in this case.)

This function traps into kernel space using standard platform specific mechanisms. The kernel reads the protocol value on the message tag, checks that the partner capability ID refers to a valid object that speaks that protocol, and dispatches the message to the appropriate handler. The handler fills the first 64 bits of the message registers with the computed time value. This will cover `MR[0]` on 64-bit architectures and `MR[0]` and `MR[1]` on 32-bit architectures. It then sets up the return message tag:

- The number of untyped items is 1 or 2.
- The number of typed items is 0.
- The protocol value contains the return value and is set to 0.
- An error would be signalled as a negative protocol value. Under certain conditions (e.g. wrong kernel object capability specified), the error is signalled as IPC error (see `L4_MSGTAG_ERROR` in the description of the `IPC Message Tag`).
- (The return label is as irrelevant in this case as the send label.)

This reply is received by the receive phase of the original `l4_ipc()` call from userland. Finally the `l4_ipc()` function copies the time value out of the message registers and forwards it with a possible error from the message tag flags to its caller.

4.1.2.2 User Thread to User Thread

When the target kernel object is of type `L4::Thread` (or `L4::ipc_gate`, but we will cover this in the example below) and the message tag's protocol value is not `L4_PROTO_THREAD`, the kernel will forward the IPC message to the userland thread represented by the kernel object. There it can be received by a call to `l4_ipc()`. The restriction of the protocol number is necessary because one also wants to invoke `L4::Thread`'s control methods such as `L4::Thread.switch_to()` or `L4::Thread.ex_regs()`. Besides this restriction, the interpretation of all the IPC parameters and the untyped items of the UTCB is up to the communication partners.

4.1.2.2.1 Simple Messages

A simple example is a client calling a server to have a computation performed on a value: Similar to IPC from a userland thread to a kernel object, the client writes the value to `MR[0]`. It sets up the message tag with some agreed upon protocol value (which, as explained above, must be different from `L4_PROTO_THREAD`), number of untyped items to 1, typed items to 0, and no flags set. The client may want to pass a label that identifies itself, as many clients can use the server. In this context, the identifier might also be passed via the message registers, but the label is the proper place for this, as it gets a special treatment by the kernel for IPC gates (covered by the example below). The client then calls `l4_ipc()` with the tag, label and flags indicating it wants a send phase and a receive phase (as it wants a result back). The target is the capability index referring to a capability for the `L4::Thread` kernel object of the server.

To be able to receive an IPC message, the server has set up a UTCB of its own and called `l4_ipc()` with flags indicating it only wants to enter a receive phase and it accepts IPCs from any partner. This is called an **open wait**. (The alternative would be to specify a capability index referring to a `L4::Thread` capability to exclusively receive from.)

Both system calls (the send IPC initiated by the client and the receive IPC initiated by the server) may be seen by the kernel in any order but the IPC will not start before sender and receiver are ready. In that case the kernel will copy the relevant message registers the client specified in the message tag from the client's UTCB to the server's UTCB, in the current example just `MR[0]`. It will then switch the client to the receive phase of its call (which cannot yet be executed) and return the server's call with the message tag and label.

The server inspects the tag for the correct protocol value and number of untyped items passed, inspect the label to decide whether it maybe wants special treatment of this particular client, performs the computation on `MR[0]` and writes the result back to `MR[0]` (or to more words, depending on what exactly it computes). It sets up the tag in the usual way, but probably needs to pass no label, as the client knows who it is talking to.

For the reply, the server makes use the reply capability (see above). Since the client sent the last IPC to the server, the reply capability will point to it. So when the server calls `l4_ipc()` with the computed result in the message registers and using the reply capability as target, the kernel knows to forward this to the client's thread. The kernel copies the message registers from the server's UTCB to the client's UTCB, and the client's `l4_ipc()` system call, which is still stuck in the receive phase, is returned with the tag.

The client looks at the tag and then the message registers for its wanted result and the example is concluded.

4.1.2.2.2 Send Items

IPC between userland threads is also used to transfer typed items: Memory, I/O ports, and objects, all represented as flexpages. Typed items and untyped items can be part of the same IPC. As general rule, the sender specifies what he wants to send, the receiver where and how much it wants to receive, and the kernel checks the required permissions before doing the actual transfer. As written before, this mechanism is synchronous and the receiver cannot be transferred items against its will.

See also

[Flexpages](#)

Suppose a client wants a server to have read only access to a page of its memory. The client sets up a flexpage covering the page and with only the `L4_FPAGE_RO` right set. The server sets up a flexpage of a memory region where it will receive the mapping. This may be larger than one page, suppose for our case four pages, in which case the exact position of the mapping will be resolved by the send base provided by the sender. The client combines the hot spot and some flags into a machine word and writes it to `MR[0]` (see also `l4_map_control()`). At `MR[1]` follows the flexpage it wants to send (see also `l4_fpage()`). The server does almost the same but writes the words to `BR[0]` and `BR[1]`. (The server could also specify a hot spot, but it is currently ignored by the kernel.) The

client specifies 1 typed and 0 untyped items in the message tag. The server writes 0 to `BDR` to specify that the first receive item starts at the first buffer register.

Both client and server initiate their IPC, the client with only a send phase to the server, and the server with an open wait receive phase. The kernel checks the compatibility of the send items and the receive buffers (e.g. that no object capability flexpage is sent to a receive buffer describing a memory mapping flexpage) and updates its internal structures to reflect the change. In our case, the sender's hot spot indicates to which of the four pages that make up the receive buffer the sent page should be mapped. The kernel also translates the typed send item to the server's address space and stores it in the server's UTCB at `MR[0]` and `MR[1]`.

Once the server returns from its syscall, it will have read access to the client's shared page.

4.1.2.3 User Thread to User Object

A common use case for thread to thread communication is when a server implements a number of object interfaces and a client wants to invoke methods on them. For security reasons, the server does not want to hand out its thread capability to everyone it nonetheless wants to serve. It also may not want to allow every client access to everyone of its interfaces. For this purpose, IPC gates implemented by the kernel object `L4::ipc_gate` can be used. An IPC gate can be bound to a thread and forwards IPC to it. In doing so it applies two transformations

1. It sets the label to a predefined value.
2. It changes the rights of transferred items (see `L4_CAP_FPAGE_S`).

For each object of every interface the server implements, it creates an IPC gate and binds it to itself (see `L4::ipc_gate::bind_thread()`). It sets the gate's label to a unique value identifying the object. Then it hands the gate out to clients. Several clients can be handed the same gate and will all end up invoking methods on the same object.

Instead of setting the target as the server's thread kernel object, the client uses the IPC gate's instead. The label the client sends is irrelevant, as the gate will overwrite it with the value the server has set during the bind operation. The server executes an open wait, and the kernel performs the same operation as in the above [example](#) with the transformed IPC finally ending up in the server's thread.

The server checks that the received label refers to one of its objects. It then checks if the protocol value in the message tag matches the interface the object implements. Then it invokes the method specified in `MR[0]` with the rest of the arguments. Finally it returns the results via UTCB and message tag to the reply capability and waits for the next IPC.

4.2 Kernel ABI

This section details the binary representation of the IPC interface of the kernel. It accompanies the [L4 Inter-Process Communication \(IPC\)](#) section. The details presented here are usually not relevant when developing L4Re applications and can therefore be skipped by many readers.

Note

The kernel ABI is subject to change. Please use the API instead of relying on particular binary representations.

The following notation is used to indicate how particular data fields are used:

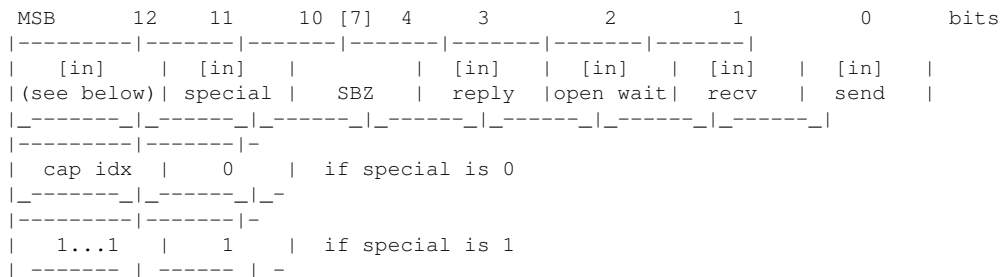
- `[in]`: The kernel reads and interprets this field.
- `[out]`: The kernel writes this field with information provided by the kernel.
- `[cpy]`: The kernel copies this field from sender to receiver (without interpretation if `[in]` is not listed as well).

The above indications may be combined.

4.2.1 Capability selector and flags

See [partner capability selector](#) and [IPC flags](#).

The kernel reads and interprets all the fields ([in]).



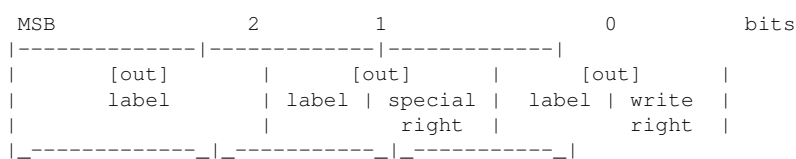
- Bits 0...3 [in]: These bits correspond to the flags defined in [l4_syscall_flags_t](#). The individual bits correspond to [L4_SYSF_REPLY](#), [L4_SYSF_OPEN_WAIT](#), [L4_SYSF_RECV](#), [L4_SYSF_SEND](#). Note that not all combinations of those bits are defined; see [l4_syscall_flags_t](#).
- Bits 4...10 [in] SBZ: should be zero
- Bit 11 [in] special: Set when using [L4_INVALID_CAP](#), otherwise unset.
- Bits 12...MSB [in]: Capability index if special is unset, otherwise all those bits should be one (see [L4_INVALID_CAP](#), [partner capability selector](#) and [l4_cap_idx_t](#)).

4.2.2 Label

See [IPC label](#).

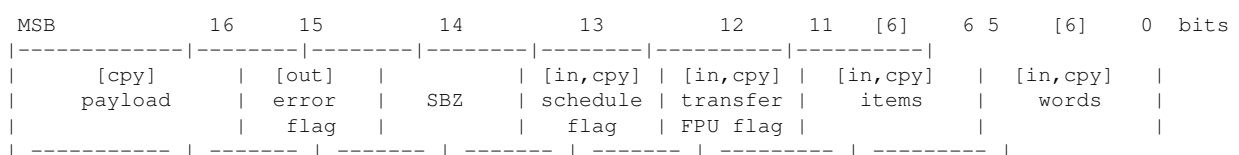
When IPC is sent via a thread capability, the label is copied to the receiver unchanged ([cpy]).

When IPC is sent via an IPC gate, the sent label is ignored and the kernel provides the bitwise OR (|) of the IPC gate label and the sender's write and special permissions (see [L4_CAP_FPAGE_W](#) and [L4_CAP_FPAGE_S](#)) of the used capability ([out]):



4.2.3 Message tag

See [IPC message tag](#). Note that, for a message tag returned by the kernel, if the error flag is set, all other contents of the message tag is undefined.



- Bits 0...5 [in,cpy] `words`: Number of (untyped) message words in the UTCB's message registers. See [l4_msgtag_words\(\)](#) and [l4_msgtag_t::words\(\)](#).
- Bits 6...11 [in,cpy] `items`: Number of typed message items in the UTCB's message registers. See [l4_msgtag_items\(\)](#) and [l4_msgtag_t::items\(\)](#).
- Bit 12 [in,cpy] `transfer` FPU flag: See [L4_MSGTAG_TRANSFER_FPU](#).
- Bit 13 [in,cpy] `schedule` flag: See [L4_MSGTAG_SCHEDULE](#).
- Bit 14 SBZ: should be zero
- Bit 15 [out] `error`: See [L4_MSGTAG_ERROR](#), [l4_msgtag_has_error\(\)](#) and [l4_msgtag_t::has_error\(\)](#).
- Bits 16...MSB [cpy] `payload`: Transferred to receiver unchanged; not interpreted by kernel (unless it is the communication partner). For IPC calls or send-only IPC, this is usually the protocol. For replies, this is usually used for return values and server error signaling. See [l4_msgtag_label\(\)](#) and [l4_msgtag_t::label\(\)](#).

4.2.4 Timeouts

See [IPC timeouts](#) and [l4_timeout_t](#).

The kernel reads and interprets all the fields ([in]).

```

31      [16]      16 15      [16]      0 bits
|-----|-----|
|      [in]      |      [in]      |
| send timeout  | receive timeout |
|_-----|_-----|

```

A timeout has the following format. There are two special timeout values:

- *Zero timeout*: Only bit 10 is set. See [L4_IPC_TIMEOUT_0](#).

```

15  [5]  11  10   9      [10]      0 bits
|-----|-----|-----|
|      0      |  1  |      0      |
|_-----|_-----|_-----|

```

- *Infinite timeout*: All bits are unset. See [L4_IPC_TIMEOUT_NEVER](#).

```

15      [16]      0 bits
|-----|
|      0      |
|_-----|

```

Otherwise, the timeout is either relative or absolute, which is specified by bit 15.

- *Relative timeout*: If bit 15 is unset, the timeout is `mantissa * 2 ^ exponent` micro seconds relative to the current time. The `mantissa` must not be zero:

```

15  14  [5]  10  9      [10]      0 bits
|----|-----|-----|
|  0  | exponent | mantissa 0  |
|_---|_-----|_-----|

```

- *Absolute timeout*: If bit 15 is set, an absolute timeout is specified in the UTCB's buffer registers starting at `buf reg idx` (the particular number of registers depends on the architecture; see [l4_timeout_s](#)):

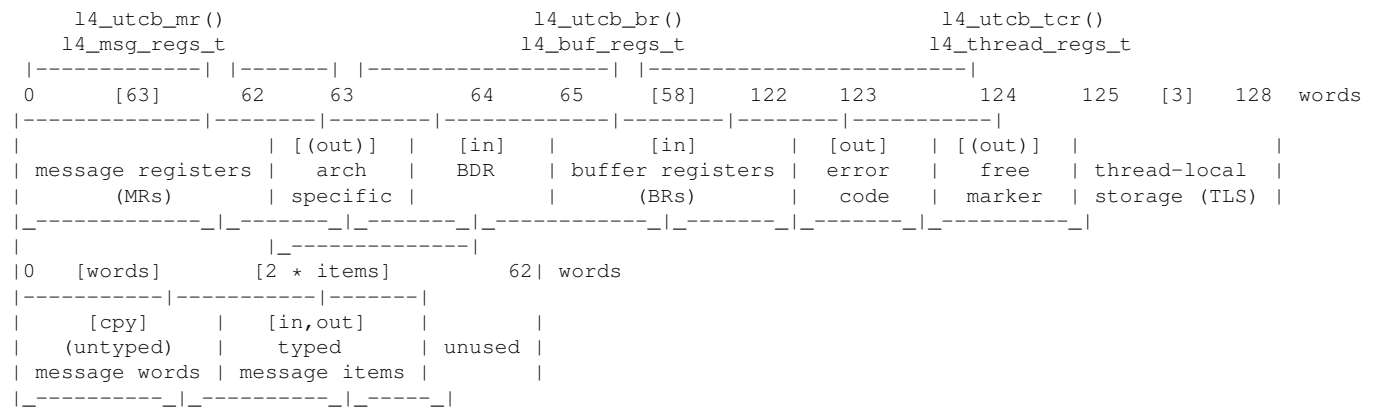
```

15  14      [9]      6 5      [6]      0 bits
|----|-----|-----|
|  1  |      SBZ      | buf reg idx |
|_---|_-----|_-----|

```

4.2.5 User-level thread control block (UTCB)

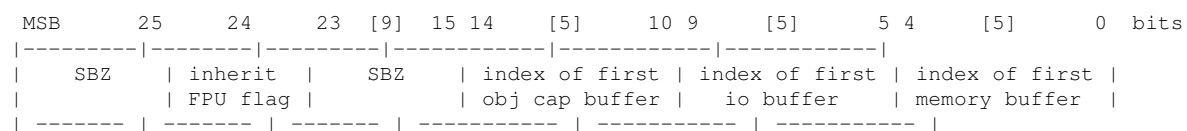
See [User-level thread control block \(UTCB\)](#).



- Words 0...62 MRs: See [IPC Message registers](#) and [l4_utcb_mr\(\)](#). The number of message registers is defined by [L4_UTCB_GENERIC_DATA_SIZE](#). The actually used message registers are defined by `words` and `items` in the [message tag](#). The layout of a typed message item varies depending on being an input or output value, see [typed message items](#).
- Word 63 [(out)]: Depending on the architecture, this word may be used by the kernel to signify the position of a thread's UTCB in memory. See architecture-specific implementation of [l4_utcb\(\)](#). If at all, the kernel writes this word when kernel-user memory is set up as UTCB while binding a thread to a task; see [l4_thread_control_bind\(\)](#), [L4::Thread::Attr::bind\(\)](#).
- Word 64 [in] BDR: See [buffer descriptor register](#).
- Words 65...122 [in] BRs: See [IPC Buffer Registers](#), [receive items](#) and [l4_utcb_br\(\)->br](#). The number of buffer registers is defined by [L4_UTCB_GENERIC_BUFFERS_SIZE](#).
- Word 123 [out] `error code`: See [IPC Thread Control Registers](#) and [l4_utcb_tcr\(\)->error](#).
- Word 124 [(out)] `free marker`: Written by the kernel, but not necessarily during IPC. See [IPC Thread Control Registers](#) and [l4_utcb_tcr\(\)->free_marker](#).
- Word 125...128 TLS: Ignored and left untouched by the kernel. See [IPC Thread Control Registers](#) and [l4_utcb_tcr\(\)->user](#).

4.2.5.1 Buffer descriptor register

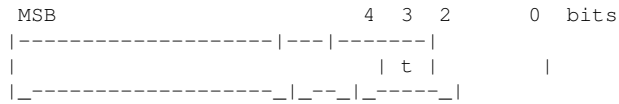
See [IPC Buffer Descriptor Register](#) and [l4_utcb_br\(\)->bdr](#).



4.2.6 Typed message items

The number of words in a typed message item varies depending on the particular kind of item. However, for the first word, the following properties are shared:

- *Void item*: If all bits of the first word of a typed message item are zero, then it is a void item.
- *Non-void item*: The first word of a non-void typed message item has the following binary layout:



Bit 3 (t) is the type bit. If t is set, the item is a map item. Currently, map item is the only supported type. Hence, this bit must be set for all items except for void items.

There are three sub-types of typed message items: *send items*, *receive items*, and *return items*; see [Message Items](#).

Many typed items make use of flexpages, therefore, these are described before the various kinds of typed items. Note that flexpages are also used outside of typed message items, e.g., for [L4::Task::unmap\(\)](#).

4.2.6.1 Flexpages

A flexpage consists of a single word and, except for some special values, describes a range in an address space, see [flex pages](#).

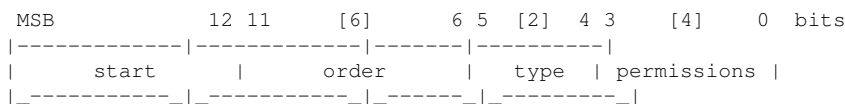
The general layout is defined as follows:



- Bits 4...5 type: See [l4_fpage_type\(\)](#) and [L4_fpage_type](#).

The type [L4_FPAGE_SPECIAL](#) only supports some selected values, which are only supported for selected interfaces; see [L4_FPAGE_SPECIAL](#).

The other types share the same layout:



- Bits 0...3 permissions: See [l4_fpage_rights\(\)](#), [L4_fpage_rights](#) (memory space) and [L4_cap_fpage_rights](#) (object space). Should be zero for I/O port space.
- Bits 6...11 order: The \log_2 size of the flexpage. See [l4_fpage_size](#).
- Bits 12...MSB start: The starting page number / I/O port number / capability index of the flexpage. Must be aligned to the flexpage size. See [l4_fpage_page\(\)](#), [l4_fpage_memaddr\(\)](#), [l4_fpage_ioport\(\)](#) and [l4_fpage_obj\(\)](#).

Also see [l4_fpage\(\)](#) (memory space), [l4_iofpage\(\)](#) (I/O port space) and [l4_fpage_obj\(\)](#) (object space).


```

|-----|-----|-----|---|-----|---|-----|
| hot_spot | order | type | 1 | 01 | c | undefined |
|_-----|_-----|_-----|_--|_-----|_--|_-----|

```

10: Used if the receive item's `rcv_id` bit was set and the conditions for transferring an IPC gate label were fulfilled. In that case, no mapping is done for this item and the payload consists of the bitwise OR (|) of the IPC gate label and the write and special permissions (see [L4_CAP_FPAGE_W](#) and [L4_CAP_FPAGE_S](#)) that would have been mapped (also see [L4::lpc::Snd_fpage::id_received\(\)](#)):

```

                                     2 1      0 bits
|-----|-----|-----|---|-----|---|-----|
| hot_spot | order | type | 1 | 10 | c | label | rights |
|_-----|_-----|_-----|_--|_-----|_--|_-----|

```

11: Used if the receive item's `rcv_id` bit was set and the conditions for transferring the sender's flexpage word were fulfilled. In that case, no mapping is done for this item and the payload is a copy of the sender's flexpage word (also see [L4::lpc::Snd_fpage::local_id_received\(\)](#)):

```

|-----|-----|-----|---|-----|---|-----|
| hot_spot | order | type | 1 | 11 | c | send flexpage |
|_-----|_-----|_-----|_--|_-----|_--|_-----|

```

4.3 Capabilities and Naming

The [L4Re](#) system is a capability based system which uses and offers capabilities to implement fine-grained access control.

Generally, owning a capability means to be allowed to communicate with the object the capability points to. All user-visible kernel objects, such as tasks, threads, and IRQs, can only be accessed through a capability. Please refer to the [Kernel Objects](#) documentation for details. Capabilities are stored in per-task capability tables (the object space) and are referenced by capability selectors or object flexpages. In a simplified view, a capability selector is a natural number indexing into the capability table of the current task.

As a matter of fact, a system designed solely based on capabilities uses so-called 'local names' because each task can only access those objects made available to this task. Other objects are not visible to and accessible by the task.

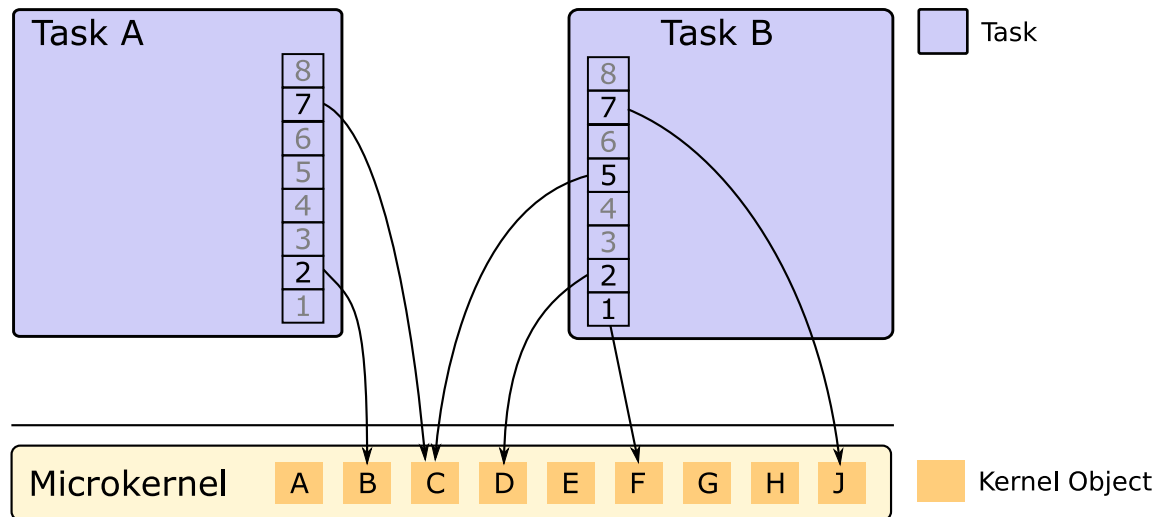


Figure 4.1 Capabilities and Local Naming in L4

So how does an application get access to a service? In general all applications are started with an initial set of available objects. This set of objects is predetermined by the creator of a new application process and granted directly to the new task before starting the first application thread. The application can then use these initial objects to request access to further objects or to transfer capabilities to its own objects to other applications. A central L4Re object for exchanging capabilities at runtime is the name-space object, implementing a store of named capabilities.

From a security perspective, the set of initial capabilities (access rights to objects) completely define the execution environment of an application. Mandatory security policies can be defined by well known properties of the initial objects and carefully handled access rights to them.

4.4 Spaces and Mappings

Each task in the L4Re system has access to two resource spaces (three on IA32) which are maintained by the kernel. These are the

1. object space,
2. memory space, and
3. IO-port space (only on IA32).

The entities addressed in each space are capabilities to objects, virtual memory pages, and IO ports. The addresses are unsigned integers and the largest valid address depends on which space is referenced, the hardware, and the configuration of the kernel. Although a program can access memory at byte granularity, from the kernel's point of

view the address granularity in the memory space is not bytes but pages, as determined by the hardware. The address of a capability is also called its "capability slot".

Flexpages describe a range in any of the spaces that has a power-of-two length and is also aligned to this length. They additionally hold access rights information and further space specific information.

When a resource is present at some address in a task's corresponding resource space, then we say that resource is mapped to that task. For example, a capability to the task's main thread may be mapped to capability slot 5, or the first page of the code segment a thread executes is mapped to virtual memory page 12345. However, there need not be any resource mapped to an address.

Tasks can exchange resources through a process called "mapping" during IPC and using the `L4::Task::map()` method. The sending task specifies a send flexpage and the receiving task a receive flexpage. The resources mapped to the send flexpage will then be mapped to the receive flexpage by the kernel.

Memory mappings and IO port mappings are hierarchical: If a resource of such a type is subject of a map operation, the received mapping is a child mapping of the corresponding mapping in the sending task (parent mapping). The kernel usually respects the relationship between these two mappings (granting is an exception; see below): If rights of a parent mapping are revoked using `L4::Task::unmap()`, these rights are also removed from its child mappings. Also, if a mapping is completely removed (via `L4::Task::unmap()` or by mapping something else at its place), then also all child mappings are removed. In contrast, revoking rights of a child mapping leaves the rights of its parent mapping untouched.

The mapping of a resource can be performed as *grant* operation (see `L4_MAP_ITEM_GRANT`): Such an operation includes the removal of all involved mappings from the send flexpage (basically a move operation). While with a map operation without grant the mapping in the send flexpage remains the parent of all child mappings (including the new child mapping in the receive flexpage), a grant operation moves the mappings covered by the send flexpage to the corresponding addresses from the receive flexpage while leaving the parent/child relationship of the moved mappings with other mappings untouched.

During a map operation at most the access rights of the source mapping(s) can be transferred but no additional rights can be added. So only rights that are present in the source mapping and that are specified in the send item/flexpage are transferred. This also holds for grant mappings, however, rights revocation is *not* guaranteed to be applied to descendant mappings in case of grant.

There are cases where a grant operation is not or cannot be performed as requested; see `L4_MAP_ITEM_GRANT` for details.

Object capabilities are not hierarchical – they have no children. The result of the map operation on an object capability is a copy of that capability in the object space of the destination task.

4.5 Initial Environment and Application Bootstrapping

New applications that are started by a loader conforming to `L4Re` get provided an *Initial Environment*. This environment comprises a set of capabilities to initial `L4Re` objects that are required to bootstrap and run this application. These capabilities include:

- A capability to an initial memory allocator for obtaining memory in the form of data spaces
- A capability to a factory which can be used to create additional kernel objects
- A capability to a Vcon object for debugging output and maybe input
- A set of named capabilities to application specific objects

During the bootstrapping of the application, the loader establishes data spaces for each individual region in the ELF binary. These include data spaces for the code and data sections, and a data space backed with RAM for the stack of the program's first thread.

One loader implementation is the `moe` root task. Moe usually starts an *init* process that is responsible for coordinating the further boot process. The default *init* process is `ned`, which implements a script-based configuration and startup of other processes. Ned uses Lua (<http://www.lua.org>) as its scripting language, see [Ned Script example](#) for more details.

4.5.1 Configuring an application before startup

The default L4Re init process (Ned) provides a Lua script based configuration of initial capabilities and application startup. Ned itself also has a set of initial objects available that can be used to create the environment for an application. The most important object is a kernel object factory that allows creation of kernel objects such as IPC gates (communication channels), tasks, threads, etc. Ned uses Lua tables (associative arrays) to represent sets of capabilities that shall be granted to application processes.

```
local caps = {
    name = some_capability
}
```

The L4 Lua package in Ned also has support functions to create application tasks, region-map objects, etc. to start an ELF binary in a new task. The package also contains Lua bindings for basic L4Re objects, for example, to generic factory objects, which are used to create kernel objects and also user-level objects provided by user-level servers.

```
L4.default_loader:start({ caps = { some_service = service } }, "rom/program --arg");
```

4.5.2 Connecting clients and servers

In general, a connection between a client and a server is represented by a communication channel (IPC gate) that is available to both of them. You can see the simplest connection between a client and a server in the following example.

```
local loader = L4.default_loader; -- which is Moe
local svc = loader:new_channel(); -- create an IPC gate
loader:start({ caps = { service = svc:svr() } }, "rom/my_server");
loader:start({ caps = { service = svc:m("rw") } }, "rom/my_client");
```

As you can see in the snippet, the first action is to create a new channel (IPC gate) using `loader:new_channel()`. The capability to the gate is stored in the variable `svc`. Then the binary `my_server` is started in a new task, and full (`svr()`) access to the IPC gate is granted to the server as initial object. The gate is accessible to the server application as "service" in the set of its initial capabilities. Virtually in parallel a second task, running the client application, is started and also given access to the IPC gate with less rights (`m("rw")`), note, this is essential). The server can now receive messages via the IPC gate and provide some service and the client can call operations on the IPC gate to communicate with the server.

Services that keep client specific state need to implement per-client server objects. Usually it is the responsibility of some authority (e.g., Ned) to request such an object from the service via a generic factory object that the service provides initially.

```
local loader = L4.default_loader; -- which is Moe
local svc = loader:new_channel():m("rws"); -- create an IPC gate with rws rights
loader:start({ caps = { service = svc:svr() } }, "rom/my-service");
loader:start({ caps = { foo_service = svc:create(object_to_create, "param") } }, "rom/client");
```

This example is quite similar to the first one, however, the difference is that Ned itself calls the create method on the factory object provided by the server and passes the returned capability of that request as "foo_service" to the client process.

Note

The `svc:create(...)` call blocks on the server. This means the script execution blocks until the my-service application handles the create request.

4.6 Memory management - Data Spaces and the Region Map

4.6.1 User-level paging

Memory management in L4-based systems is done by user-level applications, the role is usually called *pager*. Tasks can give other tasks full or restricted access rights to parts of their own memory. The kernel offers means to give access to memory in a secure way, often referred to as *memory mapping*.

The mapping mechanism allows one task to resolve page faults of another: A thread usually has a pager assigned to it. When the thread causes a page fault, the kernel sends an IPC message to the pager with information about the page fault. The pager answers this IPC by either providing a backing page, or with an error. The kernel will map the backing page into the address space of the faulting thread's task.

These mechanisms can be used to construct a memory and paging hierarchy among tasks. The root of the hierarchy is `sigma0`, which initially gets all system resources and hands them out once on a first-come-first-served basis. Memory resources can be mapped between tasks at a page-size granularity. This size is predetermined by the CPU's memory management unit and is commonly set to 4 kB.

4.6.1.1 Data spaces

A data space is the [L4Re](#) abstraction for objects which may be accessed in a memory mapped fashion (i.e., using normal memory read and write instructions). Examples include the sections of a binary which the loader attaches to the application's address space, files in the ROM or on disk provided by a file server, the registers of memory-mapped devices and anonymous memory such as the heap or the stack.

Anonymous memory data spaces in particular (but in general all data spaces except memory mapped IO) can either be constructed entirely from a portion of the RAM or the current working set may be multiplexed on some portion of the RAM. In the first case it is possible to eagerly insert all pages (more precisely page-frame capabilities) into the application's address space such that no further page faults occur when this data space is accessed. In general, however, only the pages for some portion are provided and further pages are inserted by the pager as a result of page faults.

4.6.1.2 Virtual Memory Handling

The virtual memory of each task is constructed from data spaces backing virtual memory regions (VMRs). The management of the VMRs is provided by an object called *region map*. A dedicated region-map object is associated with each task; it allows attaching and detaching data spaces to an address space as well as reserving areas of virtual memory. Since the region-map object possesses all knowledge about the virtual memory layout of a task, it also serves as an application's default pager.

4.6.1.3 Memory Allocation

Operating systems commonly use anonymous memory for implementing dynamic memory allocation (e.g., using `malloc` or `new`). In an L4Re-based system, each task gets assigned a memory allocator providing anonymous memory using data spaces.

See also

[L4Re::Dataspace](#) and [L4Re::Rm](#).

4.7 Program Input and Output

The initial environment provides a Vcon capability used as the standard input/output stream. Output is usually connected to the parent of the program and displayed as debugging output. The standard output is also used as a back end to the C-style printf functions and the C++ streams.

Vcon services are implemented in Moe and the loader as well as by the [L4Re](#) Microkernel and connected either to the serial line or to the screen if available.

See also

[Virtual Console](#)

4.8 Initial Memory Allocator and Factory

The purpose of the memory allocator and of the factory is to provide the application with the means to allocate memory (in the form of data spaces) and kernel objects respectively. An initial memory allocator and an initial factory are accessible via the initial [L4Re](#) environment.

See also

[L4Re::Mem_alloc](#)

The factory is a kernel object that provides the ability to create new kernel objects dynamically. A factory imposes a resource limit for kernel memory, and is thus a means to prevent denial of service attacks on kernel resources. A factory can also be used to create new factory objects.

See also

[Factory](#)

4.9 Application and Server Building Blocks

So far we have discussed the environment of applications in which a single thread runs and which may invoke services provided through their initial objects. In the following we describe some building blocks to extend the application in various dimensions and to eventually implement a server which implements user-level objects that may in turn be accessed by other applications and servers.

4.9.1 Creating Additional Application Threads

To create application threads, one must allocate a stack on which this thread may execute, create a thread kernel object and setup the information required at startup time (instruction pointer, stack pointer, etc.). In [L4Re](#) this functionality is encapsulated in the pthread library.

4.9.2 Providing a Service

In capability systems, services are typically provided by transferring a capability to those applications that are authorised to access the object to which the capability refers to.

Let us discuss an example to illustrate how two parties can communicate with each other: Assume a simple file server, which implements an interface for accessing individual files: `read(pos, buf, length)` and `write(pos, data, length)`.

L4Re provides support for building servers based on the class `L4::Server_object`. `L4::Server_object` provides an abstract interface to be used with the `L4::Server` class. Specific server objects such as, in our case, files inherit from `L4::Server_object`. Let us call this class `File_object`. When invoked upon receiving a message, the `L4::Server` will automatically identify the corresponding server object based on the capability that has been provided to its clients and invoke this object's `dispatch` function with the incoming message as a parameter. Based on this message, the server must then decide which of the protocols it implements was invoked (if any). Usually, it will evaluate a protocol specific opcode that clients are required to transmit as one of the first words in the message. For example, assume our server assigns the following opcodes: `Read = 0` and `Write = 1`. The `dispatch` function calls the corresponding server function (i.e., `File_object::read()` or `File_object::write()`), which will in turn parse additional parameters given to the function. In our case, this would be the position and the amount of data to be read or written. In case the write function was called the server will now update the contents of the file with the data supplied. In case of a read it will store the requested part of the file in the message buffer. A reply to the client finishes the client request.

4.10 Pthread Support

L4Re supports the standard pthread library functionality. Therefore L4Re itself does not contain any documentation for pthreads itself. Please refer to the standard pthread documentation instead.

The L4Re specific parts will be described herein.

- Include pthread-l4.h header file:

```
#include <pthread-l4.h>
```

- Return the local thread capability of a pthread thread:

Use `pthread_l4_cap(pthread_t t)` to get the capability index of the pthread `t`.

For example:

```
pthread_l4_cap(pthread_self());
```

- Setting the L4 priority of an L4 thread works with a special scheduling policy (other policies do not affect the L4 thread priority):

```
pthread_t t;
pthread_attr_t a;
struct sched_param sp;

pthread_attr_init(&a);
sp.sched_priority = l4_priority;
pthread_attr_setschedpolicy(&a, SCHED_L4);
pthread_attr_setschedparam(&a, &sp);
pthread_attr_setinheritsched(&a, PTHREAD_EXPLICIT_SCHED);

if (pthread_create(&t, &a, pthread_func, NULL))
    // failure...

pthread_attr_destroy(&a);
```

- You can prevent your pthread from running immediately after the call to `pthread_create(..)` by adding `PTHREAD_L4_ATTR_NO_START` to the `create_flags` of the pthread attributes. To finally start the thread you need to call `scheduler()->run_thread()` passing the capability of the pthread and scheduling parameters.

```
pthread_t t;
pthread_attr_t attr;

pthread_attr_init(&attr);
attr.create_flags |= PTHREAD_L4_ATTR_NO_START;

if (pthread_create(&t, &attr, pthread_func, nullptr))
    // failure...

pthread_attr_destroy(&attr);

// do stuff

auto ret = L4Re::Env::env()->scheduler()->run_thread(pthread_l4_cap(t),
                                                    l4_sched_param(2));
if (l4_error(ret))
    // failure...
```

Constraints on pthread_t, user-land capability slot, and kernel thread-object

- `pthread_l4_cap()` is guaranteed to return the valid capability slot of the pthread (A) until `pthread_join()` or `pthread_detach()` is invoked on (A)'s `pthread_t`.
- `pthread_l4_cap()` exposes internal state of the pthread management, take the necessary precautions as you would for any shared data in concurrent environments. If you use `pthread_l4_cap()` guarding against concurrency issues is your duty.
- There is no guarantee that a valid capability slot points to a present capability.

• Example

It is possible to obtain a valid thread capability slot and for `l4_task_cap_valid()` to return the capability as not present. The following example showcases a possible sequence of events.

```
// Assume: void some_func(void *)
pthread_t pthread = nullptr;
pthread_create(&pthread, nullptr, some_func, nullptr);

// pthread running some_func()
l4_cap_idx_t cap_idx = pthread_l4_cap(pthread);
l4_is_valid_cap(cap_idx); // ---> true

long valid = l4_task_cap_valid(L4RE_THIS_TASK_CAP, cap_idx).label();
// valid == 1 --> capability object is present (refers to a kernel object).

// some_func() exits

cap_idx = pthread_l4_cap(pthread);
l4_is_valid_cap(cap_idx); // ---> true

valid = l4_task_cap_valid(L4RE_THIS_TASK_CAP, cap_idx).label();
// valid == 0 --> capability object is not present (refers to no kernel object).

pthread_join(pthread, nullptr); // invalidates the cap slot and frees
                                // the pthread's data structures

// using cap_idx here is undefined behavior.
```

4.11 Interface Definition Language

An interface definition in **L4Re** is normally declared as a class derived from `L4::Kobject_t`. For example, the [simple calculator example](#) declares its interface like that:

```
struct Calc : L4::Kobject_t<Calc, L4::Kobject>
{
    L4_INLINE_RPC(long, sub, (l4_uint32_t a, l4_uint32_t b, l4_uint32_t *res));
```

```
L4_INLINE_RPC(long, neg, (l4_uint32_t a, l4_uint32_t *res));

typedef L4::Typeid::Rpc<sub_t, neg_t> Rpc<sub_t, neg_t>;
```

The signature of each function is first declared using one of the RPC macros (see below) and then all the functions need to be listed in the `Rpcs` type.

Clients invoke these functions with the name given to the RPC macros, `sub` and `neg` above. Servers implement them by defining functions with an `op_` prepended, `op_sub` and `op_neg`. The types of the parameters in the macro definition, on the server side, and on the client side are not the same. The following section describes how they are related to each other.

4.11.1 Parameter types for RPC

Generally all value parameters, const reference parameters, and const pointer parameters to an RPC interface are considered as input parameters for the RPC and are transmitted from the client to the server.

Note

This means that `char const *` is treated as an input `char` and not as a zero terminated string value, for strings see `L4::lpc::String<>`.

Parameters that are non-const references or non-const pointers are treated as output parameters going from the server to the client.

There are special data types that appear on only one side (client or server) when used, see the following table for details.

```
L4_RPC(long, test, (int arg1, char const *arg2, unsigned *ret1));
```

The example shows the declaration of a method called `test` with `long` as return type, `arg1` is an `int` input, `arg2` a `char` input, and `ret1` an `unsigned` output parameter.

Type	Direction	Client-Type	Server-Type
T	Input	T	T
T const &	Input	T const &	T const &
T const *	Input	T const *	T const &
T &	Output	T &	T &
T *	Output	T *	T &
L4::Ipc::In_out<T &>	In/Out	T &	T &
L4::Ipc::In_out<T *>	In/Out	T *	T &
L4::Ipc::Cap<T>	Input	L4::Ipc::Cap<T>	L4::Ipc::Snd_fpage
L4::Ipc::Out<L4::Cap<T>>	Output	L4::Cap<T>	L4::Ipc::Cap<T> &
L4::Ipc::Rcv_fpage	Input	L4::Ipc::Rcv_fpage	void
L4::Ipc::Small_buf	Input	L4::Ipc::Small_buf	void

Array types can be used to transmit arrays of variable length. They can either be stored in a client-provided buffer ([L4::lpc::Array](#)), copied into a server-provided buffer ([L4::lpc::Array_in_buf](#)) or directly read and written into the UTCB ([L4::lpc::Array_ref](#)).

Constraints on [L4::lpc::Array_ref](#):

- the start position of this array type needs to be known in advance.
- it must be the last parameter of a message.
- the size of the array type is not transmitted to the server, only the client side knows the intended size of the input array. The server assumes the rest of the UTCB as the actual array. Different sever-side behavior must be steered otherwise, e.g. through another parameter.

Type	Direction	Client-Type	Server-Type
<code>L4::Ipc::Array<const T></code>	Input	<code>L4::Ipc::Array<const T></code>	<code>L4::Ipc::Array_ref<const T></code>
<code>L4::Ipc::Array<const T></code>	Input	<code>L4::Ipc::Array<const T></code>	<code>L4::Ipc::Array_in_buf<const T></code>
<code>L4::Ipc::Array<T> &</code>	Output	<code>L4::Ipc::Array<T> &</code>	<code>L4::Ipc::Array_ref<T> &</code>
<code>L4::Ipc::Array_ref<T> &</code>	Output	<code>L4::Ipc::Array_ref<T> &</code>	<code>L4::Ipc::Array_ref<T> &</code>

Finally, there are some optional types where the sender can choose if the parameter should be included in the message. These types are for the implementation of some legacy message formats and should generally not be needed for the definition of ordinary interfaces.

Type	Direction	Client-Type	Server-Type
<code>L4::Ipc::Opt<T></code>	Input	<code>L4::Ipc::Opt<T></code>	<code>T</code>
<code>L4::Ipc::Opt<const T*></code>	Input	<code>L4::Ipc::Opt<const T*></code>	<code>T</code>
<code>L4::Ipc::Opt<T &></code>	Output	<code>T &</code>	<code>L4::Ipc::Opt<T> &</code>
<code>L4::Ipc::Opt<T *></code>	Output	<code>T *</code>	<code>L4::Ipc::Opt<T> &</code>
<code>L4::Ipc::Opt<Array↔_ref<T> &></code>	Output	<code>Array_ref<T> &</code>	<code>L4::Ipc::Opt<Array↔_ref<T>> &</code>

4.11.2 Server Side Interface

The server side function signature for the Calc example above is

```
long op_sub(Calc::Rights, 14_uint32_t a, 14_uint32_t b, 14_uint32_t &res);
```

The first rights parameter is a bitfield encoding the rights the client has on the used capability in the lower two bits. Currently, the W-right and S-right are supported. The rest of the Rights-bitfield is reserved.

The second and third arguments are input parameters as can be deduced from the tables above. The fourth parameter is an output parameter, delivered to the client with the return value of type long.

4.11.3 RPC Return Types

On the server side, the return type of an RPC handling function is always long. The return value is transmitted via the label field in `l4_msgtag_t` and is therefore restricted to its length. Per convention, a negative return value is interpreted as an error condition. If the return value is negative, output parameters are not transmitted back to the client.

Attention

The client must never rely on the content of output parameters when the return value is negative.

On the client-side, the return value of the RPC is set as defined in the RPC macro. If `l4_msgtag_t` is given, then the client has access to the full message tag, otherwise the return type should be `long`. Note that the client might not only receive the server return value in response but also an IPC error code.

4.11.4 RPC Method Declaration

RPC member functions can be declared using one of the following C++ macros.

For inline RPC stubs, where the RPC stub code itself is `inline`:

- `L4_INLINE_RPC(res, name, (args...), flags)`
Define an inline RPC call (type and callable).
- `L4_INLINE_RPC_OP(op, res, name, (args...), flags)`
Define an inline RPC call with specific opcode (type and callable).
- `L4_INLINE_RPC_NF(res, name, (args...), flags)`
Define an inline RPC call type (the type only, no callable).
- `L4_INLINE_RPC_NF_OP(opcode, Ret_type, func_name, (args...), flags)`
Define an inline RPC call type with specific opcode (the type only, no callable).

For external RPC stubs, where the RPC stub code must be defined in a separate compile unit (usually a `.cc` file):

- `L4_RPC(Ret_type, func_name, (args...), flags)`
Define an RPC call (type and callable).
- `L4_RPC_OP(opcode, Ret_type, func_name, (args...), flags)`
Define an RPC call with specific opcode (type and callable).
- `L4_RPC_NF(Ret_type, func_name, (args...), flags)`
Define an RPC call type (the type only, no callable).
- `L4_RPC_NF_OP(opcode, Ret_type, func_name, (args...), flags)`
Define an RPC call type with specific opcode (the type only, no callable).

To generate the implementation of an external RPC stub:

- `L4_RPC_DEF(class_name::func_name)`
Generate the definition of an RPC stub.

The NF versions of the macro generally do not generate a callable member function named `<name>` but do only generate the type `<name>_t`. This data type can be used to call the RPC stub explicitly using `<name>_t::call(L4::Cap<Iface_class> cap, args...)`.

4.12 L4Re Build System

L4Re uses a custom make-based build system, often simply referred to as *BID*. This section explains how to use BID when writing applications and libraries for L4Re.

4.12.1 Building L4Re

Setting up the Build Directory

L4Re must be built out-of-source. Therefore the first mandatory step is creating and populating a build directory. From the root of the L4Re source tree run

```
make B=<builddir>
```

Other targets that can be executed in the source directory are

update

Update the source directory from svn. Only makes sense when you have downloaded L4Re from the official subversion repository.

help

Show a short help with the most important targets.

Invoking Make

Once the build directory is set up, BID make can be invoked in one of two ways:

1. Go to the build directory and invoke make without special options.
2. Go to a source directory with a BID make file and invoke `make O=<builddir> . . .`

The default target builds the source (as you would expect), other targets that are available in build mode are

cleanfast

Quickly cleans the build directory by removing all subdirectories that contain generated files. The configuration will remain untouched.

clean

Remove generated files. Slower than `make cleanfast` but can be used on selected packages. Use `S= . . .` to select the target package.

In addition to these targets, there are a number of targets to generate images which are explained elsewhere.

4.12.2 Writing BID Make Files

The BID build system exports different roles that define what should be done in the subdirectory. So a BID make file essentially consists of defining the role and a number of role-dependent make variables. The basic layout should look like this:

```
PKGDIR  ?= <path to package's root directory> # e.g., '.' or '..'
L4DIR   ?= <path to L4Re source directory>    # e.g. '$(PKGDIR)/../..'

<various definitions>

include $(L4DIR)/mk/<role>.mk
```

PKGDIR in the first line defines the root directory of the current package. L4DIR in the next line must be pointed to the root of the [L4Re](#) source tree the package should be built against. After this custom variable definitions for the role follow. In the final line of the file, the make file with the role-specific rules must be sourced.

The following roles are currently defined:

- project.mk - Sub-project Role
- subdir.mk - Directory Role
- [prog.mk - Application Role](#)
- lib.mk - Library Role
- [include.mk - Header File Role](#)
- doc.mk - Documentation Role
- [test.mk - Test Application Role](#)
- idl.mk - IDL File Role (currently unused)
- runux.mk - Tests in FiascoUX Role

BID-global Variables

This section lists variables that configure how the BID build system behaves. They are applicable for all roles.

Variable	Description
CC	C compiler for target
CXX	C++ compiler for target
HOST_CC	C compiler for host
HOST_CXX	C++ compiler for host

4.12.3 prog.mk - Application Role

The prog role is used to build executable programs.

General Configuration Variables

The following variables can only be set globally for the Makefile:

MODE

Kind of target to build for. The following values are possible:

- `static` - build a statically linked binary (default)
- `shared` - build a dynamically linked binary
- `l4linux` - build a binary for running on L4Linux on the target platform
- `host` - build for host system
- `targetsys` - build a binary for the target platform with the compiler's default settings

SYSTEMS

List of architectures the target can be built for. The entries must be space-separated entries either naming an architecture (e.g. `amd64`) or an architecture and ABI (e.g. `arm-l4f`). When not defined, the target will be built for all possible platforms.

TARGET

Name or names of the binaries to compile. This variable may also be postfixed with a specific architecture.

Target-specific Configuration Variables

The following variables may either be used with or without a description suffix. Without suffix they will be used for all operations. With a specific description their use is restricted to a subset. These specifications include a target file and the architecture, both optional but in this order, separated by underscores. The specific variables will be used in addition to the more general ones.

`SRC_C / SRC_CC / SRC_F / SRC_S`

`.c, .cc, .f90, .S` source files.

`REQUIRES_LIBS`

List of libraries the binary depends on. This works only with libraries that export a `pkg_config` configuration file. Automatically adds any required include and link options.

DEPENDS_PKGS

List of packages this binary depends on. If one these packages is missing then building of the binary will be skipped.

CPPFLAGS / CFLAGS / CXXFLAGS / FFLAGS / ASFLAGS

Options for the C preprocessor, C compiler, C++ compiler, Fortran compiler and assembler. When used with suffix, the referred element is the source file, not the target file.

LDFLAGS

Options for the linker ld.

LIBS

Additional libraries to link against (with -l).

PRIVATE_LIBDIR

Additional directories to search for libraries.

CRT0 / CRTN

(expert use only) Files containing custom startup and finish code.

LDSCRIPT

(expert use only) Custom link script to use.

4.12.4 include.mk - Header File Role

The header file role is responsible for installing header file at the appropriate location. The following variables can be used for customizing the process:

INCSRC_DIR

Source directory where the headers can be found. Default is the directory where the Makefile resides.

TARGET

List of header files to install. If left undefined, then INCSRC_DIR will be scanned for files with suffix .h or .i.

Supports the specification of special filenames to allow for different source and target filenames to be installed. The syntax is TARGET<SRC, where a filename including the path of SRC is installed as TARGET. An example is

```
libfoo.h<contrib/libfoo_linux.h
```

which installs the header from the contrib directory under the name without that contrib directory and without the platform specific suffix.

EXTRA_TARGET

When TARGET is undefined, then add these files to the headers found by scanning the source directory. Has no effect if TARGET has been defined.

The filenames specified allow for the same rule specifications as supported by TARGET.

CONTRIB_HEADERS

When set, the headers will be installed in \${BUILDDIR}/include/contrib/\${PKGNAME} rather than \${BUILDDIR}/include/l4/\${PKGNAME}.

INSTALL_INC_PREFIX

Base directory where to install the headers. Overwrites CONTRIB_HEADERS. The headers will then be found under \${BUILDDIR}/include/\${INSTALL_INC_PREFIX}.

PC_FILENAME

When set, a pkg_config configuration file is created with the given name.

4.12.5 test.mk - Test Application Role

The test role is very similar to the application role, it also builds an executable binary. The difference is that it also builds for each target a test script that executes the test target either on the host (MODE=host) or a target platform (currently only qemu).

The role accepts all make variables that are accepted by the application role. The only difference is that the `TARGET` variable is not required. If it is missing, the source directory will be scanned for source files that fit the pattern `test_*.c[c]` and create one target for each of them.

Note

It is possible to still use `SRC_C[C]` when targets are determined automatically. In that case the specified sources will be used in addition* to the main `test_*.c[c]` source.

In addition to the variables above, there are a number of variables that control how the test is executed. All these variables may be used as a global variable that applies to all test or, if the target name is added as a suffix, set for a specific target only.

TEST_TARGET

Name of binary containing the test (default: same as `TARGET`).

TARGET_\$ (ARCH)

When `TARGET` is undefined, these targets are added to the list of targets for the specified architecture. For all targets `SRC_C[C]` files must be defined separately.

TEST_KERNEL_ARGS

Arguments to append to the kernel command line. These are also appended when specifying custom ones via a .t-file's `-f` parameter or when using `-d`.

TEST_EXPECTED

File containing expected output. By default the variable is empty, which means the test binary is expected to produce TAP test output, that can be directly processed. When the `TEST_TAP_PLUGINS` variable is given, `TEST_EXPECTED` is ignored.

TEST_EXPECTED_REPEAT

Number of times the expected output should be repeated, by default 1. When set to 0 then output is expected to repeat forever. This is particularly useful to make sure that stress tests that are meant to run in an endless loop are still alive. Note that such endless tests can only be run by directly executing the test script. They will be skipped when run in a test harness like `prove`.

TEST_TAP_PLUGINS

Specify the plugins that are used to process the output of the test run. The syntax is of the values is:

plugin1:arg1=a,arg2=b plugin2:arg=foo

Multiple plugins separated by a space are loaded in order. Spaces are not allowed inside a plugin specification. One or more arguments are optionally passed to the plugin separated by commas and delimited by a colon.

If the variable is not specified the plugins for TAPOutput and OutputMatching (depending on the TEST_EXPECTED variable) are automatically loaded.

For the supported plugins and their options please refer to their in-line documentation in tool/lib/L4/TapWrapper/↵ Plugin/. The plugin name corresponds to the file stem name in that directory.

TEST_TIMEOUT

Non-standard timeout after which the test run is aborted (useful for tests involving sleep).

NED_CFG

LUA configuration file for startup to give to Ned

REQUIRED_MODULES

Additional modules needed to run the test. By adding [opts] to the name of a module you can add module options that are reflected in the generated modules.list.

BOOTSTRAP_ARGS

Additional parameters to supply to bootstrap.

QEMU_ARGS

Additional parameters to supply to QEMU.

MOE_ARGS

Additional parameters to supply to moe.

TEST_ARGS

Additional arguments for the TEST_STARTER (tapper-wrapper per default).

TEST_ROOT_TASK

Alternative root task to be used during a test instead of moe.

TEST_ROOT_TASK_ARGS

Arguments passed to TEST_ROOT_TASK if TEST_ROOT_TASK is different from moe.

`KERNEL_CONF`

Features the [L4Re](#) Microkernel must have been compiled with. A space-separated list of config options as used by Kconfig. `run_test` looks for a `globalconfig.out` file in the same directory as the kernel and checks that all options are enabled. If not, the test is skipped. Has only an effect if the `globalconfig.out` file is present.

`L4RE_CONF`

Features the [L4Re](#) userland must have been compiled with. A space-separated list of config options as used by Kconfig. `run_test` will look for these in the `.kconfig` file in the [L4Re](#) build directory.

`L4LINUX_CONF`

Features the L4Linux kernel must have been compiled with. Similar to `KERNEL_CONF` but checks for a `.config` file in the directory of the L4Linux kernel.

`TEST_SETUP`

Command to execute before the test is run. The test will only be executed if the command returns 0. If the exit code is 69, the test is marked as skipped with the reason provided in the final line of stdout.

`TEST_LOGFILE`

Append output of test execution to the given file unless `TEST_WORKDIR` is given.

`TEST_WORKDIR`

Create logs, temp and other files below the given directory. That directory is taken as base dir for more automatically created subdir levels using the current test path, in order to guarantee conflict-free usage when running many different tests with a common workdir. When `TEST_WORKDIR` is provided then `TEST_LOGFILE` is ignored as it is organized below workdir.

`TEST_TAGS`

List of conditions for tags provided during execution of a test. A tag can be set to 1, set to 0 or be unspecified via `TEST_RUN_TAGS` during execution. Therefore there are 4 possible conditions for a tag that can be specified in `TEST_TAGS`: `tag`, `!tag`, `+tag` and `-tag`. The following table shows the conditions they represent.

<code>TEST_RUN_TAGS \ TEST_TAGS</code>	<code>tag</code>	<code>!tag</code>	<code>+tag</code>	<code>-tag</code>
<code>tag</code> or <code>tag=1</code>	y		y	
unspecified		y	y	
<code>tag=0</code>		y		y

Example usage:

The tag `long-running` is used by tests which take a long time and should be skipped by default. These tests are marked with the tag `long-running` unprefixd.

The tag `hardware` is set to 1 at runtime when the tests will run on real hardware. Tests that must not run on real hardware are marked with `!hardware`.

The tag `+impl-def` is used by tests that test implementation details. Due to the nature of this flag we require the "+" prefix to be used, so they are run by default but can be excluded from execution by setting `TEST_RUN_TAGS` to `impl-def=0` at runtime.

If you want to specify multiple tag conditions they need to be separated with a comma.

TEST_PLATFORM_ALLOW and TEST_PLATFORM_DENY

Deny and allow lists of platforms a test is banned from or limited to. If you list platforms in the `TEST_PLATFORM_ALLOW` variable the test will only be run on these listed platforms and will be skipped on any other platform. If you list platforms in the `TEST_PLATFORM_DENY` variable the test will be skipped on the listed platforms and will be run on any other platform. You can only use one of these variables per test, not both. See `mk/platforms/` for the various identifiers.

Example usage:

```
# Do not run this test on the Raspberry Pi platform
TEST_PLATFORM_DENY_test_xyz := rpi

# Only run this test on this test on the RCar3 platform.
TEST_PLATFORM_ALLOW_test_abc := rcar3
```

TAPARCHIVE

Filename for an archive file to store the resulting TAP output.

In addition to compiled tests, it is also possible to create tests where the test binary or script comes from a different source. These tests must be listed in `EXTRA_TARGET` and for each target a custom `TEST_TARGET` must be provided.

Running Tests

The make role creates a test script which can be found in `<builddir>/test/t/<arch>/<api>`. It is possible to organise the tests further in subdirectories below by specifying a `TEST_GROUP`.

To be able to execute the test, a minimal test environment needs to be set up by exporting the following environment variables:

`KERNEL_<arch>, KERNEL`

L4Re Microkernel binary to use. The test runner is able to check if the kernel has all features necessary for the test and skip tests accordingly. In order for this to work, the `globalconfig.out` config file from the build directory needs to be available in the same directory as the kernel.

`L4LX_KERNEL_<arch>, L4LX_KERNEL`

L4Linux binary to use. This is only required to run tests in `mode=l4linux`. If no L4Linux kernel is set then these tests will simply be skipped. The test runner is also able to check if the kernel has all features compiled in that are required to run the test successfully (see make variable `L4LINUX_CONF` above). For this to work, the `.config` configuration file from the build directory needs to be available in the same directory as the kernel.

`LINUX_RAMDISK_<arch>, LINUX_RAMDISK`

Ramdisk to mount as root in L4Linux. This is only required to run tests in `mode=l4linux`. If not supplied, L4Linux tests will be skipped. The ramdisk must be set up to start the test directly after the initial startup is finished. The name of the test binary is supplied via the kernel command line option `l4re_testprog`. The `tool/test` directory contains an example script `launch-l4linux-test`, which can be copied onto the ramdisk and started by the init script.

TEST_HWCONFIG and TEST_FIASCOCONFIG

Some userland tests rely on external information about the underlying platform and the configuration of the [L4Re](#) Microkernel to decide whether or not to test specific features or to determine which and how much resources are available. Some examples for this are whether or not virtualization is supported by the platform, how many cores the platform has, how many cores the kernel supports or how much memory the platform provides. To convey this information to these tests you can set the two environment variables `TEST_HWCONFIG` and `TEST_FIASCOCONFIG`.

Using `TEST_HWCONFIG` requires a plain text document containing key-value pairs separated by a `=` symbol. On top of that comment lines starting with `#` are supported. Simply create a plain text file such as the following and set `TEST_HWCONFIG` to its absolute path.

```
VIRTUALIZATION=y
MP=y
NUM_CORES=4
MEMORY=2048
```

Using `TEST_FIASCOCONFIG` is easier since it only needs to contain the absolute path of the `globalconfig.out` file in the [L4Re](#) Microkernel's build directory. The build system will then extract the information when a test is started.

When starting a test the build system will read both files and provide their content as a lua table to the test. A test script can then make decisions based on them. To simplify some decisions the build system merges some information by itself, e.g. virtualization is only available if both the platform and the [L4Re](#) Microkernel support this feature. More details can be obtained from the perl module in `tool/lib/L4/TestEnvLua.pm`.

In addition to these variables, the following BID variables can be overwritten at runtime: `PT` (for the platform type) and `TEST_TIMEOUT`. You may also supply `QEMU_ARGS` and `MOE_ARGS` which will be appended to the parameters specified in the BID test make file.

Once the environment is set up, the tests can be run either by simply executing all of them from the build directory with

```
make test
```

or executing them directly, like

```
test/t/amd64_amdfam10/14f/14re-core/moe/test_namespace.t
```

or running one or more tests through the test harness `prove`, like

```
prove test/t/amd64_amdfam10/14f/14re-core/moe/test_namespace.t
prove -r test/t/amd64_amdfam10/14f/14re-core/
prove -rv test/t/amd64_amdfam10/14f/14re-core/
```

`TEST_TAGS` allow for a way to include or exclude whole groups of tests during execution, primarily with `prove`. You can specify which tests to run at runtime using one of the following ways:

```
$ test/t/amd64_amdfam10/14f/14re-core/test_one.t --run-tags slow,gtest-shuffle=0
$ test/t/amd64_amdfam10/14f/14re-core/test_one.t -T slow,gtest-shuffle=0
$ prove -r test/t/amd64_amdfam10/14f/14re-core/ : -T slow,gtest-shuffle=0
$ TEST_RUN_TAGS=slow,gtest-shuffle=0 prove -r test/t/amd64_amdfam10/14f/14re-core/
```

For each test tag requirements defined in the corresponding `TEST_TAGS` Makefile variable are tested. If the requirements for tags do not match the test is skipped. The `SKIP` message will provide insight why the test was skipped:

```
$ make test
...
test/t/amd64_amdfam10/test_one.t .... ok
test/t/amd64_amdfam10/test_two.t .... skipped: Running this test requires tag slow to be set to 1.
test/t/amd64_amdfam10/test_three.t .. ok
```

When tags are provided, the tests requiring those tags are now also executed while the tests that forbid them are skipped:

```
$ TEST_RUN_TAGS=slow,gtest-shuffle
$ make test
...
test/t/amd64_amdfam10/test_one.t .... ok
test/t/amd64_amdfam10/test_two.t .... ok
test/t/amd64_amdfam10/test_three.t .. skipped: Running this test requires tag gtest-shuffle to be set to 0 or
```

For further details on how values in `TEST_TAGS` and `TEST_RUN_TAGS` interact, see the help text for `TEST_TAGS`.

Running Tests in External Programs

You can hand-over test execution to an external program by setting the environment variable `EXTERNAL_TEST_STARTER` to the full path of that program:

```
export EXTERNAL_TEST_STARTER=/path/to/external/test-starter
make test
```

`EXTERNAL_TEST_STARTER`

This variable is evaluated by `tool/bin/run_test` (the backend behind `make test`) and contains the full path to the tool which actually starts the test instead of the test itself.

The `EXTERNAL_TEST_STARTER` can be any program instead of the default execution via `make qemu E=maketest`. Its output is taken by `run_test` as the test output.

Usually it is just a bridge to prepare the test execution, e.g., it could create the test as image and start that image via a simulator.

Running Tests in a Simulator

Based on above mechanism there is a dedicated external test starter `tool/bin/teststarter-image-telnet.pl` shipped in BID which assumes an image to be started with another program which provides test execution output on a network port.

This can be used to execute tests in a simulator, like this:

```
export EXTERNAL_TEST_STARTER=$L4RE_SRC/tool/bin/teststarter-image-telnet.pl
export SIMULATOR_START=/path/to/configured/simulator-exe
make test
```

After building the image and starting the simulator it contacts the simulator via a network port (sometimes called "telnet" port) to pass-through its execution output as its own output so it gets captured by `run_test` as usual.

The following variables control `teststarter-image-telnet.pl`:

SIMULATOR_START

This points to the full path of the program that actually starts the prepared test image. Most often this is the frontend script of your simulator environment which is pre-configured so that it actually works in the way that `teststarter-image-telnet.pl` expects from the following settings.

SIMULATOR_IMAGETYPE

The image type to be generated via `make $SIMULATOR_IMAGETYPE E=maketest`. Default is `elfimage`.

SIMULATOR_HOST

The simulator will be contacted via socket on that host to read its output. Default is `localhost`.

SIMULATOR_PORT

The simulator will be contacted via socket on that port to read its output. Default is `11111`.

SIMULATOR_START_SLEEPTIME

After starting the simulator it waits that many seconds before reading from the port. Default is `1` (second).

Running tests without tapper-wrapper

In case you want to replace the `tapper-wrapper` test starter, you can replace the default one by setting the environment variable `TEST_STARTER` to the path of your test starter. Then your test starter can use the same environment which is normally set up for the default starter, which includes environment variables provided by the build system as well as the test itself. Among these are `SEARCHPATH`, `MODE`, `ARCH`, `MOE_CFG`, `MOE_ARGS`, `TEST_TIMEOUT`, `TEST_TARGET`, `TEST_EXPECTED`, `QEMU_ARGS` and many more.

Debugging Tests

The test script is only a thin wrapper that sets up the test environment as it was defined in the make file and then executes two scripts: `tapper-wrapper` and `run_test`.

The main work horse of the two is `tool/bin/run_test`. It collects the necessary files and starts `qemu` to execute the test. This script is always required.

There is then a second script wrapped around the test runner: `tool/bin/tapper-wrapper`. This tool inspects the output of the test runner and reformats it, so that it can be read by tools like `prove`. If the test produces tap output, then the script scans for this output and filters away all the debug output. If `TEST_EXPECTED` was defined, then the script scans the output for the expected lines and prints a suitable TAP message with success or failure. It also makes sure that `qemu` is killed as soon as the test is finished.

There are a number of command-line parameters that allow to quickly change test parameters for debugging purposes. Run the test with `'-help'` for more information about available parameters.

4.13 Kernel Factory

The kernel factory is a kernel object that provides the ability to create new kernel objects dynamically.

The kernel factory enforces a memory quota. This quota defines the maximum amount of kernel memory the factory service can use to construct the requested objects. When the quota is depleted, the factory refuses the creation of new objects.

The quota may be higher than the amount of available kernel memory; ultimately, the amount of available kernel memory is the strict limit for the factory to remain operational.

The kernel factory creates the following kinds of objects:

- [DMA space](#)
- [L4::Factory](#)
- [L4::lpc_gate](#) ([L4_PROTO_NONE](#), [L4_PROTO_KOBJECT](#))
- [L4::Irq](#) ([L4_PROTO_IRQ_SENDER](#))
- [L4::Semaphore](#)
- [L4::Task](#)
- [L4::Thread](#)
- [L4::Vm](#)
- [L4::Vcpu_context](#)

The protocol IDs for objects in this list are given in [L4_msgtag_protocol](#). Kernel objects whose protocol ID is not immediately clear from the documentation of [L4_msgtag_protocol](#) have their protocol IDs stated within parenthesis. As an exception, [L4::lpc_gate](#) can be identified by more than one protocol IDs. The protocol ID shall be used as the second argument for [L4::Factory.create](#)([Cap<void>](#), long, [l4_utcb_t *](#)).

For the C++ interface see [L4::Factory](#), for the C interface see [Factory](#).

4.13.1 Passing parameters for the create stream

[L4::Factory.create\(\)](#) returns a [create stream](#) that allows arguments to be forwarded to the constructor of the object to be created.

Objects that support additional parameters on their creation are presented with a non-empty list of parameters. The parameters are listed in the order they should be provided to a create stream returned by [L4::Factory.create\(\)](#).

- [Dmar_space\(\)](#)
- [L4::Factory\(l4_umword_t\)](#)
 - Argument: factory quota (in bytes).
 - See [L4::Factory.create_factory\(\)](#) for details.
- [L4::lpc_gate\(\)](#)
 - Creates an unbound IPC gate.
 - Alternatively, an IPC gate can be immediately bound to a thread upon creation using [L4::Factory.create_gate\(\)](#).

- [L4::Irq\(\)](#)
- [L4::Semaphore\(\)](#)
- [L4::Task\(l4_fpage_t\)](#)
 - Argument: utcb_area
 - See [L4::Factory.create_task\(\)](#) for details.
- [L4::Thread\(\)](#)
- [L4::Vm\(\)](#)
- [L4::Vcpu_context\(\)](#)

Chapter 5

L4Re Servers

Here you shall find a quick overview over the standard services running on the [L4Re](#) Microkernel.

Sigma0, the Root Pager

Sigma0 is a special server running on [L4](#) because it is responsible of resolving page faults for the root task, the first useful task on [L4Re](#). Sigma0 can be seen as part of the kernel, however it runs in unprivileged mode. To run something useful on the [L4Re](#) Microkernel you usually need to run Sigma0, nevertheless it is possible to replace Sigma0 by a different implementation.

For more details see [Sigma0, the Root-Pager](#)

Moe, the Root Task

Moe is our implementation of the [L4](#) root task that is responsible for bootstrapping the system, and to provide basic resource management services to the applications on top. Therefore Moe provides [L4Re](#) resource management and multiplexing services:

- **Memory** in the form of memory allocators ([L4Re::Mem_alloc](#), [L4::Factory](#)) and data spaces ([L4Re::Dataspace](#))
- **Cpu** in the form of basic scheduler objects ([L4::Scheduler](#))
- **Vcon** multiplexing for debug output (output only)
- **Virtual memory management** for applications, [L4Re::Rm](#)

Moe further provides an implementation of [L4Re](#) name spaces ([L4Re::Namespace](#)), which are for example used to provide a read only directory of all multi-boot modules. In the case of a boot loader, like grub that enables a VESA frame buffer, there is also a single instance of an [L4Re](#) graphics session ([L4Re::Goos](#)).

To start the system Moe starts a single ELF program, the init process. The init process (usually Ned, see the next section) gets access to all resources managed by Moe and to the Sigma0 root pager interface.

For more details see [Moe, the Root-Task](#).

Ned, the Default Init Process

To keep the root task free from complicated scripting engines and to avoid circular dependencies in application startup (that could lead to dead locks) the configuration and startup of the real system is managed by an extra task, the init process.

Ned is such an init process that allows system configuration via Lua scripts.

For more information see [Ned](#).

Io, the Platform and Device Resource Manager

Because all peripheral management in [L4Re](#) is done in user-level applications, there is the need to have a centralized management of the resources belonging to the platform and to peripheral devices.

This is the job of Io. Io provides portable abstractions for iterating and accessing devices and their resources (IRQ's, IO Memory...), as well as delegating access to those resources to other applications (e.g., device drivers).

For more details see [Io, the Io Server](#).

Other Servers

The following additional server package are available on top of the core [L4Re](#) environment.

- [Rtc](#), the Real-Time Clock Server
is a simple multiplexer for real-time clock hardware on your platform.
- [fb-driv](#), the Low-Level Graphics Driver
provides low-level access and initialization of various graphics hardware. It has support for running VESA BIOS calls on Intel x86 platforms, as well as support for various ARM display controllers. `fb-driv` provides a single instance of the L4Re::Goos interface and can serve as a back end for the Mag server, in particular, if there is no graphics support in the boot loader.
- [l4vio_net_p2p](#), a virtual network point-to-point link
- [l4vio_switch](#), a virtual network switch
- [Uvmm](#), the virtual machine monitor
- [RTC driver](#)
- [NVMe server](#)
- [Mag](#), the GUI Multiplexer
Our default multiplexer for the graphics hardware is Mag. Mag is a Nitpicker (TODO: ref) derivate that allows secure multiplexing of the graphics and input hardware among multiple applications and multiple complete windowing environments.
- [Sigma0](#), the Root-Pager
- [eMMC driver](#)
- [Cons](#), the Console Multiplexer
- [AHCI driver](#)

5.1 Sigma0, the Root-Pager

Sigma0 is a special [L4](#) server that serves as the origin for mapping memory. It is started by Fiasco.OC on the system boot and gets full access to all userland RAM and device memory. It functions as the pager (main memory provider) for Moe and as the provider for device memory for Io. Moe and Io are trusted and usually the only applications besides Ned that get a capability for Sigma0. Memory can be requested from Sigma0 directly via an IPC, or indirectly by causing page faults and having them resolved by Sigma0.

5.1.1 Factory

There is only one instance of Sigma0 in an [L4Re](#) system, which is made accessible to Moe via an IPC gate capability. Using this capability, Moe can request Sigma0 to create new communication channels to itself by creating additional IPC gate capabilities. This request is done using the [L4::Factory](#) interface. This is the only kind of object that can be created by the factory in Sigma0.

List of objects that the Sigma0 Factory can create:

- Sigma0 ()
 - Use protocol id [L4_PROTO_SIGMA0](#) for creation
 - No arguments supported

See also

[Sigma0 API](#)

5.2 Moe, the Root-Task

Moe is the default root-task implementation for L4Re-based systems.

Moe is the first task which is usually started in L4Re-based systems. The micro kernel starts *Moe* as the Root-Task.

5.2.1 Moe objects

Moe provides a default implementation for the basic [L4Re](#) abstractions, such as data spaces ([L4Re::Dataspace](#)), region maps ([L4Re::Rm](#)), memory allocators ([L4::Factory](#), [L4Re::Mem_alloc](#)), name spaces ([L4Re::Namespace](#)) and so on (see [L4Re Interface](#)). These are described in the following subsections.

5.2.1.1 Factory

The factory in Moe is responsible for all kinds of dynamic object allocation.

Moe's factory allows allocation of the following objects:

- [L4Re::Namespace](#)
- [L4Re::Dataspace](#), RAM allocation
- [L4Re::Dma_space](#), memory management for DMA-capable devices
- [L4Re::Rm](#), virtual memory management for application tasks
- [L4::Vcon](#) (output only)
- [L4::Scheduler](#), to provide a restricted priority / CPU range for clients
- [L4::Factory](#), to provide a quota limited allocation for clients

Note

[L4::Scheduler](#) objects can be only created through the user factory provided by Moe to the initial application. Other factory instances cannot create this object.

5.2.1.1.1 Passing parameters to the create stream

[L4::Factory.create\(\)](#) returns a [create stream](#) that allows arguments to be forwarded to the object creation in Moe.

Objects that support additional parameters on their creation are presented next with a non-empty list of parameters. The parameters are listed in the order they should be provided to a create stream. Optional parameters are identified by their default values. Multiple entries in the next list denote different ways of initializing an object.

- [L4Re::Namespace](#) ()
 - For more details see [Namespace](#)
- [L4Re::Dataspace](#) ([l4_mword_t](#) size, [l4_umword_t](#) flags = 0, [l4_umword_t](#) align = 0)
 - Argument size: size in bytes (mandatory)
 - Argument flags: special dataspace properties, see [L4Re::Mem_alloc::Mem_alloc_flags](#)
 - Argument align: Log2 alignment of dataspace if supported by allocator
 - See detailed description of the parameters in [L4Re::Mem_alloc::alloc\(\)](#)
 - For details on the types of dataspace provided by Moe, see [Dataspace](#)
- [L4Re::Dma_space](#) ()
 - For more details see [DMA Space](#)
- [L4Re::Rm](#) ()
 - For more details see [Region Map](#)
- [L4::Vcon](#) (char const *label, [l4_mword_t](#) color = 7)
 - Argument label: label used as prefix for the console output
 - Argument color: color code 0..15
 - For more details see [Log Subsystem](#)
- [L4::Vcon](#) (char const *label, char const *color = "w")
 - Argument label: label used as prefix for the console output
 - Argument color: color code
 - * The color is identified by a single character
 - * Supported colors: N, n, R, r, G, g, Y, y, B, b, M, m, C, c, W, w
 - For more details see [Log Subsystem](#)
- [L4::Scheduler](#) ([l4_mword_t](#) limit, [l4_mword_t](#) offset, [l4_umword_t](#) bitmap = ~0UL, ...)
 - Argument limit: maximum priority
 - Argument offset: priority offset
 - Argument bitmap: bitmap of CPUs - can be repeated to address higher order CPUs
 - Argument limit must be greater than offset
 - For more details see [Scheduler subsystem](#)
- [L4::Factory](#) ([l4_mword_t](#) quota)
 - Argument quota: limit in bytes (not zero)
 - The limit is deducted from the limit of the factory that creates the new factory

5.2.1.2 Namespace

Moe provides a name space conforming to the [L4Re::Namespace](#) interface (see [Name-space API](#)). Per default Moe creates a single name space for the [Boot FS](#). That is available as `rom` in the initial objects of the init process.

5.2.1.2.1 Boot FS

The Boot FS subsystem provides access to the files loaded during the platform boot (or available in ROM). These files are either linked into the boot image or loaded via a flexible boot loader, such as GRUB.

The subsystem provides an [L4Re::Namespace](#) object as directory and an [L4Re::Dataspace](#) object for each file.

By default all files are read only and visible in the namespace `rom`. As an option, files can be supplied with the argument `:rw` to mark them as writable modules. Moe will allow read and write access to these dataspace and make them visible in a different namespace called ``rwfs``.

An example entry in 'modules.list' would look like this:

```
module somemodule :rw
```

Note

In order for a client to receive write permissions to the dataspace, the corresponding cap also needs write permissions.

5.2.1.3 Dataspace

Dataspaces can be allocated with an arbitrary size. The granularity for memory allocation however is the machine page size ([L4_PAGESIZE](#)). A dataspace user must be aware that, as a result of this page-size granularity, there may be padding memory at the end of a dataspace which is accessible to each client. Moe currently allows most dataspace operations on this padding area. Nonetheless, the client must not make any assumptions about the size or content of the padding area, as this behaviour might change in the future.

The provided data spaces can have different characteristics:

- Physically contiguous and pre-allocated
- Non contiguous and on-demand allocated with possible copy on write (COW)

Dataspaces allocated via the Moe's factory allow mappings with any combination of the read-write-execute (RWX) rights, subject to a possible restriction of the writable right for client capabilities lacking the 'W' right.

5.2.1.4 Log Subsystem

The logging facility of Moe provides per application tagged and synchronized log output.

5.2.1.5 DMA Space

5.2.1.6 Scheduler subsystem

The scheduler subsystem provides a simple scheduler proxy for scheduling policy enforcement.

The priority offset provided on the creation of a scheduler proxy defines the minimum priority assigned to threads which are scheduled by that instance of the scheduler proxy. The offset is implicitly added to priorities provided to [L4::Scheduler.run_thread\(\)](#).

5.2.1.7 Region Map

5.2.2 Command Line Options

Moe's command-line syntax is:

```
moe [--debug=<flags>] [--init=<binary>] [--l4re-dbg=<flags>] [--ldr-flags=<flags>] [-- <init options>]
```

```
--debug=<debug flags>
```

This option enables debug messages from Moe itself, the `<debug flags>` values are a combination of `info`, `warn`, `boot`, `server`, `loader`, `exceptions`, and `ns` (or all for full verbosity).

```
--init=<init process>
```

This options allows to override the default init process binary, which is 'rom/ned'.

```
--l4re-dbg=<debug flags>
```

This option allows to set the debug options for the [L4Re](#) runtime environment of the init process. The flags are the same as for `--debug=`.

```
--ldr-flags=<loader flags>
```

This option allows setting some loader options for the [L4Re](#) runtime environment. The flags are `pre_alloc`, `all_segs_cow`, and `pinned_segs`.

```
--brk=<address>
```

This option is only present on systems without MMU. It restricts dynamic allocations to addresses equal or higher than `<address>`. The argument is parsed as hexadecimal number without any `0x` prefix. Use it to prevent moe from allocating memory in regions that shall later be used by other applications or virtual machines.

```
-- <init options>
```

All command-line parameters after the special `--` option are passed directly to the init process.

5.3 Ned, the Init Process

Ned's job is to bootstrap the system running on [L4Re](#). The main thing to do here is to coordinate the startup of services and applications as well as to provide the communication channels for them. The central facility in Ned is the Lua (<http://www.lua.org>) script interpreter with the [L4Re](#) and ELF-loader bindings.

The boot process is based on the execution of one or more Lua scripts that create communication channels (IPC gates), instantiate other [L4Re](#) objects, organize capabilities to these objects in sets, and start application processes with access to those objects (or based on those objects).

For starting applications, Ned depends on the services of [Moe](#), the [Root-Task](#) or another *loader*, which must provide data spaces and region maps. Ned also uses the 'rom' capability as source for Lua scripts and at least the 'l4re' binary (the runtime environment core) running in each application.

Each application Ned starts is equipped with an [L4Re::Env](#) environment that provides information about all the initial objects made accessible to this application.

5.3.1 Lua Bindings for L4Re

Ned provides various bindings for [L4Re](#) abstractions. These bindings are located in the 'L4' package (`require "L4"`).

5.3.1.1 Tutorial

For a verbose example using the Ned Lua bindings, see [tutorial.lua](#).

5.3.1.2 Capabilities in Lua

Capabilities are handled as normal values in Lua. They can be stored in normal variables or Lua compound structures (tables). A capability in Lua possesses additional information about the access rights that shall be transferred to other tasks when the capability is transferred. To support implementation of the Principle of Least Privilege, minimal rights are assigned by default. Extended rights can be added using the method `mode("...")` (short `m("...")`) that returns a new reference to the capability with the given rights.

Note

It is generally impossible to elevate the real access rights to an object. This means that if Ned has only restricted rights to an object it is not possible to upgrade the access rights with the `mode` method.

The capabilities in Lua also carry dynamic type information about the referenced objects. They thereby provide type-specific operations on the objects, such as the `create` operation on a generic factory or the `query` and `register` operations on a name space.

5.3.1.3 Access to L4Re::Env Capabilities

The initial objects provided to Ned itself are accessible via the table `L4.Env`. The default (usually unnamed) capabilities are accessible as `factory`, `log`, `mem_alloc`, `parent`, `rm`, and `scheduler` in the `L4.Env` table.

5.3.1.4 Constants

Protocols

The protocol constants are defined by default in the [L4](#) package's table `L4.Proto`. The definition is not complete and only covers what is usually needed to configure and start applications. The protocols are for example used as first argument to the `Factory:create` method.

```
Proto = {
  Dataspace = 0x4000,
  Namespace = 0x4001,
  Goos      = 0x4003,
  Mem_alloc = 0x4004,
  Rm        = 0x4005,
  Event     = 0x4006,
  Inhibitor = 0x4007,
  Sigma0    = -6,
  Log       = -13,
  Scheduler = -14,
  Factory   = -15,
  Vm        = -16,
  Dma_space = -17,
  Irq_sender = -18,
  Semaphore = -20,
```

```

Iommu      = -22,
Ipc_gate   = 0,
}

```

Rights

The rights of a Lua capability can be defined in two ways via the `:mode()` interface. Either via a string representation of the rights or via an integer value. An example for the former is `:mode("rsnc")` while the latter equivalent is `:mode(L4.Rights.r | L4.Rights.s | L4.Rights.n | L4.Rights.c)`. The following listing shows the integer constants. The constant names can be used in the string parameter to `:mode()`.

```

Rights = {
  s = 2,
  w = 1,
  r = 4,
  d = 8,
  n = 16,
  c = 32,
  ro = 4,
  rw = 5,
  rws = 7,
}

```

Debugging Flags

Debugging flags used for the applications [L4Re](#) core:

```

Dbg = {
  Info      = 1,
  Warn      = 2,
  Boot      = 4,
  Server    = 0x10,
  Exceptions = 0x20,
  Cmd_line  = 0x40,
  Loader    = 0x80,
  Name_space = 0x400,
  All       = 0xffffffff,
}

```

Loader Flags

Flags for configuring the loading process of an application.

```

Ldr_flags = {
  eager_map      = 0x1, -- L4RE_AUX_LDR_FLAG_EAGER_MAP
  all_segs_cow  = 0x2, -- L4RE_AUX_LDR_FLAG_ALL_SEGS_COW
  pinned_segs   = 0x4, -- L4RE_AUX_LDR_FLAG_PINNED_SEGS
}

```

5.3.1.5 Application Startup Details

The central facility for starting a new task with Ned is the class `L4.Loader`. This class provides interfaces for conveniently configuring and starting programs. It provides three operations:

- `new_channel()` Returns a new IPC gate that can be used to connect two applications
- `start()` and `startv()` Start a new application process and return a process object

The `new_channel()` call is used to provide a service application with a communication channel to bind its initial service to. The concrete behavior of the object and the number of IPC gates required by a server depends on the server implementation. The channel can be passed to client applications as well or can be used for operations within the script itself.

`start()` and `startv()` always require at least two arguments. The first one is a table that contains information about the initial objects an application shall get. The second argument is a string, which for `start()` is the program name plus a white-space-separated list of program arguments (`argv`). For `startv()` the second argument is just the program binary name – which may contain spaces –, and the program arguments are provided as separate string arguments following the binary name (allowing spaces in arguments, too). The last optional argument is a table containing the POSIX environment variables for the program.

The Loader class uses reasonable defaults for most of the initial objects. However, you can override any initial object with some user-defined values. The main elements of the initial object table are:

- `factory` The factory used by the new process to create new kernel objects, such as threads etc. This must be a capability to an object implementing the [L4Re::Factory](#) protocol and defaults to the factory object provided to Ned.
- `mem` The memory allocator provided to the application and used by Ned allocates data spaces for the process. This defaults to Ned's memory allocator object (see [L4Re::Mem_alloc](#)).
- `rm_fab` The generic factory object used to allocate the region-map object for the process. (defaults to Ned's memory allocator).
- `log_fab` The generic factory to create the [L4Re::Log](#) object for the application's output (defaults to Ned's memory allocator). The `create` method of the `log_fab` object is called with `log_tag` and `log_color`, from this table, as arguments.
- `log` A table with parameters passed to the `log_fab`:
 - The first item is a short string defining the tag used for tagging log output of this process (defaults to the program name).
 - The second item is a string defining the color used for the log tag and the log string (defaults to "white").
 - Further parameters might be evaluated by certain implementations of the [L4Re::Log](#) interface.
- `scheduler` The scheduler object used for the process' threads (defaults to Ned's own scheduler).
- `caps` The table with application-specific named capabilities (default is an empty table). If the table does not contain a capability with the name 'rom', the 'rom' capability from Ned's initial caps is inserted into the table.

Less frequently used parameters:

- `l4re_dbg` An integer value overriding the debug level of the ITAS used for this application. Default is 2 (Warn). See *Debugging Flags* above.
- `ldr_flags` An integer value for setting additional flags for attaching regions, see *Loader Flags* above.

The `start()` and `startv()` calls return a task object that supports a number of operations.

- `state()` returns a string with the current task state: "running" or "zombie" if the task has terminated. If the task was already reaped (`wait()` returned) or if `kill()` was called, then the function will return `nil`.
- `exit_code()` returns the exit code if the task has terminated or `nil` if it was either killed or has been reaped.
- `kill()` forcefully terminates the task. Returns "killed" if the task was terminated or the exit code if the task was already gone. Returns `nil` if the task was already reaped.
- `exit_handler(function)` registers a Lua function that is invoked when the task terminates. If the task has already terminated, the function is called immediately. Returns `true` if the callback is pending, otherwise `false`. The callback function gets the exit code (`nil` if killed) of the task as its only parameter. The return value of the function is ignored. Only one callback can be registered.
- `wait()` suspends execution until the task has terminated. It's better to use `exit_handler()` instead. While the Lua code executes `wait()`, no `exit_handler()` will be dispatched.

5.3.1.6 Reacting on task termination

Ned can react on the termination of child tasks. The preferred mechanism is to register an `exit_handler()` for a task:

```
task = L4.default_loader:start(...)
task:exit_handler(function(exit_code)
  if exit_code == nil then
    print("Task was killed")
  else
    print("Task has terminated w/ code [" .. exit_code .. "]")
  end
end)
```

If the task did already terminate then the callback is invoked immediately. It is also possible to suspend execution of the Ned script until a task has terminated:

```
task = L4.default_loader:start(...)
task:wait()
```

This method should be used with caution, though. The Ned script will wait until the child task has terminated. Neither will any of the registered `exit_handler()` will be called during this time, nor will the [remote command interface](#) be able to execute commands either.

5.3.1.7 Control scheduling

Scheduling of [L4Re](#) applications is controlled by creating scheduler proxies. The proxy restricts the threads of an application to run on a subset of the available CPUs and to set their minimum and maximum priority. Use the loader factory function to create the proxy and pass it to the application:

```
sched_proxy = L4.default_loader:create_sched_proxy{ cpus=L4.Cpu_set:new("0-3") }
L4.default_loader:start({ scheduler = sched_proxy }, ...)
```

The `create_sched_proxy` function takes the following table arguments. All arguments are optional:

`cpus`: Set of allowed CPUs. Default: all CPUs. `prio_offset`: Base priority of all threads. Default: 0. `prio_limit`: Maximum priority of all threads. Default: `prio_offset + 10`. `fab`: Scheduler proxy factory capability. Default: `loader sched_fab`.

The `Cpu_set` constructor takes any number of CPU numbers or ranges:

```
Cpu_set:new()           -- all CPUs (because no argument was passed)
Cpu_set:new{}           -- empty CPU set (because an empty table was passed)
Cpu_set:new(42)         -- single CPU: 42
Cpu_set:new("0-3")      -- 4 CPUs: 0..3
Cpu_set:new("0-3", 42)  -- 5 CPUs: 0..3, 42
```

A limited number of operations are defined on CPU sets. To compute the union of two sets (all CPUs of both sets), use the `|` operator. To compute the intersection of two sets (all CPUs common to both sets), use the `&` operator.

5.3.1.8 Access to the kernel debugger

Applications can enrich the kernel debugger with information using the API defined in [l4/sys/debugger](#). In order to do so, the developer has to assign access to the kernel debugger kernel object to the application. This can be done like this:

```
L4.default_loader:start({ caps = { jdb = L4.Env.jdb; }}, "rom/example")
```

5.3.1.9 Using the interactive ned prompt

Ned can be used in interactive mode by connecting the small ned-prompt helper tool to the command capability. Add the following code snippet at the end of your ned script:

```
local L4 = require("L4");
local l = L4.default_loader;

cmd = l:new_channel()

l:start({ log = L4.Env.log, caps = { svr = cmd } }, "rom/ned-prompt")

L4.server_loop(cmd)
```

The script hands in ned's own log capability to `ned-prompt`. This ensures that input and output of ned and the prompt appear on the same console.

`ned-prompt` needs to be added to your modules list.

5.3.2 Command Line Options

Ned's command line syntax is:

```
[--exit|--wait-and-exit] [--disable-backtracer-autoload] [--execute|-e STATEMENT] <lua script> [options passed]
```

Ned interprets the first non-option argument `<lua script>` as the Lua script which it should load and run. All arguments following the first non-option argument are passed as arguments to the Lua script via Lua's global `arg` table.

- Exit Options: **exit**, **wait-and-exit** (must be first if used)
 - **exit**: terminates the application after the script has run through even if there are still tasks running. `wait` for tasks at the end of the script to ensure they do not die forcefully.
 - **exit-and-wait**: terminates the application after the script has run through and all tasks started by ned have signaled their exit.
- Execute Statement Option: **execute**, **e**
Execute the Lua statement `STATEMENT`.
- **disable-backtracer-autoload**: Do not load the backtracer automatically.

5.4 Io, the Io Server

The Io server handles all platform devices and resources such as I/O memory, ports (on x86) and interrupts, and grants access to those to clients.

Upon startup Io discovers all platform devices using available means on the system, e.g. on x86 the PCI bus is scanned and the ACPI subsystem initialised. Available I/O resource can also be configured via configuration scripts.

Io's configuration consists of two parts:

- the description of the real hardware
- the description of virtual buses

Both descriptions represent a hierarchical (tree) structure of device nodes. Where each device has a set of resources attached to it. And a device that has child devices can be considered a bus.

Hardware Description

The hardware description represents the devices that are available on the particular platform including their resource descriptions, such as MMIO regions, IO-Port regions, IRQs, bus numbers etc.

The root of the hardware devices is formed by a system bus device (accessible in the configuration via `lo.system↔_bus()`). As mentioned before, platforms that support methods for device discovery may populate the hardware description automatically, for example from ACPI. On platforms that do not have support for such methods you have to specify the hardware description by hand. A simple example for this is `x86-legacy.devs`.

Virtual Bus Description

Each lo server client is provided with its own virtual bus which it can iterate to find devices. A virtual PCI bus may be a part of this virtual bus.

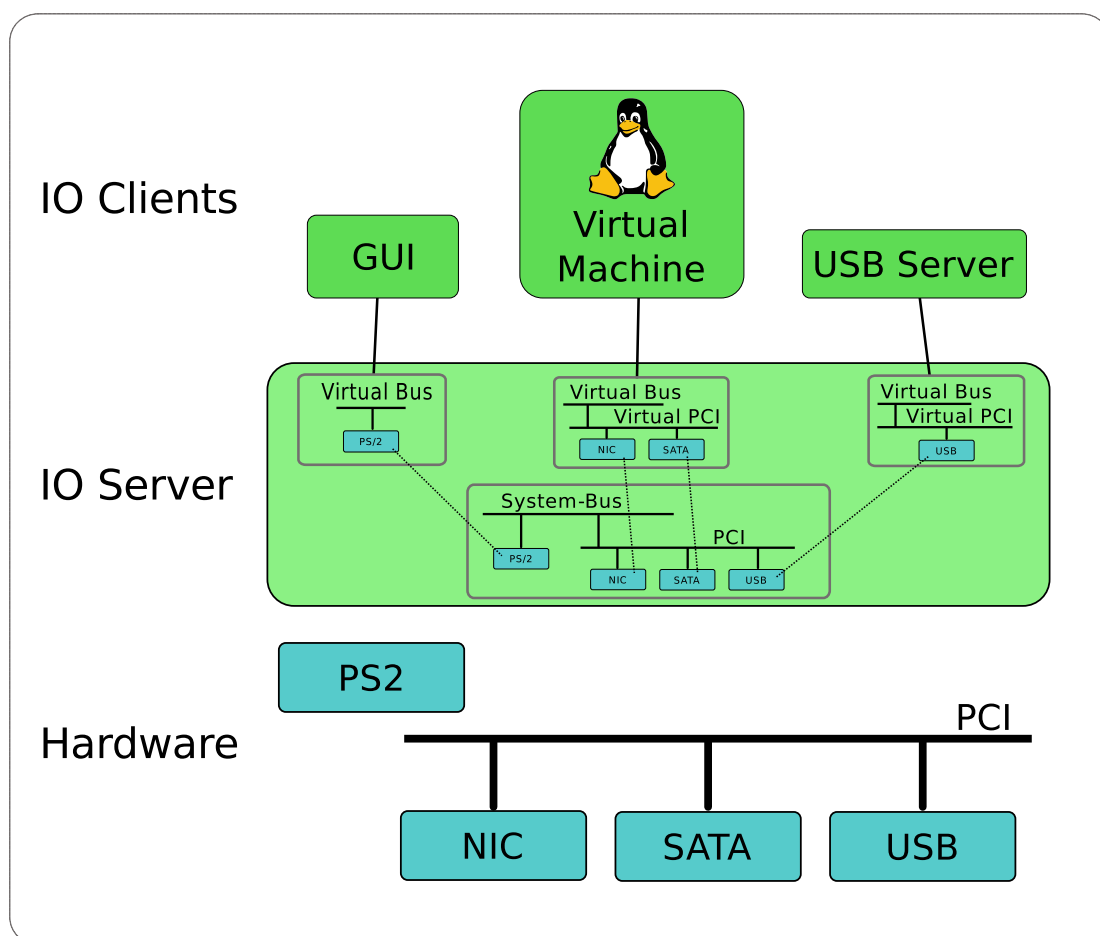


Figure 5.1 IO Service Architecture Overview

The lo server must be configured to create virtual buses for its clients.

This is done with at least one configuration file specifying static resources as well as virtual buses for clients. The configuration may be split across several configuration files passed to lo through the command line.

To allow clients access to available devices, a virtual system bus needs to be created that lists the devices and their resources that should be available to that client. The names of the buses correspond to the capabilities given to lo in its launch configuration.

A very simple configuration for Io could look like this:

```
-- vim:ft=lua
-- Example configuration for io

-- Configure two platform devices to be known to io
Io.Dt.add_children(Io.system_bus(), function()

    -- create a new hardware device called "FOODEVICE"
    FOODEVICE = Io.Hw.Device(function()
        -- set the compatibility IDs for this device
        -- a client tries to match against these IDs and configures
        -- itself accordingly
        -- the list should be sorted from specific to less specific IDs
        compatible = {"dev-foo,mmio", "dev-foo"};

        -- set the 'hid' property of the device, the hid can also be used
        -- as a compatible ID when matching clients
        Property.hid = "dev-foo,Example";

        -- note: names for resources are truncated to 4 letters and a client
        -- can determine the name from the ID field of a l4vbus_resource_t
        -- add two resources 'irq0' and 'reg0' to the device
        Resource.irq0 = Io.Res.irq(17);
        Resource.reg0 = Io.Res.mmio(0x6f000000, 0x6f007fff);
    end);

    -- create a new hardware device called "BARDEVICE"
    BARDEVICE = Io.Hw.Device(function()
        -- set the compatibility IDs for this device
        -- a client tries to match against these IDs and configures
        -- itself accordingly
        -- the list should be sorted from specific to less specific IDs
        compatible = {"dev-bar,mmio", "dev-bar"};

        -- set the 'hid' property of the device, the hid can also be used
        -- as a compatible ID when matching clients
        Property.hid = "dev-bar,Example";

        -- Specify that this device is able to use direct memory access (DMA).
        -- This is needed to allow clients to gain access to DMA addresses
        -- used by this device to directly access memory.
        Property.flags = Io.Hw_device_DF_dma_supported;

        -- note: names for resources are truncated to 4 letters and a client
        -- can determine the name from the ID field of a l4vbus_resource_t
        -- add three resources 'irq0', 'irq1', and 'reg0' to the device
        Resource.irq0 = Io.Res.irq(19);
        Resource.irq1 = Io.Res.irq(20);
        Resource.reg0 = Io.Res.mmio(0x6f100000, 0x6f100fff);
    end);
end);

Io.add_vbusses
{
    -- Create a virtual bus for a client and give access to FOODEVICE
    client1 = Io.Vi.System_bus(function ()
        dev = wrap(Io.system_bus():match("dev-foo,mmio"));
    end);

    -- Create a virtual bus for another client and give it access to BARDEVICE
    client2 = Io.Vi.System_bus(function ()
        dev = wrap(Io.system_bus():match("dev-bar,Example"));
    end);
}
```

Each device supports a 'compatible' property. It is a list of compatibility strings. A client matches itself against one (or multiple) compatibility IDs and configures itself accordingly. All other device members are handled according to their type. If the type is a resource (Io.Res) it is added as a named resource. Note that resource names are truncated to 4 letters and are stored in the ID field of a [l4vbus_resource_t](#). If the type is a device it is added as a child device to the current one. All other types are treated as a device property which can be used to configure a device driver. Right now, device properties are internal to Io only.

Matching and Assigning PCI Devices

Assigning clients PCI devices could look like this:

```
-- This is a configuration snippet for PCI device selection

local hw = Io.system_bus();

Io.add_vbusses
{
  pciclient = Io.Vi.System_bus(function ()
    PCI = Io.Vi.PCI_bus(function ()
      pci_mm      = wrap(hw:match("PCI/CC_04"));
      pci_net     = wrap(hw:match("PCI/CC_02"));
      pci_storage = wrap(hw:match("PCI/CC_01"));
    end)
  end)
}
```

The "PCI/" is followed by a bus-specific ID string. The format of the PCI ID string may be one of the following:

- PCI/CC_cc
- PCI/CC_ccss
- PCI/CC_ccssp
- PCI/VEN_vvvv
- PCI/DEV_dddd
- PCI/SUBSYS_ssssssss
- PCI/REV_rr
- PCI/ADR_xxxx:xx:xx.x

Where:

- `cc` is the hexadecimal representation of the class code byte
- `ss` is the hexadecimal representation of the subclass code byte
- `pp` is the hexadecimal representation of the programming interface byte
- `vvvv` is the hexadecimal representation of the vendor ID
- `dddd` is the hexadecimal representation of the device ID
- `ssssssss` is the hexadecimal representation of the subsystem ID
- `rr` is the hexadecimal representation of the revision byte
- `xxxx:xx:xx.x` is the bus address in PCI nomenclature

As a special extension Io supports replacing the ID string with a human-readable common PCI class name. The following table gives an overview of the names known to Io and their respective PCI class and subclass.

Common Name	Description	PCI ID string
storage	Mass storage controller	CC_01
scsi	SCSI storage controller	CC_0100
ide	IDE interface	CC_0101
floppy	Floppy disk controller	CC_0102
raid	RAID bus controller	CC_0104
ata	ATA controller	CC_0105

Common Name	Description	PCI ID string
sata	SATA controller	CC_0106
sas	Serial attached SCSI controller	CC_0107
nvm	Non-volatile memory controller	CC_0108
-	-	-
network	Network controller	CC_02
ethernet	Ethernet controller	CC_0200
token_ring	Token ring network controller	CC_0201
fddi	FDDI network controller	CC_0202
atm	ATM network controller	CC_0203
isdn	ISDN controller	CC_0204
picmg	PICMG controller	CC_0206
net_infiniband	Infiniband controller	CC_0207
fabric	Fabric controller	CC_0208
network_nw	Network controller e.g. Wifi	CC_0280
-	-	-
display	Display controller	CC_03
vga	VGA compatible controller	CC_0300
xga	XGA compatible controller	CC_0301
-	-	-
media	Multimedia controller	CC_04
mm_video	Multimedia video controller	CC_0400
mm_audio	Multimedia audio controller	CC_0401
telephony	Computer telephony device	CC_0402
audio	Audio device	CC_0403
-	-	-
bridge	Bridge	CC_06
br_host	Host bridge	CC_0600
br_isa	ISA bridge	CC_0601
br_eisa	EISA bridge	CC_0602
br_microchannel	MicroChannel bridge	CC_0603
br_pci	PCI bridge	CC_0604
br_pcmcia	PCMCIA bridge	CC_0605
br_nubus	NuBus bridge	CC_0606
br_cardbus	CardBus bridge	CC_0607
br_raceway	RACEway bridge	CC_0608
br_semi_pci	Semi-transparent PCI-to-PCI bridge	CC_0609
br_infiniband_to_pci	InfiniBand to PCI host bridge	CC_060a
-	-	-
com	Communication controller	CC_07
com_serial	Serial controller	CC_0700
com_parallel	Parallel controller	CC_0701

Common Name	Description	PCI ID string
com_multiport_ser	Multiport serial controller	CC_0702
com_modem	Modem	CC_0703
com_gpib	GPiB controller	CC_0704
com_smart_card	Smart card controller	CC_0705
-	-	-
serial_bus	Serial bus controller	CC_0c
firewire	FireWire (IEEE 1394)	CC_0c00
access_bus	ACCESS bus	CC_0c01
ssa	SSA	CC_0c02
usb	USB controller	CC_0c03
fibre_channel	Fibre channel	CC_0c04
smbus	SMBus	CC_0c05
bus_infiniband	InfiniBand	CC_0c06
ipmi_smic	IPMI SMIC interface	CC_0c07
sercos	SERCOS interface	CC_0c08
canbus	CAN bus	CC_0c09
-	-	-
wireless	Wireless controller	CC_0d
bluetooth	Bluetooth	CC_0d11
w_8021a	802.1a controller	CC_0d20
w_8021b	802.1b controller	CC_0d21

Strong Matching of PCI Devices

If more specific matching of PCI devices is required it is possible to concatenate multiple ID strings using &. An example where a specific device from a specific vendor at a fixed bus address is matched would use the string `PCI/VEN_vvvv&DEV_dddd&ADR_xxxx:xx:xx.x`.

Isolation of PCIe devices

PCIe encodes device communication with a network-like protocol with destination headers and packet fragmentation allowing a devices to talk directly to other devices. This potentially works against security boundaries for a system. E.g. two network cards could exchange packets and thereby leak information from one security domain to the other without involvement of the OS.

PCIe introduced an optional capability named PCI Access Control Services (PCI/ACS) to control communication between PCIe devices.

With PCI/ACS it is possible to restrict inter-device communication between PCIe devices.

PCI/ACS is optional and for Intel chipsets, it is usually only implemented on high-end PCI platform controller hubs (PCHs), and is missing on low-end and mobile PCHs. On some Intel-PCHs there exist facilities that allow for similar isolation.

If IO encounters a supported PCH, it will enable those facilities in order to enforce device isolation.

Command Line Options

The lo Server supports the following optional parameters:

```
[--verbose|v] [--transparent-msi] [--trace <trace_mask>] [--acpi-debug-level <debug_level>] [config_files]
```

- **verbose|v**

By default, error debug messages are enabled. This option increments the verboseness level, and can be applied multiple times to reach the desired debug level. The available debug levels are ordered as: DBG_ERR (default, level 1), DBG_WARN, DBG_INFO, DBG_DEBUG, DBG_DEBUG2 and DBG_ALL (level 6).

- **transparent-msi**

Enable MSI on PCI devices which support this feature. This is transparent to clients, as there are no changes in the API used to interact with PCI device via interrupts.

- **acpi-debug-level <level_mask>**

Set the ACPI debug level. The <level_mask> is a mask that selects components of interest for debugging. It can be constructed from the ACPI debug constants defined in the linux kernel, see [ACPI Debug Output](#) for details. By default, the ACPI debug level is set to ACPI_LV_INIT | ACPI_LV_TABLES | ACPI_LV_VERBOSE_INFO.

- **trace <trace_mask>**

Enable tracing of events matching trace_mask. The only supported trace mask is 1 and this matches ACPI events.

- **config_files**

Space separated list of Lua configuration files specifying real hardware and virtual buses. See example on [Virtual Bus Description](#).

5.5 l4vio_net_p2p, a virtual network point-to-point link

The virtual network point-to-point server (p2p) connects two clients with a virtual network connection. It uses virtio as the transport mechanism. Each virtual network p2p endpoint implements the device-side of a virtio network device. Each client can access its endpoint using the driver-side semantics of a virtio network device.

Building and Configuration

The virtual network p2p server can be built using the [L4Re](#) build system by placing this project into the pkg directory.

Starting the service

The virtual network p2p server can be started with Lua like this:

```
local p2p = L4.default_loader:new_channel();
L4.default_loader:start(
{
  caps = {
    svr = p2p:svr(),
  },
},
"rom/l4vio_net_p2p [<options>]");
```

First an IPC gate (p2p) is created which is used between the virtual network p2p server and a client to create new virtual ports. The server-side is assigned to the mandatory svr capability of the virtual network p2p server. See the section below on how to create a new virtual port and connect a client to it.

Options

The following command line options are supported:

- `-p <num_usec>, --poll <num_usec>`
Enable polling mode and set the poll interval. IRQ notification is disabled for queues while in polling mode. Must be a positive integer specified in microseconds.
- `-s <num>, --size <num>`
Set the maximum queue size for the device-side virtio queues. Must be a power of 2 in the range of 1 to 32768 inclusive.

Connecting a client

Prior to connecting a client to a virtual network p2p server port it has to be created using the following Lua function. It has to be called on the client side of the IPC gate capability whose server side is bound to the virtual network p2p server.

The "key=value" pairs passed to `create()` can be omitted and their order is not important.

```
create(obj_type, [ "ds-max=<max>" , "mac=<mac_address>" ])
```

- `obj_type`
The type of object that should be created by the server. The type must be a positive integer. Currently the following objects are supported:
 - 0: Virtual p2p port
- `"ds-max=<max>"`
Specifies the upper limit of the number of dataspaces the client is allowed to register with the server for virtio DMA. Must be in the range of 1 to 80 inclusive. The default value is 2.
- `"mac=<mac_address>"`
Specify the MAC address of the endpoint where `<mac_address>` is of the form X:XX:Xx:x:xx:XX.

If the `create()` call is successful a new capability which references a virtual network p2p server port is returned. A client uses this capability to talk to the virtual network p2p server using the virtio network protocol.

A couple of examples on how to create ports with different properties are listed below.

```
-- two normal ports with at most 4 data spaces
net0 = p2p:create(0, "ds-max=4")
net1 = p2p:create(0, "ds-max=4")
-- normal port with 4 data spaces and MAC address
net0 = p2p:create(0, "ds-max=4", "mac=11:22:33:44:55:66")
```

5.6 l4vio_switch, a virtual network switch

The virtual network switch connects multiple clients with a virtual network connection. It uses Virtio as the transport mechanism. Each virtual switch port implements the host-side of a Virtio network device (virtio-net).

The virtual network switch can be setup to feature exactly one monitor port. All traffic passing through the switch is mirrored to the monitor port. The monitor port is read-only, and has no TX capability. An optional packet filter can be configured and implemented to filter data sent to the monitor port.

Configuration

Certain features of the virtual network switch are configurable at compile-time. Configuration is done through the build-time configuration of the [L4Re](#) build tree.

Starting the service

The virtual network switch can be started in Ned like this:

```
local switch = L4.default_loader:new_channel();
L4.default_loader:start(
{
  caps = {
    svr = switch:svr(),
  },
},
"rom/l4vio_switch");
```

First a communication channel (`switch`) is created which is used to create virtual network ports. It is connected to the switch component via its mandatory `svr` capability. See the section below on how to create a new virtual port and connect a client to it.

Options

In the example above the virtual network switch is started in its default configuration with a maximum of 5 virtual ports. To customize the configuration the virtual network switch accepts the following command line options:

- `-D <component=level>, --debug <component=level>`

Configure individual debug levels per component. Allowed components are:

`core, virtio, port, request, queue, packet`

Possible debug levels with increasing verbosity are:

`quiet, warn, info, debug, trace`

- `-m, --mac`

Ignored. Provided for compatibility with older switch versions.

- `-M`

Do not assign a random MAC address to ports by default. It is always possible to set an explicit MAC address by passing the `mac=` to the factory call, regardless of this option. If `-M` is passed and no `mac=` address was given, it is the responsibility of the Virtio driver to choose an appropriate address.

- `-p <num>, --ports <num>`

Set the maximum number of virtual ports. The default is 5.

- `-q, --quiet`

Silence all output except for error messages.

- `-s <num>, --size <num>`

Set the maximum queue size for the device-side Virtio queues. Must be a power of 2 in the range of 1 to 32768 inclusive.

- `-v, --verbose`

Increase the global verbosity level. Individual levels per component can be set using the `-D` option.

- `-d <cap_name>, --register-ds <cap_name>` Register a trusted dataspace capability. If this option gets used, it is not possible to communicate with the server via dataspace capabilities other than the registered ones. Can be used multiple times for multiple dataspace capabilities.

The option parameter is the name of a dataspace capability.

Hardware devices

To plug hardware devices into the switch, provide a Vbus capability with the name `vbus` when starting the switch. To use this feature, you have to enable the `VNS_IXL` config option.

Connecting a client

First, a virtual network port has to be created using the following Ned-Lua function. It has to be called on the communication channel called `switch`, which has been created earlier.

```
create(obj_type, ["ds-max=<max>", "name=<name>", "type=<port type>",
                 "vlan=<options>", "mac=<mac_address>"])
```

- `obj_type`

The type of object that should be created by the switch. The type must be a positive integer. Currently the following objects are supported:

- 0: Virtual switch port

- `ds-max=<max>`

Specifies the upper limit of the number of dataspace the client is allowed to register with the virtual network switch for Virtio DMA.

- `name=<name>`

Sets the name of port in debug messages to `<name>`. A name may consist of at most 19 characters, all other characters are dropped. If there is enough space left, the name will get a postfix of "`[<port number>]`", e.g. "`name=foo`" -> `foo[1]`.

- `type=<port type>`

Optionally specify the port type, either `normal` or a `monitor` port. Valid types are `[monitor|normal]`. The default is `type=normal` (if no type is given).

- `vlan=(access=<vlan id>|trunk=[<vlan id>[,<vlan id>]*])`

Configure the port to participate in an IEEE 802.1Q compatible VLAN. Fundamentally there are two types of ports: access ports and trunk ports:

- `vlan=access=<vlan id>`

Configures the port as access port for VLAN `<vlan id>` where the id must be a decimal number greater than 0 and less than 4095 in accordance to the standard. Packets on an access port belong to the configured VLAN and are only forwarded to ports that belong to the same VLAN or trunk ports that participate in the particular VLAN. The packets on this port will not have a VLAN tag attached to them so that a guest connected to this port does not see that the port is part of a VLAN.

An optional monitor port will see packets from an access port as VLAN tagged packets with the `<vlan id>` given for the port.

- `vlan=trunk=all| [<vlan id>[,<vlan id>]*]`

Configures the port as trunk port. It participates either in all VLANs, if specified by the keyword 'all', or in the list of VLANs given as comma separated list. There must be no whitespace in the list. Each id must be a decimal number greater than 0 and less than 4095 in accordance to the standard. Outgoing packets on this port will be tagged with an IEEE 802.1Q compatible tag. Incoming packets must be tagged with a VLAN tag from the given list. Packets that have no tag or a tag not in the vlan id list are dropped silently. They are not forwarded to the monitor port either.

Currently there is no support for IEEE 802.1p. The PCP and DEI sub-fields in the TCI field will be set to zero on outgoing packets and are ignored for incoming packets.

- `mac=xx:xx:xx:xx:xx:xx`

Explicitly sets the MAC address of the port. It will be checked that no other port on the switch has the same address. It is the responsibility of the user to ensure the validity of the address and its global uniqueness, though.

If the `create()` call is successful a new capability which references a virtual switch port is returned. A client uses this capability to talk to the virtual network switch using the Virtio network protocol.

Here are couple of examples on how to create ports with different properties:

```
-- normal port with at most 4 data spaces
net0 = switch:create(0, "ds-max=4")
-- like the previous but with name foo
net0 = switch:create(0, "ds-max=4", "name=foo")
-- like the previous but the port is a monitor port
net0 = switch:create(0, "ds-max=4", "name=foo", "type=monitor")
-- normal port with 4 data spaces as access port to VLAN 1
net0 = switch:create(0, "ds-max=4", "name=v11", "vlan=access=1")
-- normal port with 4 data spaces as trunk port participating in VLAN 1 & 2
net0 = switch:create(0, "ds-max=4", "name=v11", "vlan=trunk=1,2")
```

5.7 Uvmm, the virtual machine monitor

Uvmm provides a virtual machine for running an unmodified guest in non-privileged mode.

Command Line Options

uvmm provides the following command line options:

- `-c, --cmdline=<guest command line>`
Command line that is passed to the guest on boot.
- `-k, --kernel=<kernel image name>`
The name of the guest-kernel image file present in the ROM namespace.
- `-d, --dtb=<DTB overlay>`
The name of the device tree file present in the ROM namespace. The device tree will be placed in the upmost region of guest memory. Optionally, a user may use an additional parameter in the form of "`<DTB overlay>:limit=0xffffffff`" to set an upper limit for the device tree location.
- `-r, --ramdisk=<RAM disk name>`
The name of the RAM disk file present in the ROM namespace
- `-b, --rambase=<Base address of the guest RAM>`
Physical start address for the guest RAM. This value is platform specific.
- `-D, --debug=[<component>=] [level]`
Control the verbosity level of the uvmm. Possible `level` values are: `quiet`, `warn`, `info`, `trace`
Using the `component` prefix, the verbosity level of each uvmm component is configurable. The component names are: `core`, `cpu`, `mmio`, `irq`, `dev`, `pm`, `vbus_event`
For example, the following command line sets the verbosity of all uvmm components to `info` except for IRQ handling, which is set to `trace`.

```
uvmm -D info -D irq=trace
```

- `-f, --fault-mode`

Control the handling of guest reads/writes to non-existing memory. Possible values are:

- `ignore` - Invalid writes are ignored. Invalid reads either return 0 or are skipped. The guest may experience undefined behaviour.
- `halt` - Halt the VM on the first invalid memory access.
- `inject` - Try to forward the invalid access to the guest. This is not supported on all architectures. Falls back to `halt` if the error could not be forwarded to the guest.

Defaults to `ignore`.

- `-q, --quiet`

Silence all uvmm output.

- `-v, --verbose`

Increase the verbosity of the uvmm. Repeating the option increases the verbosity by another level.

- `-W, --wakeup-on-system-resume`

When set, the uvmm resumes when the host system resumes after a suspend call.

- `-i`

When set, the option forces the guest RAM to be mapped to its corresponding host-physical addresses.

Note

Options `-q, --quiet`, `-v, --verbose` and `-D, --debug` cancel each other out.

Setting up guest memory

In the most simple setup, memory for the guest can be provided via a simple dataspace. In your ned script, create a new dataspace of the required size and hand it into uvmm as the `ram` capability:

```
local ramds = L4.Env.user_factory:create(L4.Proto.Dataspace, 60 * 1024 * 1024)

L4.default_loader::startv({caps = {ram = ramds:m("rw")}}, "rom/uvmm")
```

The memory will be mapped to the most appropriate place and a memory node added to the device tree, so that the guest can find the memory.

For a more complex setup, the memory can be configured via the device tree. uvmm scans for memory nodes and tries to set up the memory from them. A memory device node should look like this:

```
memory@0 {
    device_type = "memory";
    reg = <0x00000000 0x00100000
          0x00200000 0xffffffff>;
    l4vmm,dscap = "memcap";
    dma-ranges = <>;
};
```

The `device_type` property is mandatory and needs to be set to `memory`.

`l4vmm,dscap` contains the name of the capability containing the dataspace to be used for the RAM. `reg` describe the memory regions to use for the memory. The regions will be filled up to the size of the supplied dataspace. If they are larger, then the remaining area will be cut.

If the optional `dma-ranges` property is given, the host-physical address ranges for the memory regions will be added here. Note that the property is not cleared first, so it should be left empty.

For more details see [RAM configuration](#).

Memory layout

uvmm populates the RAM with the following data:

- kernel binary
- (optional) ramdisk
- (optional) device tree

The kernel binary is put at the predefined address. For ELF binaries, this is an absolute physical address. If the binary supports relative addressing, the binary is put to the requested offset relative to beginning of the first 'memory' region defined in the device tree.

The ramdisk and device tree are placed as far as possible to the end of the regions defined in the first 'memory' node.

If there is a part of RAM that must remain empty, then define an extra memory node for it in the device tree. uvmm only writes to memory in the first memory node it finds.

Warning: uvmm does not touch any unpopulated memory. In particular, it does not ensure that the memory is cleared. It is the responsibility of the provider of the RAM dataspace to make sure that no data leakage can happen. Normally this is not an issue because dataspaces are guaranteed to be cleaned when they are newly created but users should be careful when reusing memory or dataspaces, for example, when restarting the uvmm.

Forwarding hardware resources to the guest

Hardware resources must be specified in two places: the device tree contains the description of all hardware devices the guest could see and the Vbus describes which resources are actually available to the uvmm.

The vbus allows the uvmm access to hardware resources in the same way as any other [L4](#) application. uvmm expects a capability named 'vbus' where it can access its hardware resources. It is possible to leave out the capability for purely virtual guests (Note that this is not actually practical on some architectures. On ARM, for example, the guest needs hardware access to the interrupt controller. Without a 'vbus' capability, interrupts will not work.) For information on how to configure a vbus, see the [IO documentation](#).

The device tree needs to contain the hardware description the guest should see. For hardware devices this usually means to use a device tree that would also be used when running the guest directly on hardware.

On startup, uvmm scans the device tree for any devices that require memory or interrupt resources and compares the required resources with the ones available from its vbus. When all resources are available, it sets up the appropriate forwarding, so that the guest now has direct access to the hardware. If the resources are not available, the device will be marked as 'disabled'. This mechanism allows to work with a standard device tree for all guests in the system while handling the actual resource allocation in a flexible manner via the vbus configuration.

The default mechanism assigns all resources 1:1, i.e. with the same memory address and interrupt number as on hardware. It is also possible to map a hardware device to a different location. In this case, the assignment between vbus device and device tree device must be known in advance and marked in the device tree using the `l4vmm, vbus-dev` property.

The following device will for example be bound with the vbus device with the HID 'l4-test,dev':

```
test@e0000000 {
    compatible = "memdev,bar";
    reg = <0 0xe0000000 0 0x50000>,
        <0 0xe1000000 0 0x50000>;
    l4vmm,vbus-dev = "l4-test,dev";
    interrupts-extended = <&gic 0 139 4>;
};
```

Resources are then matched by name. Memory resources in the vbus must be named `reg0` to `reg9` to match against the address ranges in the device tree `reg` property. Interrupts must be called `irq0` to `irq9` and will be matched against `interrupts` or `interrupts-extended` entries in the device tree. The vbus must expose resources for all resources defined in the device tree entry or the initialisation will fail.

An appropriate IO entry for the above device would thus be:

```
MEM = Io.Hw.Device(function()
    Property.hid = "l4-test,dev"
    Resource.reg0 = Io.Res.mmio(0x41000000, 0x4104ffff)
    Resource.reg1 = Io.Res.mmio(0x42000000, 0x4204ffff)
    Resource.irq0 = Io.Res.irq(134);
end)
```

Please note that HIDs on the vbus are not necessarily unique. If multiple devices with the HID given in `l4vmm,vbus-dev` are available on the vbus, then one device is chosen at random.

If no vbus device with the given HID is available, the device is disabled.

How to enable guest suspend/resume

Note

Currently only supported on ARM. It should work fine with Linux version 4.4 or newer.

Uvmm (partially) implements the power state coordination interface (PSCI), which is the standard ARM power management interface. To make use of this interface, you have to announce its availability to the guest operating system via the device tree like so:

```
psci {
    compatible = "arm,psci-0.2";
    method = "hvc";
};
```

The Linux guest must be configured with at least these options:

```
CONFIG_SUSPEND=y
CONFIG_ARM_PSCI=y
```

How to communicate power management (PM) events

Uvmm can be instructed to inform a PM manager of PM events through the [L4::Platform_control](#) interface. To that end, uvmm may be equipped with a `pfc` cap. On suspend, uvmm will call [l4_platform_ctl_system_suspend\(\)](#).

The `pfc` cap can also be implemented by IO. In that case the guest can start a machine suspend/shutdown/reboot.

Ram block device support

The example ramdisk works by loading a file system into RAM, which needs RAM block device support to work. In the Linux kernel configuration add: `CONFIG_BLK_DEV_RAM=y`

Framebuffer support for uvmm/amd64 guests

Uvmm can be instructed to pass along a framebuffer to the Linux guest. To enable this three things need to be done:

1. Configure Linux to support a simple framebuffer by enabling `CONFIG_FB_SIMPLE=y` `CONFIG_X86_`↔
`SYSFB=y`
2. Configure a simple framebuffer device in the device tree (currently only read by uvmm, linearer framebuffer at [0xf0000000 - 0xf1000000])

```
simplefb { compatible = "simple-framebuffer"; reg = <0x0 0xf0000000 0x0 0x1000000>; l4vmm,fbcap = "fb";
};
```
3. Start a framebuffer instance and connect it to uvmm e.g. – Start fb-drv (but only if we need to) local `fbdrv_fb`
`= L4.Env.vesa;` if (not `fbdrv_fb`) then `fbdrv_fb = l:new_channel(); l:start({ caps = { vbus = io_busses.fbdrv, fb`
`= fbdrv_fb:svr(), }, log = { "fbdrv", "r" }, }, "rom/fb-drv"); end vmm.start_vm{ ext_caps = { fb = fbdrv_fb }, – ...`

Requirements on the Fiasco.OC configuration on amd64

The kernel configuration must feature `CONFIG_SYNC_TSC=y` in order for the emulated timers to reach a sufficiently high resolution.

Recommended Linux configuration options for uvmm/amd64 guests

The following options are recommended in addition to the amd64 defaults provided by a `make defconfig`:

Virtio support is required to access virtual devices provided by uvmm:

```
CONFIG_VIRTIO=y
CONFIG_VIRTIO_PCI=y
CONFIG_VIRTIO_BLK=y
CONFIG_BLK_MQ_VIRTIO=y
CONFIG_VIRTIO_CONSOLE=y
CONFIG_VIRTIO_INPUT=y
CONFIG_VIRTIO_NET=y
```

It is highly recommended to use the X2APIC, which needs virtualization awareness to work under uvmm:

```
CONFIG_X86_X2APIC=y
CONFIG_PARAVIRT=y
CONFIG_PARAVIRT_SPINLOCKS=y
```

KVM clock for uvmm/amd64 guests

When executing L4Re + uvmm on QEMU, the PIT as clock source is not reliable. The paravirtualized KVM clock provides the guest with a stable clock source.

A KVM clock device is available to the guest, if the device tree contains the corresponding entry:

```
kvm_clock {
    compatible = "kvm-clock";
    reg = <0x0 0x0 0x0 0x0>;
};
```

To make use of this clock, the Linux guest must be built with the following configuration options:

```
CONFIG_HYPERVISOR_GUEST=y
CONFIG_KVM_GUEST=y
CONFIG_PTP_1588_CLOCK_KVM is not set
```

Note: KVM calls besides the KVM clock are unhandled and lead to failure in the uvmm, e.g. vmcall 0x9 for the PTP_1588_CLOCK_KVM.

This is considered a development feature. The KVM clock is not required when running on physical hardware as TSC calibration via the PIT works as expected.

Development notes for amd64

When you are developing on Linux using QEMU please note that nested virtualization support is necessary on your host system to run uvmm guests. Your host Linux version should be 4.12 or greater, **excluding 4.20**.

Check if your KVM module has nested virtualization enabled via:

```
> cat /sys/module/kvm_intel/parameters/nested
Y
```

In case it shows N instead of Y enable nested virtualization support via:

```
modprobe kvm_intel nested=1
```

On AMD platforms the module name is `kvm_amd`.

QEMU network setup for a uvmm guest on amd64

```
qemu-system-x86_64 -M q35 -cpu host -enable-kvm -device intel-iommu -device e1000e,netdev=net0 -netdev
bridge,id=net0,br=virbr0
```

where 'virbr0' is the name of the host's bridge device. The line 'allow virbr0' needs to be present in /etc/qemu/bridge.conf. The bridge can either be created via the network manager or via the command line↵:

```
brctl addbr virbr0
ip addr add 192.168.124.1/24 dev virbr0
ip link set up dev virbr0
```

In the guest linux with eth0 as network device:

```
ip a a 192.168.124.5/24 dev eth0
ip li se up dev eth0
```

Now the host and guest can ping each other using their respective IPs.

Of course, uvmm needs to be connected to io and io needs a vbus configuration for the uvmm client like this:

```
Io.add_vbusses
{
  vm_pci = Io.Vi.System_bus(function ()
    Property.num_msis = 6
    PCI = Io.Vi.PCI_bus(function ()
      pci_net = wrap(Io.system_bus():match("PCI/CC_0200"))
    end)
  end)
}
```

QEMU emulated VirtIO devices and IO-MMU on amd64

QEMU does not route VirtIO devices through the IO-MMU per default. To use QEMU emulated VirtIO devices add the `disable-legacy=on,disable-modern=off,iommu_platform=on` flags to the option list of the device. The e1000e card in the network example above can be replaced with an virtio-net-pci card like this:

```
-device virtio-net-pci,disable-legacy=on,disable-modern=off,
      iommu_platform=on,netdev=net0
```

For more information on VirtIO devices and their options see <https://wiki.qemu.org/Features/VT-d>.

Using the uvmm monitor interface

Uvmm implements an interface with which parts of the guest's state can be queried and manipulated at runtime. This monitor interface needs to be enabled during compilation as well as during startup of uvmm. This is described in detail below.

Compiling uvmm with monitor interface support

To compile uvmm with monitor interface support pass the `CONFIG_MONITOR=y`, option during the `make` step (or set in in the `Makefile.config`). This option is available on all architectures but note that the set of available monitor interface features may vary significantly between them. Also note that the monitor interface will always be disabled in release mode, i.e. if `CONFIG_RELEASE_MODE=y`.

Enabling the monitor interface at runtime

When starting a uvmm instance from inside a `ned` script using the `vmm.start_vm` function, the `mon` argument controls whether the monitor interface is enabled at runtime. There are three cases to distinguish:

- `mon=true` (default): The monitor interface is enabled but no server implementing the client side of the monitor interface is started. The monitor interface can still be utilized via `cons` but no readline functionality will be available.
- `mon='some_binary'`: If a string is passed as the value of `mon`, the monitor interface is enabled and the string is interpreted as the name of a server binary which implements the client side of the monitor interface. This server is automatically started and has access to a `vcon` capability named `mon` at startup through which it can make use of the monitor interface. Unless you have written your own server you should specify `'uvmm_cli'` which is a server implementing a simple readline interface.
- `mon=false`: The monitor interface is disabled at runtime.

Using the monitor interface

If the monitor interface was enabled you can connect to it via `cons` under the name `mon<n>` where `<n>` is a unique integer for every uvmm instance that is started with the monitor interface enabled (numbered starting from one in order of corresponding `vmm.start_vm` calls). If `mon='uvmm_cli'` was specified, readline functionality such as command completion and history will be available. Enter a command followed by enter to run that command. To obtain a list of all available commands issue the `help` command, to obtain usage information for a specific command `foo` issue `help foo`.

Note

Some commands will modify the guests state. Since it should be obvious to which ones this applies this is usually not specifically highlighted. Exercise reasonable caution.

Using the guest debugger

The guest debugger provides monitoring functionality akin to a very bare-bone GDB interface, e.g. guest RAM and page table dumping, breakpointing and single stepping. Additional functionality might be added in the future.

Note

The guest debugger is currently still under development. The guest debugger may also not be available on all architectures. To check whether the guest debugger is available check if `help dbg` returns usage information.

If the guest debugger is available, you have to manually load it at runtime using the monitor interface. This saves resources if the guest debugger is not used. To enable the guest debugger, issue the `dbg on` monitor command. Once enabled, the guest debugger can not be disabled again.

To list available guest debugger subcommands, issue `dbg help` after `dbg on`.

Note

When using SMP, most guest debugger subcommands require you to explicitly specify a guest vcpu using an index starting from zero.

5.7.1 RAM configuration

RAM configuration for uvmm

Without a memory node in the device tree

- setup default RAM for guest VM.
- RAM starts either
 - at base-address which defaults to 0x0 or the base address value set via the -b cmdline option or
 - in case of identity mapping at the host-physical address of the dataspace allocated for the RAM

With a memory node in the device tree

The memory node needs at least the properties `device_type` and `l4vmm,dscap`:

```
memory@0 {  
    device_type = "memory";  
    l4vmm,dscap = "ram";  
}
```

Where the given `l4vmm,dscap` name is accessible in the capability namespace of the uvmm. If the capability is invalid, the memory node is disabled.

If memory nodes are given, but none provides valid RAM the configuration is invalid and uvmm refuses to boot.

Additional properties of the memory node are `reg` and `dma-ranges`.

The `reg` property describes the location in the guest's address space that should be backed by RAM.

The `dma-ranges` property describes the offset between guest-physical and host-physical addresses. The guest can evaluate this non-standard property to derive the correct DMA addresses to program into passed-through devices. Usage of this property **requires** modification of guest code.

Without `reg` and `dma-ranges` properties

The `reg` property is optional only in case the uvmm maps the guest's RAM into the VM under the host-physical addresses of the backing memory (`l4vmm,dscap`).

This case can be forced via the cmdline parameter `-i` and is the default for platforms without IOMMU, but with DMA capable devices on the configured vBus.

Without a `reg` property, but with a `dma-ranges` property

If the `-i` cmdline parameter is given, identity mapping is forced and the behavior is the same as in the case above. Additionally, the `dma-ranges` property is written

In case no `-i` cmdline parameter is given, the configuration is invalid and uvmm refuses to boot.

With a reg property

uvmm parses the reg property of the memory node and maps the memory into the VM to the given range(s).

If the -i cmdline parameter is set, the reg property is ignored and the memory is mapped into the VM under the corresponding host-physical addresses of the backing memory (l4vmm,dscap)

With a reg and dma-ranges property

uvmm parses the reg property of the memory node and maps the memory into the VM to the given range(s).

The dma-ranges property is filled with the corresponding host-physical addresses of the backing memory (l4vmm,dscap).

5.8 RTC driver

The RTC driver can drive various real-time clocks and provides an interface for other components, e.g., uvmm, to read and write the time.

It needs access to the hardware, depending on the clock, either via a vbus or via an I2C device.

Command Line Options

There are no command line options.

Environment

Several capabilities can be used to interact with the environment:

- 'rtc' (server)
The capability with which clients talk to the service. Note that writing to the RTC is only possible when having the rtc capability with write rights, read-only clients must have the capability with read rights only.
- 'vbus'
The vbus used to access hardware for port-based clock on X86 and pl031 on arm.
The vbus is also needed for receiving the inhibitor signal from IO.

5.9 NVMe server

The NVMe server is a driver for PCI Express NVMe controllers.

The NVMe server is capable of exposing entire disks (i.e. NVMe namespaces) (by serial number and namespace identifier) or individual partitions (by their partition UUID) of a hard drive to clients via the Virtio block interface.

The server consists of two parts. The first one is the hardware driver itself that takes care of the communication with the underlying hardware and interrupt handling. The second part implements a virtual block device and is responsible to communicate with clients. The virtual block device translates commands it receives into NVMe requests and issues them to the hardware driver.

The NVMe server allows both statically and dynamically configured clients. A static configuration is given priority over dynamically connecting clients and configured while the service starts. Dynamic clients can connect and disconnect during runtime of the NVMe server.

Building and Configuration

The NVMe server can be built using the [L4Re](#) build system. Just place this project into your `pkg` directory. The resulting binary is called `nvme-drv`

Starting the service

The NVMe server can be started with Lua like this:

```
local nvme_bus = L4.default_loader:new_channel();
L4.default_loader:start({
  caps = {
    vbus = vbus_nvme,
    svr = nvme_bus:svr(),
  },
},
"rom/nvme-drv");
```

First an IPC gate (`nvme_bus`) is created which is used between the NVMe server and a client to request access to a particular disk or partition. The server-side is assigned to the mandatory `svr` capability of the NVMe server. See the section below on how to configure access to a disk or partition.

The NVMe server needs access to a virtual bus capability (`vbus`). On the virtual bus the NVMe server searches for NVMe compliant storage controllers. Please see `io`'s documentation about how to setup a virtual bus.

Options

In the example above the NVMe server is started in its default configuration. To customize the configuration of the NVMe-server it accepts the following command line options:

- `-v`
Enable verbose mode. You can repeat this option up to three times to increase verbosity up to trace level.
- `-q`
This option enables the quiet mode. All output is silenced.
- `--client <cap_name>`
This option starts a new static client option context. The following `device`, `ds-max` and `readonly` options belong to this context until a new client option context is created.
The option parameter is the name of a local IPC gate capability with server rights.
- `--device <UUID | SN:n<NAMESPACE_ID>>`
This option denotes the partition UUID or serial number of the preceding `client` option followed by a colon, letter 'n' and the identifier of the requested NVMe namespace.
- `--ds-max <max>`
This option sets the upper limit of the number of dataspace the client is able to register with the NVMe server for virtio DMA.
- `--readonly`
This option sets the access to disks or partitions to read only for the preceding `client` option.

- `--nosgl`
This option disables support for SGLs.
- `--nomsi`
This option disables support for MSI interrupts.
- `--nomsix`
This option disables support for MSI-X interrupts.
- `-d <cap_name>, --register-ds <cap_name>` This option registers a trusted dataspace capability. If this option gets used, it is not possible to communicate to the driver via dataspaces other than the registered ones. Can be used multiple times for multiple dataspaces.
The option parameter is the name of a dataspace capability.

Connecting a client

Prior to connecting a client to a virtual block session it has to be created using the following Lua function. It has to be called on the client side of the IPC gate capability whose server side is bound to the NVMe server.

```
create(obj_type, "device=<UUID | SN:n<NAMESPACE_ID>>", "ds-max=<max>", "read-only")
```

- `obj_type`
The type of object that should be created by the server. The type must be a positive integer. Currently the following objects are supported:
 - 0: Virtio block host
- `"device=<UUID | SN>"`
This string denotes either a partition UUID, or a disk serial number the client wants to be exported via the Virtio block interface followed by a colon, letter 'n' and the identifier of the requested NVMe namespace.
- `"ds-max=<max>"`
Specifies the upper limit of the number of dataspaces the client is allowed to register with the NVMe server for virtio DMA.
- `"read-only"`
This string sets the access to disks or partitions to read only for the client.

If the `create()` call is successful a new capability which references an NVMe virtio device is returned. A client uses this capability to communicate with the NVMe server using the Virtio block protocol.

Examples

A couple of examples on how to request different disks or partitions are listed below.

- Request a partition with the given UUID

```
vda1 = nvme_bus:create(0, "ds-max=5", "device=88E59675-4DC8-469A-98E4-B7B021DC7FBE")
```
- Request complete namespace with the given serial number

```
vda = nvme_bus:create(0, "ds-max=4", "device=1234:n1")
```


- A more elaborate example with a static client. The client uses the client side of the `nvme_cl1` capability to communicate with the NVMe server.

```
local nvme_cl1 = L4.default_loader:new_channel();
local nvme_bus = L4.default_loader:new_channel();
L4.default_loader:start({
  caps = {
    vbus = vbus_nvme,
    svr = nvme_bus:svr(),
    cl1 = nvme_cl1:svr(),
  },
},
"rom/nvme-drv --client cl1 --device 88E59675-4DC8-469A-98E4-B7B021DC7FBE --ds-max 5");
```

5.10 Mag, the GUI Multiplexer

Mag is the default multiplexer for graphics hardware. It is not, and does not attempt to be, a fully-fledged window manager.

Command Line Options

As Mag's only command line option it supports loading additional plugins via the application's command line. Plugins must be either a Lua file or a shared library. Shared libraries must be named `libmag-$LIBNAME.so`.

Mag Sessions

Mag provides two types of sessions which a client can create via the Factory interface.

1. Mag client session

A client with a mag client session gets access to the whole screen. The client has to allocate and manage its own buffers and has to position them on the screen on its own. Mag provides the factory to create client sessions via the capability named `mag`.

2. Client framebuffer session

For a client framebuffer session mag allocates a view of the requested size and displays it at the requested coordinates on the screen. Mag provides the factory to create framebuffer sessions via the capability named `SVC`.

The options described below are options the client provides to the `L4::Factory::create()` call. These options influence the appearance and behaviour of the newly created session.

Session Factory Options

As a simple nitpicker clone Mag supports the so-called Xray mode. This mode displays all session labels and draws a colored frame around them. The session that currently has the input focus is highlighted. The Xray mode is activated via the special keys *Scroll* or *NEXTSONG*.

Mag allows to define a text label and a color for all client session types. The label and the color are displayed when Mag enters the Xray mode.

- Label Option: **label**

`l=LABEL, label=LABEL` Set the session's text label to LABEL. The label is restricted to a length of 256 characters.

- Color Option: **col**

`col=COLOR` Set the session's color which is used in Xray mode to tint the session's screen area and the border drawn around it. The argument can be either one of the following letters or a hexadecimal representation of the RGB values.

- `r`, `R` Red color
- `g`, `G` Green color
- `b`, `B` Blue color
- `w`, `W` White color
- `y`, `Y` Yellow color
- `v`, `V` Magenta color

Example

```
-- set label to "Linux" and use a light blue color
fb = mag_client:create(L4.Proto.Goos, "l=Linux", "col=98d9ff");
```

Mag Client Session Options

These options only apply to Mag client sessions.

- Default Background Option: **default-background**

`df1-bg, default-background` Marks this session as the default background.

Mag Client Framebuffer Session Options

These options only apply to Mag client framebuffer sessions.

- Geometry Option: **geometry**

`g=GEOMETRY, geometry=GEOMETRY` Set the session's geometry and position on the screen. GEOMETRY is provided in an X11-style format, e.g. `g=WIDTHxHEIGHT+X_OFFSET+Y_OFFSET`.

- Focus Option: **focus**

`focus` Set the focus to this session.

- Collapsed Option: **shaded**

`shaded` The window is collapsed and only the title bar is visible. The window can be expanded by clicking into the title bar with the middle mouse button. Collapsing and expanding works also independently of this option.

- Fixed Option: **fixed** `fixed` The window cannot be moved on the screen.

- Barheight Option: **barheight**

`barheight=X` Set the height of the title bar in pixels.

Example

```
-- create a window of 640x480 pixels at position (100,100) on the screen.
fb = mag_fb:create(L4.Proto.Goos, "g=640x480+100+100");
```

5.11 eMMC driver

The eMMC driver is a driver for PCI Express eMMC controllers.

Starting the service

The eMMC driver can be started with Lua like this:

```
local emmc_bus = L4.default_loader:new_channel();
L4.default_loader:start({
  caps = {
    vbus = vbus_emmc,
    svr = emmc_bus:svr(),
  },
},
"rom/emmc-drv");
```

First, an IPC gate (`emmc_bus`) is created which is used between the eMMC driver and a client to request access to a particular disk or partition. The server side is assigned to the mandatory `svr` capability of the eMMC driver. See the sector below on how to configure access to a disk or partition.

The eMMC driver needs access to a virtual bus capability (`vbus`). On the virtual bus the eMMC driver searches for eMMC compliant storage controllers. Please see `io`'s documentation about how to setup a virtual bus.

Supported devices

The eMMC driver supports SDHCI and SDHI controllers, in particular

- SDHI interfaces found on RCar3 r8a7795 boards
- SDHCI interfaces found on RPI4
- uSDHCI interfaces found on i.MX8 boards
- uSDHCI interfaces found on the S32G SoC
- the QEMU SD card emulation (SDHCI, see `doc/pcie-ecam.io`),
- the QEMU eMMC emulation (provided by extending the QEMU SD card emulation by `doc/qemu-patch.↵diff`).

Options

In the example above the eMMC driver is started in its default configuration. To customize the configuration of the eMMC driver it accepts the following command line options:

- `-v`
Enable verbose mode. You can repeat this option up to three times to increase verbosity up to trace level.
- `-q`
This option enables the quiet mode. All output is silenced.

- `--disable-mode <mode>`

This option allows to disable certain eMMC/SD card modes from autodetection. The modes `hs26`, `hs52`, `hs52_ddr`, `hs200`, and `hs400` are determined for eMMC devices. The modes `sdr12`, `sdr25`, `sdr50`, `sdr104`, `ddr50` are determined for SD card devices. This option can be specified multiple times to disable multiple modes.

- `--max-seg <number>`

Maximum number of segments per request. This number is announced to the virtio interface and is also relevant for the required bounce buffer size, see below. Default is 64.

- `--client <cap_name>`

This option starts a new static client option context. The following `device`, `ds-max`, `readonly` and `dma-map-all` options belong to this context until a new client option context is created.

The option parameter is the name of a local IPC gate capability with server rights.

- `--device <UUID>`

This option denotes the partition UUID of the partition to be exported for the client specified in the preceding `client` option.

- `--ds-max <max>`

This option sets the upper limit of the number of dataspace the client is able to register with the eMMC driver for virtio DMA.

- `--readonly`

This option sets the access to disks or partitions to read only for the preceding `client` option.

- `--dma-map-all`

Map the entire client dataspace into the DMA space at the first I/O request and never unmap the dataspace until the client is destroyed. The default behavior is to map the relevant part of the dataspace before an I/O request and unmap it after the request.

Connecting a client

Prior to connecting a client to a virtual block session it has to be created using the following Lua function. It has to be called on the client side of the IPC gate capability whose server side is bound to the eMMC driver.

```
create(obj_type, "device=<UUID>", "ds-max=<max>")
```

- `obj_type`

The type of object that should be created by the driver. The type must be a positive integer. Currently the following objects are supported:

- 0: Virtio block host

- `"device=<UUID>"`

This string denotes a partition UUID the client wants to be exported via the Virtio block interface.

- `"ds-max=<max>"`

Specifies the upper limit of the number of dataspace the client is allowed to register with the eMMC driver for virtio DMA.

- `"readonly"`

This option sets the access to disks or partitions to read only for this client connection.

- `"dma-map-all"`

Map the entire client dataspace into the DMA space at the first I/O request and never unmap the dataspace until the client is destroyed. The default behavior is to map the relevant part of the dataspace before an I/O request and unmap it after the request.

If the `create()` call is successful a new capability which references an eMMC virtio driver is returned. A client uses this capability to communicate with the eMMC driver using the Virtio block protocol.

Recognized capabilities

The driver makes use of certain capabilities:

- `vbus`

Required for finding the device which should be driven by this driver.

- `bbds`

Only used by the SDHCI driver.

Certain SDHCI devices cannot handle DMA requests with DMA buffers beyond 4GiB. The provided dataspace is used as bounce buffer if the driver detects that a certain request needs it.

Note: The bounce buffer needs to be able to hold the memory for an entire read/write request. That means that the buffer is divided into the number of maximum segments (see `--max-seg` parameter). **Note:** The physical memory of the provided dataspace must be contiguous.

- `sdhci_adma_buf`

Only used by the SDHCI driver.

Page (4096 bytes) for storing DMA descriptors for the SDHCI driver. If this capability is not provided, the driver will allocate an arbitrary page.

- `bcm2835_mbox_mem`

Only used by the SDHCI driver when attaching to an bcm2711-compatible device.

Page (4096 bytes) for storing bcm2835 mbox messages. The firmware mbox is used to perform voltage switching for certain SD card configurations. If this capability is not provided, the driver will allocate an arbitrary page.

Examples

A couple of examples on how to request different disks or partitions are listed below.

- Request a partition with the given UUID

```
vdal = emmc_bus:create(0, "ds-max=5", "device=AFFA05B0-9379-480E-B9C6-5FF57FB1D194")
```

- A more elaborate example with a static client. The client uses the client side of the `emmc_cl1` capability to communicate with the eMMC driver.

```
local emmc_cl1 = L4.default_loader:new_channel();
local emmc_bus = L4.default_loader:new_channel();
L4.default_loader:start({
  caps = {
    vbus = vbus_emmc,
    svr = emmc_bus:svr(),
    cl1 = emmc_cl1:svr(),
  },
},
"rom/emmc-drv --client cl1 --device 88E59675-4DC8-469A-98E4-B7B021DC7FBE --ds-max 5");
```

- Accessing a device from QEMU:

The file `pcie-ecam.io` contains an IO config file which is able to use the QEMU PCI controller to search for attached eMMC devices.

- eMMC emulation with QEMU:

The attached patch extends QEMU SD card emulation to emulate eMMC devices. After applying the patch and recompiling QEMU, attach the following parameters to your QEMU command line (assuming that `$HOME/foobar.img` is the eMMC medium):

```
-drive id=sd_disk,file=$(HOME)/foobar.img,if=none,format=raw \
-device sdhci-pci,id=sdhci \
-device sd-card,drive=sd_disk,spec_version=3,emmc=on
```

5.12 Cons, the Console Multiplexer

`cons` is an interactive multiplexer for console input and output. It buffers the output from different L4 clients and allows to switch between them to redirect input.

Multiplexers and Frontends

`cons` is able to connect multiple clients to multiple console I/O servers. All clients are connected to all configured multiplexers

Multiplexers and frontends come in pairs where the actual I/O is handled by the frontend. From the point-of-view of `cons`, a frontend consists of an IPC channel to a server that speaks an appropriate server protocol. By default the `L4.Env.log` capability is used. Only frontends that speak the `L4::Vcon` protocol are supported.

A multiplexer handles and routes the I/O of each client to and from its respective frontend.

Each client's console settings (e.g. color, visibility) apply to all multiplexers and cannot be changed individually. A user can connect to all clients through all multiplexers. It is not possible to assign individual clients to distinct multiplexers.

Client sessions

For clients `cons` implements the `L4::Vcon` and the Virtio console interface. A client session can be requested through a `create` call to `cons`' factory. `cons` binds its factory capability to the `cons` capability. See the example below on how this can be set up.

Starting the service

The `cons` server can be started with Lua like this:

```
local log_server = L4.default_loader:new_channel()
L4.default_loader:start({
  caps = {
    cons = log_server:svr(),
  },
  log = L4.Env.log,
},
"rom/cons")
```

First an IPC gate (`log_server`) is created which is used between the `cons` server and a client to request a new session. The server side is assigned to the mandatory `cons` capability of `cons`. This example explicitly assigns the kernel's log capability (`L4.Env.log`) to `cons`' `log` capability in order to allow `cons` to provide input and output for its clients. The default log factory (usually provided by `moe`) doesn't provide input capabilities.

Command Line Options

cons accepts the following command line switches:

- `-a, --show-all`
Initially show output from all clients.
- `-B <size>, --defaultbufsize <size>`
Default buffer size per client in bytes. Default: 40960
- `-c <client>, --autoconnect <client>`
Automatically connect to the client with the given name. That means that output of this client will be visible and input will be routed to it.
- `-f <cap>, --frontend <cap>`
Set the frontend for the current multiplexer. Output for the multiplexer is then sent to the capability with the given name `<cap>`. The server connected to the capability needs to understand the [L4::Vcon](#) protocol.
- `-k, --keep`
Keep the console buffer when a client disconnects.
- `-l, --no-line-buffering`
By default, merge the client output to entire lines. If the client writes characters without a final newline, the following client output is merged with the current line content. Specifying this switch disables the line buffered mode by default.
- `--line-buffering-ms <timeout>`
Timeout in milliseconds before buffered client output is written even without a newline. Default value is 50.
- `-m <prompt name>, --mux <prompt name>`
Add a new multiplexer named `<prompt name>`. This is necessary if output should be sent to different frontends. This option must be used in conjunction with the `-f` frontend option
- `-n, --defaultname`
Default name for the multiplexer prompt. Default: `cons`.
- `-t, --timestamp`
Prefix the output with timestamps.

Connecting a client

```
create(backend_type, ["client_name"], ["color"], ["option"] [,"option"] ...)
```

- `backend_type`
The type of backend that should be created for the client. The type is a positive integer and currently the following types are supported:
 - `L4.Proto.Log`: [L4::Vcon](#) client
 - `1`: Virtio console client

cons accepts the following per-client options:

- `bufsz=n`
Use a buffer of `n` bytes for this client, deviating from the default buffer size.
- `keep / no-keep`
The console buffer is kept / thrown away when the client disconnects.
- `key=<key>`
Assign `<key>` as keyboard shortcut to this client.
- `line-buffering / no-linux-buffering`
Line buffering is enabled / disabled for this client.
- `show / hide`
Output from this client is initially shown / hidden.
- `timestamp / no-timestamp`
Do / do not prefix the output of this client with timestamps.

5.13 AHCI driver

The AHCI driver is a driver for PCI serial ATA host controllers.

The AHCI driver is capable of exposing entire disks (by serial number) or individual partitions (by their partition UUID) of a hard drive to clients via the Virtio block interface.

The driver consists of two parts. The first one is the hardware driver itself that takes care of the communication with the underlying hardware and interrupt handling. The second part implements a virtual block device and is responsible to communicate with clients. The virtual block device translates commands it receives into AHCI requests and issues them to the hardware driver.

The AHCI driver allows both statically and dynamically configured clients. A static configuration is given priority over dynamically connecting clients and configured while the service starts. Dynamic clients can connect and disconnect during runtime of the AHCI driver.

Building and Configuration

The AHCI driver can be built using the [L4Re](#) build system. Just place this project into your `pkg` directory. The resulting binary is called `ahci-drv`

Starting the service

The AHCI driver can be started with Lua like this:

```
local ahci_bus = L4.default_loader:new_channel();
L4.default_loader:start({
  caps = {
    vbus = vbus_ahci,
    svr = ahci_bus:svr(),
  },
},
"rom/ahci-drv");
```

First an IPC gate (`ahci_bus`) is created which is used between the AHCI driver and a client to request access to a particular disk or partition. The server-side is assigned to the mandatory `svr` capability of the AHCI driver. See the section below on how to configure access to a disk or partition.

The `ahci` driver needs access to a virtual bus capability (`vbus`). On the virtual bus the AHCI driver searches for AHCI 1.0 compliant storage controllers. Please see `io`'s documentation about how to setup a virtual bus.

Options

In the example above the ahci driver is started in its default configuration. To customize the configuration of the ahci-driver it accepts the following command line options:

- `-A`
Disable check for address width of the device. Only do this if all physical memory is guaranteed to be below 4GB.
- `-v`
Enable verbose mode. You can repeat this option up to three times to increase verbosity up to trace level.
- `-q`
This option enables the quiet mode. All output is silenced.
- `--client <cap_name>`
This option starts a new static client option context. The following `device`, `ds-max` and `readonly` options belong to this context until a new client option context is created.
The option parameter is the name of a local IPC gate capability with server rights.
- `--device <UUID | SN>`
This option denotes the partition UUID or serial number of the preceding `client` option.
- `--ds-max <max>`
This option sets the upper limit of the number of dataspaces the client is able to register with the AHCI driver for virtio DMA.
- `--slot-max <max>`
This option defines the maximum number of requests a single client may have in parallel running on the device. If a positive number is given, then this is considered the absolute number of slots to be used. If a negative number is given, then the client may use all available slots except the number given. In any case, a client gets at least 1 slot and at most the number of slots available in hardware. This parameter is only valid when a client accesses a partition and ignored otherwise.
- `--readonly`
This option sets the access to disks or partitions to read only for the preceding `client` option.

Connecting a client

Prior to connecting a client to a virtual block session it has to be created using the following Lua function. It has to be called on the client side of the IPC gate capability whose server side is bound to the ahci driver.

```
create(obj_type, "device=<UUID | SN>", "ds-max=<max>"[, "slot-max=<max>"])
```

- `obj_type`
The type of object that should be created by the driver. The type must be a positive integer. Currently the following objects are supported:
 - 0: Virtio block host
- `"device=<UUID | SN>"`
This string denotes either a partition UUID or a disk serial number the client wants to be exported via the Virtio block interface.

- "ds-max=<max>"

Specifies the upper limit of the number of dataspace the client is allowed to register with the AHCI driver for virtio DMA.

- "slot-max=<max>"

Specifies the maximum number of requests that will be processed in parallel by the AHCI device. See `--slot-max` option above for details.

If the `create()` call is successful a new capability which references an AHCI virtio driver is returned. A client uses this capability to communicate with the AHCI driver using the Virtio block protocol.

Examples

A couple of examples on how to request different disks or partitions are listed below.

- Request a partition with the given UUID

```
vda1 = ahci_bus:create(0, "ds-max=5", "device=88E59675-4DC8-469A-98E4-B7B021DC7FBE")
```

- Request complete disk with the given serial number

```
vda = ahci_bus:create(0, "ds-max=4", "device=QM00005")
```

- A more elaborate example with a static client. The client uses the client side of the `ahci_cl1` capability to communicate with the AHCI driver.

```
local ahci_cl1 = L4.default_loader:new_channel();
local ahci_bus = L4.default_loader:new_channel();
L4.default_loader:start({
  caps = {
    vbus = vbus_ahci,
    svr = ahci_bus:svr(),
    cl1 = ahci_cl1:svr(),
  },
},
"rom/ahci-drv --client cl1 --device 88E59675-4DC8-469A-98E4-B7B021DC7FBE --ds-max 5");
```

Chapter 6

uvmm_dtg The device tree generator for Uvmm

A virtual machine in Uvmm is configured with a device tree that contains information about the VMs resources, memory layout, virtual CPUs and peripheral devices.

Uvmm_dtg is a tool to generate such a device tree at runtime according to its command line.

Usage in L4Re

Example lua script for Ned:

```
-- Create DS holding device tree
local dt = L4.Env.user_factory:create(L4.Proto.Dataspace, 4 * 1024):m("rw");

-- Start the generator
L4.default_loader:start(
{
  caps = { dt = dt },
}, "rom/uvmm_dtg dt"):wait();

-- Start uvmm
vmm.start_vm
{
  ...
  ext_caps = { dt = dt },
  fdt = "dt",
  ...
}
```

Please notice the `:wait()` when starting `uvmm_dtg`. This makes Ned pause until `uvmm_dtg` has exited and put the device tree into the dataspace such that Uvmm can commence.

Usage

`uvmm_dtg [OPTION]... <file>|--`

- `--` print to stdout
- `file`

On [L4Re](#), the string given as `<file>` is interpreted as a named capability which needs to be backed by a sufficiently large Dataspace. On Linux, a file with the given name is created. In both cases, `uvmm_dtg` will output into the named file.

Options

- `-h`
Show help.
- `--arch <target architecture>`
Select the target architecture. Valid options are `x86`, `x86_64`, `arm32`, `arm64`, `mips32` and `mips64`.
- `--format <format>`
Select the output format. Available formats are: `txt`: The device tree will be printed as plain text (`dtb`).
`bin`: The device tree will be output as binary (`dtb`).
- `--mem-base <membase>`
Configure the start of the memory distribution. `membase` can be defined in both decimal and hex notations.
`uvmm_dtg` rounds the given base up to the platforms page size.
This value can be overridden by memory devices with fixed addresses.
- `--device <devicename:[Option1,Option2=value,Option3=value,...]>`
This configures a device.
To get a list of supported devices, use `--device help`.
To get help for a specific device, use `--device devicename:help`.

Chapter 7

Bootstrap, the L4 kernel bootstrapper

Bootstrap Command Line Options

`bootstrap` and the kernel can be configured through command line switches. `bootstrap` is responsible for parsing both command lines: `bootstrap` options are the ones directly given to `bootstrap`, whereas kernel options are those directly given to the kernel, respectively.

When using Multiboot boot, the first module directly after `bootstrap` is considered the kernel: An example using GRUB2 might look like this:

```
multiboot /path/to/bootstrap bootstrap -bs-boolean-opt -bs-opt-with-argument=foo
module /path/to/fiasco fiasco -kernel-opt-with-argument=bar -kernel-boolean-opt
```

Note

The exact way to provide the command line to `bootstrap` is platform-dependent. On platforms supporting booting via Multiboot Specification the command line can be specified in the boot configuration (currently x86/amd64 only, e.g. using GRUB) whereas other platforms need the command line to be compiled into the `bootstrap` binary.

Platforms utilising flattened device trees usually also allow specifying a command line through the DT. This mechanism is **not** currently supported in `bootstrap`!

Note

`bootstrap` will ignore options it does not understand. For most cases, this also holds true if an option's arguments cannot be understood.

bootstrap options

Command line options directly understood by `bootstrap` itself are as follows (passed via `bootstrap` command line):

- `-comirq=<irqno>` (x86/amd64 only)

If serial logging is enabled (default on), `<irqno>` defines which IRQ to use for serial port communication. This option is ignored if `-noserial` is also specified (see below).

- `-comport=<portspec>` (x86/amd64 only)

If serial logging is enabled (default on), `<portspec>` defines which serial port to use, being one of:

- `<number>`
Use legacy port `<number>`, e.g. use `-comport=1` for the port commonly known as *COM1*, or
- `pci:<card>:<port>`
Use serial port number `<port>` at PCI card number `<card>`, e.g. use `-comport=pci:0:1` for the second port on the first PCI card.
- `pci:probe`
Use this to have `bootstrap` autodiscover all PCI serial lines. On each discovered line a message will be displayed, telling the correct `<portspec>` to be given to use the respective line.

Note

`bootstrap` does not support specifying the serial port's baudrate and always uses 115200 bps.

- `-noserial`

Disable serial logging.

- `-wait`

Wait for key press at early startup.

Note

This is not to be confused with the kernel's `-wait` option, see below.

- `-maxmem=<mbytes>`

Limit the available memory to at most `<mbytes>` MiB.

- `-mem=<size>@<offset>`

Add a region of memory of `<size>` at `<offset>` to the system's memory map. Both `<size>` and `<offset>` may be suffixed by either G, M, or K/k to denote GiB, MiB, or KiB, respectively.

This option may be specified multiple times. If the option is not given, a platform-specific method for determining the memory layout will be used, if available.

- `-presetmem=<intval>`

Initialise memory regions with `<intval>` before starting the kernel.

- `-modaddr=<paddr>`

Relocate modules to the physical address `<paddr>`. Use this when utilising a version of GRUB that lacks support for the `modaddr` command.

Kernel Options

Command line options for the kernel (passed on kernel command line, i.e. to first module) `bootstrap` understands are as follows.

Note

Availability of individual options might depend on used platform and/or actual kernel configuration.

- `-wait`
Enter debugger directly after startup, prior to executing any task.
- `-serial_esc`
Enable entering the debugger over serial line by pressing `Esc`.
- `-noserial`
Disable serial logging.

Note

If this is given as kernel command line argument, it does not affect `bootstrap` but only the kernel.

- `-noscreen`
Disable VGA console.
- `-esc`
Enable entering the debugger by pressing `Esc` on attached keyboard.
- `-nojdb`
Disable the kernel debugger.
- `-nohlt`
Enable quirk for broken HLT instruction.
- `-apic`
Use Advanced Programmable Interrupt Controller (APIC) if available and known to be well-behaving.
- `-loadcnt`
Use load counter for performance counting.
- `-watchdog`
Enable watchdog timer.
- `-irq0`
Allow IRQ 0 to be used by userland. This enables some sanity checks to ensure IRQ 0 is not used by the kernel, e.g. for profiling or scheduling purposes. This only has an effect on x86.
- `-nosfn`
Disable SFN (special fully nested) mode of interrupt controller. This only has an effect on x86 with PIC8259 interrupt controller.
- `-jdb_never_stop`
Prevent system from stopping to enter JDB. This only has an effect on x86.
- `-kmemsize=<KB>`
Reserve <KB> KiB of memory for the kernel.

- `-tbuf_entries=<number>`
Specify the `<number>` of trace buffer entries.
- `-out_buf=<length>`
Specify length of console buffer to be `<length>` bytes.
- `-jdb_cmd=<ctrlseq>`
Execute JDB command sequence `<ctrlseq>` right after start-up. If `-wait` is also given, `<ctrlseq>` is executed right before entering JDB.

Module options

Bootstrap supports module attributes for `sigma0` and the `roottask`. They need to be specified in `modules.list`, e.g.:

```
sigma0[attr:nodes=4-7] ...
```

Attributes are not supported when using multi-boot on platforms that support it. The following attributes are supported:

- `nodes`

This is a colon separated list of AMP node ranges. A range can also be a single number. Examples:

- `nodes=1` – Only node 1
- `nodes=1-3` – Nodes 1 to 3 (inclusive)
- `nodes=0:2-3` – Nodes 0, 2 and 3

If not present, the `sigma0/roottask` module is applicable to all AMP nodes.

- `reloc`

Normally the `sigma0` or `roottask` images are loaded at the preferred load address if the RAM is available at the desired location. If this is not possible, they will be relocated to some free RAM region. Setting the "reloc" module attribute to a non-empty string will always request the dynamic relocation.

This attribute can be used on no-MMU systems to maximize the size of contiguous free RAM regions.

Chapter 8

Deprecated List

Global `L4::Rcv_endpoint::bind_thread (Ipc::Cap< Thread > t, I4_umword_t label)`

Use `bind_snd_destination()` instead.

Global `L4_CAP_SIZE`

Superseded by `L4_CAP_OFFSET`.

Global `I4_kip_clock_lw (I4_kernel_info_t const *kip) L4_NOTHROW`

Use `I4_kip_clock()` instead.

Global `L4Re::Util::Registry_server< LOOP_HOOKS >::Registry_server (I4_utcb_t *, L4::Cap< L4::Thread > server, L4::Cap< L4::Factory > factory)`

Note that this variant of the constructor is deprecated, please do not supply the UTCB pointer, it's not used.

Global `I4util_kip_for_each_feature (s)`

Use `I4_kip_for_each_feature()`.

Global `I4util_kip_kernel_has_feature (I4_kernel_info_t const *k, char const *str)`

Use `I4_kip_kernel_has_feature()`.

Global `I4util_micros2I4to (I4_uint64_t us) L4_NOTHROW`

Use `I4_timeout_from_us()`.

Chapter 9

Topic Index

9.1 Topics

Here is a list of all topics with brief descriptions:

Base API	??
Basic Macros	??
Fiasco extensions	??
Kernel Debugger	??
Kernel Information Dump	??
Kernel Tracing	??
Flexpages	??
C++ IPC Interface Definition.	??
Internal Helpers	??
Cache Consistency	??
Memory related	??
Error codes	??
Object Invocation	??
Message Items	??
Timeouts	??
Error Handling	??
Realtime API	??
Message Tag	??
Virtual Registers (UTCBs)	??
Message Registers (MRs)	??
Exception registers	??
Buffer Registers (BRs)	??
Thread Control Registers (TCRs)	??
ARM Virtual Registers (UTCB)	??
ARM64 Virtual Registers (UTCB)	??
AMD64 Virtual Registers (UTCB)	??
x86 Virtual Registers (UTCB)	??
Kernel Objects	??
IPC-Gate API	??
DMA space	??
L4 kernel object type information	??
Factory	??
Virtual Machines	??
VM API for SVM	??

VM API for VMX	??
VM API for TZ	??
Interrupt controller	??
IRQs	??
Platform Control C API	??
Scheduler	??
Kernel-provided semaphore	??
Task	??
Thread	??
Thread control	??
vCPU API	??
Thread groups	??
Virtual Console	??
Kernel Interface Page	??
Memory descriptors (C version)	??
Capabilities	??
Memory operations.	??
Integer Types	??
EDID parsing functionality	??
IO interface	??
IPC Helpers	??
IRQ handling library	??
Interface using direct functionality.	??
Interface using direct functionality.	??
Interface for asynchronous ISR handlers.	??
Interface for asynchronous ISR handlers with a given IRQ capability.	??
L4 IPC Opcodes	??
L4 VIRTIO Interface	??
L4 VIRTIO Transport Layer	??
L4 VIRTIO Block Device	??
L4 VIRTIO Input Device	??
L4 VIRTIO Network Device	??
L4 Vbus functions	??
L4vbus GPIO functions	??
L4vbus PCI functions	??
L4vbus power management functions	??
L4Re C Interface	??
L4Re Util C Interface	??
Dataspace interface	??
Debug interface	??
DMA Space Interface	??
Event interface	??
Log interface	??
Memory allocator	??
Namespace interface	??
Parent interface	??
Region map interface	??
Capability allocator	??
Kumem allocator utility	??
Video API	??
Initial Environment	??
L4Re C++ Interface	??
L4Re Util C++ Interface	??
L4Re Capability API	??
Kumem utilities	??
Console API	??

Debugging API	??
L4Re ELF Auxiliary Information	??
Event API	??
Auxiliary data	??
Logging interface	??
Name-space API	??
Parent API	??
L4Re Protocol identifiers	??
Region map API	??
Video API	??
C++ Exceptions	??
Vbus API	??
L4SHM-based ring buffer implementation	??
Sender	??
Receiver	??
Internal	??
Shared Memory Library	??
Chunks	??
Producer	??
Consumer	??
Signals	??
Producer	??
Consumer	??
Sigma0 API	??
Internal constants	??
Small C++ Template Library	??
The L4Re IPC Framework	??
Server-Side IPC framework	??
Utility Functions	??
Bitmap graphics and fonts	??
Functions for rendering bitmap data in frame buffers	??
Functions for rendering bitmap fonts to frame buffers	??
CPU related functions	??
Timestamp Counter	??
Atomic Instructions	??
Internal functions	??
Bit Manipulation	??
ELF binary format	??
Kernel Interface Page API	??
Comfortable Command Line Parsing	??
Random number support	??
Low-Level Thread Functions	??
IA32 Port I/O API	??
Virtio Net Switch	??
vCPU Support Library	??
Extended vCPU support	??

Chapter 10

Namespace Index

10.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

cxx	Our C++ library	??
cxx::Bits	Internal helpers for the cxx package	??
L4	L4 low-level kernel interface	??
L4::lpc	IPC related functionality	??
L4::lpc::Msg	IPC Message related functionality	??
L4::lpc_svr	Helper classes for L4::Server instantiation	??
L4::Typeid	Definition of interface data-type helpers	??
L4::Types	L4 basic type helpers for C++	??
L4Re	L4Re C++ Interfaces	??
L4Re::Util	Documentation of the L4 Runtime Environment utility functionality in C++	??
L4Re::Vfs	Virtual file system for interfaces in POSIX libc	??
L4vbus	C++ interface of the Vbus API.	??
L4virtio	L4-VIRTIO Transport C++ API	??

Chapter 11

Hierarchical Index

11.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Block_device::Device_discard_feature	??
Block_device::Device_mgr< DEV, FACTORY, SCHEDULER >	??
Block_device::Device_with_notification_domain< DEV >	??
Block_device::Dma_region_info	??
Block_device::Impl::Partitioned_device_discard_mixin< PART_DEV, BASE_DEV, bool >	??
Block_device::Impl::Partitioned_device_discard_mixin< PART_DEV, BASE_DEV, true >	??
Block_device::Impl::Partitioned_device_discard_mixin< Partitioned_device< Device >, Device >	??
Block_device::Partitioned_device< BASE_DEV >	??
Block_device::Inout_block	??
Block_device::Inout_memory< DEV >	??
Block_device::Inout_memory< Device_type >	??
Block_device::Mem_region_info	??
Block_device::Notification_domain	??
Block_device::Partition_info	??
Block_device::Partition_reader< DEV >	??
Block_device::Pending_request	??
Block_device::Scheduler_base< DEV >	??
Block_device::Rr_scheduler< DEV >	??
cxx::arith::Ld< V >	??
cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >	??
cxx::Base_slab< sizeof(Type), L4_PAGESIZE, 2, New_allocator >	??
cxx::Slab< Type, Slab_size, Max_free, Alloc >	??
cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >	??
cxx::Base_slab_static< sizeof(Type), L4_PAGESIZE, 2, New_allocator >	??
cxx::Slab_static< Type, Slab_size, Max_free, Alloc >	??
cxx::Bitfield< T, LSB, MSB >	??
cxx::Bitfield< T, LSB, MSB >::Value_base< TT >	??
cxx::Bitfield< T, LSB, MSB >::Value< Base_type & >	??
cxx::Bitfield< T, LSB, MSB >::Value< Base_type volatile & >	??
cxx::Bitfield< T, LSB, MSB >::Value< Base_type const >	??
cxx::Bitfield< T, LSB, MSB >::Value_unshifted< Base_type & >	??
cxx::Bitfield< T, LSB, MSB >::Value_unshifted< Base_type volatile & >	??
cxx::Bitfield< T, LSB, MSB >::Value_unshifted< Base_type const >	??
cxx::Bitfield< T, LSB, MSB >::Value< TT >	??

cxx::Bitfield< T, LSB, MSB >::Value_unshifted< TT >	??
cxx::Bitmap_base	??
cxx::Bitmap< BITS >	??
cxx::Bitmap_base::Bit	??
cxx::Bitmap_base::Char< BITS >	??
cxx::Bitmap_base::Word< BITS >	??
cxx::Bits::Avl_map_get_key< KEY_TYPE >	??
cxx::Bits::Avl_set_get_key< KEY_TYPE >	??
cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >	??
cxx::Avl_map< Region, Hdlr, cxx::Lt_functor, Alloc >	??
cxx::Avl_set< ITEM_TYPE, COMPARE, ALLOC >	??
cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Node	??
cxx::Bits::Base_avl_set< Pair< KEY_TYPE, DATA_TYPE >, Lt_functor< KEY_TYPE >, New_allocator, Bits::Avl_map_get_key< KEY_TYPE > >	??
cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC >	??
cxx::Bits::Basic_list< POLICY >	??
cxx::H_list< Timeout >	??
cxx::H_list< Observer >	??
cxx::H_list< cxx::Base_slab::Slab_i >	??
cxx::S_list< T, POLICY >	??
cxx::H_list< T, POLICY >	??
cxx::Bits::Basic_list< Bits::Basic_list_policy< T, H_list_item_t< T > > >	??
cxx::H_list< T, Bits::Basic_list_policy< T, H_list_item_t< T > > >	??
cxx::H_list_t< T >	??
cxx::Bits::Basic_list< Bits::Basic_list_policy< T, S_list_item > >	??
cxx::S_list< T, Bits::Basic_list_policy< T, S_list_item > >	??
cxx::S_list< T, POLICY >	??
cxx::Bits::Basic_list< Bits::Basic_list_policy< Weak_ref_base, H_list_item_t< Weak_ref_base > > >	??
cxx::H_list< Weak_ref_base, Bits::Basic_list_policy< Weak_ref_base, H_list_item_t< Weak_ref_base > > >	??
cxx::H_list_t< Weak_ref_base >	??
cxx::Weak_ref_base::List	??
cxx::Bits::Bst< Node, Get_key, Compare >	??
cxx::Avl_tree< Entry, Names_get_key >	??
cxx::Avl_tree< _Node, GET_KEY, COMPARE >	??
cxx::Avl_tree< Node, Get_key, Compare >	??
cxx::Bits::Bst< _Node, Bits::Avl_map_get_key< KEY_TYPE >, Lt_functor< KEY_TYPE > > >	??
cxx::Avl_tree< _Node, Bits::Avl_map_get_key< KEY_TYPE >, Lt_functor< KEY_TYPE > > >	??
cxx::Bits::Bst< _Node, Bits::Avl_set_get_key< ITEM_TYPE >, Lt_functor< ITEM_TYPE > > >	??
cxx::Avl_tree< _Node, Bits::Avl_set_get_key< ITEM_TYPE >, Lt_functor< ITEM_TYPE > > >	??
cxx::Bits::Bst_node	??
cxx::Avl_tree_node	??
cxx::Bits::Direction	??
cxx::Bits::Smart_ptr_list< ITEM >	??
cxx::Bits::Smart_ptr_list_item< T, STORE_T >	??
cxx::Bits::Smart_ptr_list_item< T, cxx::Ref_ptr< T > >	??
cxx::Ref_obj_list_item< Connection >	??
cxx::Ref_obj_list_item< T >	??
cxx::H_list_item_t< ELEM_TYPE >	??
cxx::H_list_item_t< void >	??
L4::lpc_svr::Timeout	??
Block_device::Errand::Errand	??
Block_device::Errand::Poll_errand	??
cxx::H_list_item_t< Weak_ref_base >	??

cxx::Weak_ref_base	??
cxx::Weak_ref< T >	??
cxx::List< D, Alloc >	??
cxx::List< D, Alloc >::Iter	??
cxx::List_alloc	??
cxx::List_item	??
cxx::List_item::Iter	??
cxx::List_item::T_iter< E >	??
cxx::List_item::T_iter< T, Poly >	??
cxx::Lt_functor< Obj >	??
cxx::New_allocator< _Type >	??
cxx::Nothrow	??
cxx::Pair< First, Second >	??
cxx::Pair_first_compare< Cmp, Typ >	??
cxx::Ref_ptr< T, CNT >	??
cxx::Ref_ptr< Device_type >	??
cxx::Ref_ptr< L4Re::Vfs::File >	??
cxx::Ref_ptr< Mount_tree >	??
cxx::Ref_ptr< T, CNT >	??
cxx::static_vector< T, IDX >	??
cxx::String	??
L4Re::Util::Names::Name	??
Elf32_Auxv	??
Elf32_Dyn	??
Elf32_Ehdr	??
Elf32_Phdr	??
Elf32_Rel	??
Elf32_Rela	??
Elf32_Shdr	??
Elf32_Sym	??
Elf64_Auxv	??
Elf64_Dyn	??
Elf64_Ehdr	??
Elf64_Phdr	??
Elf64_Rel	??
Elf64_Rela	??
Elf64_Shdr	??
Elf64_Sym	??
gfxbitmap_offset	??
L4::Alloc_list	??
L4::Basic_registry	??
L4Re::Util::Object_registry	??
L4::Cap_base	??
L4::Cap< A >	??
L4::Cap< L4::Rcv_endpoint >	??
L4::Cap< L4Re::Rm >	??
L4::Cap< L4::Irq >	??
L4::Cap< L4Re::Namespace >	??
L4::Cap< L4Re::Dataspace >	??
L4::Cap< L4::Vcon >	??
L4::Cap< L4::Semaphore >	??
L4::Cap< L4::Thread >	??
L4::Cap< L4::Factory >	??
L4::Cap< L4Re::Video::Goos >	??
L4::Cap< L4vbus::Vbus >	??
L4::Cap< L4virtio::Device >	??
L4::Smart_cap< T, Smart_count_cap< L4_FP_ALL_SPACES > >	??

L4::Smart_cap< T, Smart_count_cap< L4_FP_DELETE_OBJ > >	??
L4::Cap< T >	??
L4::Smart_cap< T, SMART >	??
L4::Epiface	??
L4::Epiface_t0< void, Epiface >	??
L4::Irqep_t< Irq_object >	??
L4::Irqep_t< Host_irq >	??
L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::Host_irq	??
L4virtio::Svr::Virtio_i2c< Request_handler, Epiface >::Host_irq	??
L4virtio::Svr::Virtio_rng< Rnd_state, Epiface >::Host_irq	??
L4::Irqep_t< Del_cap_irq >	??
L4::Irqep_t< Kick_irq >	??
L4::Irqep_t< Derived, BASE, bool >	??
L4::Epiface_t0< L4virtio::Device, L4::Epiface >	??
L4::Epiface_t< Virtio_client< DEV >, L4virtio::Device >	??
L4::Epiface_t< Block_dev< Ds_data >, L4virtio::Device >	??
L4::Epiface_t< Virtio_gpio< Request_handler, L4virtio::Device >, L4virtio::Device >	??
L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >	??
L4::Epiface_t< Virtio_i2c< Request_handler, L4virtio::Device >, L4virtio::Device >	??
L4virtio::Svr::Virtio_i2c< Request_handler, Epiface >	??
L4::Epiface_t< Virtio_rng< Rnd_state >, L4virtio::Device >	??
L4virtio::Svr::Virtio_rng< Rnd_state, Epiface >	??
L4::Epiface_t< Virtio_net, L4virtio::Device >	??
Virtio_net	??
L4virtio_port	??
L4::Epiface_t0< IFACE, L4::Epiface >	??
L4::Epiface_t< Derived, IFACE, BASE, bool >	??
L4::Epiface_t0< L4::Kobject, L4::Epiface >	??
L4::Epiface_t< Null_handler, L4::Kobject >	??
L4::Epiface_t0< L4::Factory, L4::Epiface >	??
L4::Epiface_t< Switch_factory, L4::Factory >	??
Switch_factory	??
L4::Epiface_t0< Virtio_net_switch::Statistics_if, L4::Epiface >	??
L4::Epiface_t< Stats_reader, Virtio_net_switch::Statistics_if >	??
L4::Epiface_t0< RPC_IFACE, BASE >	??
L4::Server_object	??
L4::Server_object_t< Kobject >	??
L4::Irq_handler_object	??
L4::Server_object_t< IFACE, L4::Server_object >	??
L4::Server_object_x< Derived, IFACE, BASE >	??
L4::Server_object_t< IFACE, BASE >	??
L4::Server_object_x< Derived, IFACE, BASE >	??
L4::Epiface_t0< IFACE, BASE >	??
L4::Epiface_t0< void, BASE >	??
L4::Exception_tracer	??
L4::Base_exception	??
L4::Invalid_capability	??
L4::Runtime_error	??
L4::Bounds_error	??
L4::Com_error	??
L4::Element_already_exists	??
L4::Element_not_found	??
L4::Out_of_memory	??
L4::Unknown_error	??
L4::Factory::Lstr	??
L4::Factory::Nil	??

L4::Factory::S	??
L4::IOModifier	??
L4::lpc::Array_in_buf< ELEM_TYPE, LEN_TYPE, MAX >	??
L4::lpc::Array_ref< ELEM_TYPE, LEN_TYPE >	??
L4::lpc::Array_ref< A, LEN >	??
L4::lpc::Array< A, LEN >	??
L4::lpc::Array< A, LEN > &	??
L4::lpc::Array_ref< char const, unsigned long >	??
L4::lpc::Array< char const, unsigned long >	??
L4::lpc::Array_ref< ELEM_TYPE, Array_len_default >	??
L4::lpc::Array< ELEM_TYPE, LEN_TYPE >	??
L4::lpc::Array_ref< X, LEN_TYPE >	??
L4::lpc::Array< X, LEN_TYPE >	??
L4::lpc::As_value< T >	??
L4::lpc::Call	??
L4::lpc::Call_t< RIGHTS >	??
L4::lpc::Call_zero_send_timeout	??
L4::lpc::Cap< T >	??
L4::lpc::Gen_fpage	??
L4::lpc::Rcv_fpage	??
L4::lpc::Snd_fpage	??
L4::lpc::In_out< T >	??
L4::lpc::Istream	??
L4::lpc::Iostream	??
L4::lpc::Msg::Cls_buffer	??
L4::lpc::Msg::Do_rcv_buffers	??
L4::lpc::Msg::Cls_data	??
L4::lpc::Msg::Do_in_data	??
L4::lpc::Msg::Do_out_data	??
L4::lpc::Msg::Cls_item	??
L4::lpc::Msg::Do_in_items	??
L4::lpc::Msg::Do_out_items	??
L4::lpc::Msg::Dir_in	??
L4::lpc::Msg::Do_in_data	??
L4::lpc::Msg::Do_in_items	??
L4::lpc::Msg::Do_rcv_buffers	??
L4::lpc::Msg::Dir_out	??
L4::lpc::Msg::Do_out_data	??
L4::lpc::Msg::Do_out_items	??
L4::lpc::Msg::Elem< Array< A, LEN > & >	??
L4::lpc::Msg::Elem< Array< A, LEN > >	??
L4::lpc::Msg::Elem< Array_ref< A, LEN > & >	??
L4::lpc::Msg::Svr_arg_pack< IPC_TYPE >	??
L4::lpc::Msg::Svr_val_ops< MTYPE, DIR, CLASS >	??
L4::lpc::Msg::Svr_val_ops< L4::lpc::Snd_fpage, Dir_in, CLASS >	??
L4::lpc::Msg_ptr< T >	??
L4::lpc::Opt< T >	??
L4::lpc::Ostream	??
L4::lpc::Iostream	??
L4::lpc::Out< T >	??
L4::lpc::Ret_array< T >	??
L4::lpc::Send_only	??
L4::lpc::Small_buf	??
L4::lpc::Str_cp_in< T >	??

L4::lpc::Varg	??
L4::lpc::Varg_list_ref	??
L4::lpc::Varg_list< MAX >	??
L4::lpc::Varg_list_ref::iterator	??
L4::lpc_svr::Compound_reply	??
L4::lpc_svr::Default_loop_hooks	??
L4::Server< L4::lpc_svr::Default_loop_hooks >	??
L4Re::Util::Registry_server< LOOP_HOOKS >	??
L4::Server< LOOP_HOOKS >	??
L4Re::Util::Registry_server< Loop_hooks >	??
L4Re::Util::Br_manager_hooks	??
L4::lpc_svr::Default_setup_wait	??
L4::lpc_svr::Default_timeout	??
L4::lpc_svr::Default_loop_hooks	??
L4Re::Util::Br_manager_hooks	??
L4::lpc_svr::Direct_dispatch< R >	??
L4::lpc_svr::Dbg_dispatch< R, Exc, Printer >	??
L4::lpc_svr::Exc_dispatch< R, Exc >	??
L4::lpc_svr::Direct_dispatch< R * >	??
L4::lpc_svr::Ignore_errors	??
L4::lpc_svr::Default_loop_hooks	??
L4Re::Util::Br_manager_hooks	??
L4Re::Util::Br_manager_timeout_hooks	??
L4::lpc_svr::Server_iface	??
L4::lpc_svr::Timeout_queue_hooks< Loop_hooks, L4Re::Util::Br_manager >	??
L4::lpc_svr::Timeout_queue_hooks< Br_manager_timeout_hooks, Br_manager >	??
L4Re::Util::Br_manager_timeout_hooks	??
L4::lpc_svr::Br_manager_no_buffers	??
L4::lpc_svr::Default_loop_hooks	??
L4::lpc_svr::Timeout_queue_hooks< HOOKS, BR_MAN >	??
L4::lpc_svr::Timeout_queue_hooks< HOOKS, BR_MAN >	??
L4Re::Util::Br_manager	??
L4::lpc_svr::Timeout_queue_hooks< Loop_hooks, L4Re::Util::Br_manager >	??
L4Re::Util::Br_manager_hooks	??
L4::lpc_svr::Timeout_queue	??
L4::Kip::Mem_desc	??
L4::Kobject	??
L4::Kobject_t< Arm_smccc, L4::Kobject, PROTO, Type_info::Demand_t<> >	??
L4::Kobject_t< Debugger, Kobject, L4_PROTO_DEBUGGER >	??
L4::Debugger	??
L4::Kobject_t< Exception, L4::Kobject, PROTO, Type_info::Demand_t<> >	??
L4::Kobject_t< Factory, Kobject, L4_PROTO_FACTORY >	??
L4::Factory	??
L4::Kobject_t< Mem_alloc, L4::Factory, L4RE_PROTO_MEM_ALLOC >	??
L4Re::Mem_alloc	??
L4::Kobject_t< Io_pager, L4::Kobject, PROTO, Type_info::Demand_t<> >	??
L4::Kobject_t< Irq_eoi, L4::Kobject, PROTO, Type_info::Demand_t<> >	??
L4::Kobject_t< Derived, L4::Kobject, L4::PROTO_ANY, Type_info::Demand_t<> >	??
L4::Kobject_t< Meta, Kobject, L4_PROTO_META >	??
L4::Meta	??
L4::Kobject_t< Platform_control, Kobject, L4_PROTO_PLATFORM_CTL >	??
L4::Platform_control	??
L4::Kobject_t< Rcv_endpoint, Kobject, L4_PROTO_KOBJECT, Type_info::Demand_t< 1 > >	??
L4::Rcv_endpoint	??
L4::Kobject_2t< Irq, Triggerable, Rcv_endpoint, L4_PROTO_IRQ_SENDER >	??

L4::Irq	??
L4::Kobject_t< lpc_gate, Rcv_endpoint, L4_PROTO_KOBJECT, Type_info::Demand_t< 1 > >	??
L4::lpc_gate	??
L4::Kobject_t< Snd_destination, Kobject, L4_PROTO_KOBJECT >	??
L4::Kobject_t< Task, Kobject, L4_PROTO_TASK, Type_info::Demand_t< 2 > >	??
L4::Task	??
L4::Kobject_t< Vm, Task, L4_PROTO_VM >	??
L4::Vm	??
L4::Vm	??
L4::Kobject_t< Vcpu_context, Kobject, L4_PROTO_VCPU_CONTEXT >	??
L4::Kobject_t< Dataspace, L4::Kobject, L4RE_PROTO_DATASPACE, L4::Type_info::Demand_t< 1 > >	??
L4Re::Dataspace	??
L4::Kobject_3t< Vbus, L4Re::Dataspace, L4Re::Inhibitor, L4Re::Event >	??
L4vbus::Vbus	??
L4::Kobject_t< Dbg_events, L4::Kobject, 0, L4::Type_info::Demand_t< 2 > >	??
L4::Kobject_t< Debug_obj, L4::Kobject, L4RE_PROTO_DEBUG >	??
L4Re::Debug_obj	??
L4::Kobject_t< Dma_space, L4::Kobject, PROTO, L4::Type_info::Demand_t< 1 > >	??
L4::Kobject_t< Inhibitor, L4::Kobject, L4RE_PROTO_INHIBITOR >	??
L4Re::Inhibitor	??
L4::Kobject_3t< Vbus, L4Re::Dataspace, L4Re::Inhibitor, L4Re::Event >	??
L4::Kobject_t< Itas, L4::Kobject, L4RE_PROTO_ITAS, L4::Type_info::Demand_t< 2 > >	??
L4Re::Itas	??
L4::Kobject_t< Mmio_space, L4::Kobject, L4RE_PROTO_MMIO_SPACE >	??
L4Re::Mmio_space	??
L4::Kobject_t< Namespace, L4::Kobject, L4RE_PROTO_NAMESPACE, L4::Type_info::Demand_t< 1 > >	??
L4Re::Namespace	??
L4::Kobject_t< Cmd_control, L4::Kobject, L4::PROTO_ANY, Type_info::Demand_t<> >	??
L4::Kobject_t< Parent, L4::Kobject, L4RE_PROTO_PARENT >	??
L4Re::Parent	??
L4::Kobject_t< Goos, L4::Kobject, L4RE_PROTO_GOOS >	??
L4Re::Video::Goos	??
L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND >	??
L4::Kobject_2t< Console, Video::Goos, Event, L4::PROTO_EMPTY >	??
L4Re::Console	??
L4::Kobject_2t< Debug_obj_t< BASE >, BASE, Debug_obj, L4::PROTO_EMPTY >	??
L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND >	??
L4::Kobject_demand< T >	??
L4::Kobject_t< Derived, Base, PROTO, S_DEMAND >	??
L4::Kobject_t< Remote_access, Dataspace, L4RE_PROTO_REMOTE_ACCESS >	??
L4::Kobject_typeid< T >	??
L4::Kobject_typeid< void >	??
L4::Kobject_x< Derived, ARGS >	??
L4::Kobject_x< Iommu, Proto_t< L4_PROTO_IOMMU >, Type_info::Demand_t< 1 > >	??
L4::Iommu	??
L4::Lock_guard	??
L4::Poll_timeout_counter	??
L4::Poll_timeout_kipclock	??
L4::Proto_t< P >	??
L4::Registry_iface	??
L4Re::Util::Object_registry	??
L4::Server< LOOP_HOOKS >	??
L4::String	??
L4::Thread::Attr	??

L4::Thread::Modify_senders	??
L4::Type_info	??
L4::Type_info::Demand	??
L4::Type_info::Demand_t< __l::Max< Kobject_typeid< T1 >::Demand::Caps, Kobject_demand< T2... >::Caps >::Res, Kobject_typeid< T1 >::Demand::Flags Kobject_demand< T2... >::Flags, __l::Max< Kobject_typeid< T1 >::Demand::Mem, Kobject_demand< T2... >::Mem >::Res, __l::Max< Kobject_typeid< T1 >::Demand::Ports, Kobject_demand< T2... >::Ports >::Res >	??
L4::Type_info::Demand_union_t< Kobject_typeid< T1 >::Demand, Kobject_demand< T2... > >	??
L4::Type_info::Demand_t< __l::Max< D1::Caps, D2::Caps >::Res, D1::Flags D2::Flags, __l::Max< D1::Mem, D2::Mem >::Res, __l::Max< D1::Ports, D2::Ports >::Res >	??
L4::Type_info::Demand_union_t< D1, D2 >	??
L4::Type_info::Demand_t< CAPS, FLAGS, MEM, PORTS >	??
L4::Typeid::Detail::_Rpc< OPCODE, O, Default_op< R > >::Rpc< Y >	??
L4::Typeid::Detail::_Rpc< OPCODE, O, R, X... >::Rpc< Y >	??
L4::Typeid::Detail::Rpc_end	??
L4::Typeid::Detail::type< OPCODE, 1,... >	??
L4::Typeid::Detail::_Rpc< void, 0, OPERATION >	??
L4::Typeid::Rpc_nocode< OPERATION >	??
L4::Typeid::Detail::_Rpc< L4::Opcode, 0, RPCS... >	??
L4::Typeid::Rpc< set_status_t, config_queue_t, register_ds_t, device_config_t, device_notification_irq_t >	??
L4::Typeid::Rpc< execute_t >	??
L4::Typeid::Rpc< num_interfaces_t, interface_t, supports_t >	??
L4::Typeid::Rpc< map_t, clear_t, info_t, copy_in_t, allocate_t, map_info_t >	??
L4::Typeid::Rpc< request_backtrace_t >	??
L4::Typeid::Rpc< map_t, unmap_t, associate_t, disassociate_t >	??
L4::Typeid::Rpc< get_buffer_t, get_num_streams_t, get_stream_info_t, get_stream_info_for_id_t, get_axis_info_t, get_stream_state_for_id_t >	??
L4::Typeid::Rpc< acquire_t, release_t, next_lock_info_t >	??
L4::Typeid::Rpc< register_thread_t, unregister_thread_t, sigaction_t, sigaltstack_t, sigprocmask_t, sigpending_t, setitimer_t, getitimer_t, raise_t >	??
L4::Typeid::Rpc< info_t >	??
L4::Typeid::Rpc< mmio_read_t, mmio_write_t >	??
L4::Typeid::Rpc< query_t, register_obj_t, unlink_t >	??
L4::Typeid::Rpc< signal_t >	??
L4::Typeid::Rpc< get_random_t >	??
L4::Typeid::Rpc< read_mem_t, write_mem_t, terminate_t >	??
L4::Typeid::Rpc< attach_t, detach_t, find_t, reserve_area_t, free_area_t, get_regions_t, get_areas_t, get_info_t >	??
L4::Typeid::Rpc< info_t, get_static_buffer_t, create_buffer_t, create_view_t, delete_buffer_t, delete_view_t, view_info_t, set_view_info_t, view_stack_t, view_refresh_t, refresh_t >	??
L4::Typeid::Rpc< RPCS >	??
L4::Typeid::Detail::_Rpc< OPCODE_TYPE, 0, RPCS... >	??
L4::Typeid::Rpc_code< OPCODE_TYPE >::F< RPCS >	??
L4::Typeid::Detail::_Rpc< l4_umword_t, 0, ARG... >	??
L4::Typeid::Rpc_sys< bind_thread_t, get_infos_t >	??
L4::Typeid::Rpc_sys< bind_t, unbind_t, info_t, msi_info_t, unmask_t, mask_t, set_mode_t >	??
L4::Typeid::Rpc_sys< system_suspend_t, system_shutdown_t, cpu_allow_shutdown_t, cpu_enable_t, cpu_disable_t >	??
L4::Typeid::Rpc_sys< bind_thread_t >	??
L4::Typeid::Rpc_sys< info_t, run_thread_t, idle_time_t >	??
L4::Typeid::Rpc_sys< ARG >	??
L4::Typeid::Detail::_Rpc< OPCODE, O, R, X... >	??
L4::Typeid::Detail::_Rpc< OPCODE, O, X >	??
L4::Typeid::Rpc_nocode< call_t >	??
L4::Typeid::Rpc_nocode< exception_t >	??

L4::Typeid::Rpc_nocode< create_t >	??
L4::Typeid::Rpc_nocode< io_page_fault_t >	??
L4::Typeid::Rpc_nocode< page_fault_t >	??
L4::Typeid::Rpc_nocode< debug_t >	??
L4::Typeid::P_dispatch< LIST >	??
L4::Typeid::Raw_ipc< CLASS >	??
L4::Typeid::Rpcs_code< OPCODE_TYPE >	??
L4::Types::Bool< V >	??
L4::Types::Bool< false >	??
L4::Types::False	??
L4::Types::Same< A, B >	??
L4::Types::Bool< I1::Proto !=PROTO_EMPTY &&I1::Proto==I2::Proto >	??
L4::Types::Bool< true >	??
L4::Types::True	??
L4::Ipc::Msg::ls_valid_rpc_type< A * >	??
L4::Ipc::Msg::ls_valid_rpc_type< T >	??
L4::Types::Flags< BITS_ENUM, UNDERLYING >	??
L4::Types::Flags_ops_t< DT >	??
L4::Types::Flags_ops_t< Flags >	??
L4::Types::Flags_ops_t< Flags_t< DT, T > >	??
L4::Types::Flags_t< DT, T >	??
L4::Types::Int_for_size< SIZE, bool >	??
L4::Types::Int_for_type< T >	??
L4::Uart	??
L4::Uart_apb	??
l4_buf_regs_t	??
l4_exc_regs_t	??
l4_fpage_t	??
l4_icu_info_t	??
L4::Icu::Info	??
l4_icu_msi_info_t	??
l4_kernel_info_mem_desc_t	??
l4_kernel_info_t	??
l4_msg_regs_t	??
l4_msgtag_t	??
l4_sched_cpu_set_t	??
l4_sched_param_t	??
l4_snd_fpage_t	??
l4_thread_regs_t	??
l4_timeout_s	??
l4_timeout_t	??
l4_vcon_attr_t	??
l4_vcpu_arch_state_t	??
l4_vcpu_ipc_regs_t	??
l4_vcpu_regs_t	??
l4_vcpu_state_t	??
L4vcpu::Vcpu	??
l4_vm_svm_vmcb_control_area	??
l4_vm_svm_vmcb_state_save_area	??
l4_vm_svm_vmcb_state_save_area_seg	??
l4_vm_svm_vmcb_t	??
l4_vm_tz_state	??
l4_vm_vmx_vcpu_infos_t	??
l4_vm_vmx_vcpu_state_t	??
l4_vm_vmx_vcpu_vmcs_t	??
l4_vmx_offset_table_t	??

L4drivers::Register_block< MAX_BITS, BLOCK >	??
L4drivers::Register_block_base< MAX_BITS >	??
L4drivers::Register_block_impl< BASE, MAX_BITS >	??
L4drivers::Register_block_impl< Mmio_register_block< 32 >, 32 >	??
L4drivers::Mmio_register_block< MAX_BITS >	??
L4drivers::Register_block_tmpl< BLOCK >	??
L4drivers::Register_block_tmpl< Register_block_base< MAX_BITS > >	??
L4drivers::Register_block_tmpl< Register_block_base< MAX_BITS > const >	??
L4drivers::Ro_register_block< MAX_BITS, BLOCK >	??
L4drivers::Ro_register_tmpl< BITS, BLOCK >	??
L4drivers::Register_tmpl< MAX_BITS, Block >	??
L4drivers::Register_tmpl< BITS, BLOCK >	??
L4Re::Cap_alloc	??
L4Re::Util::Cap_alloc	??
L4Re::Core::Ref_ptr< T, CNT >	??
L4Re::Dataspace::F	??
L4Re::Dataspace::Stats	??
L4Re::Default_event_payload	??
L4Re::Env	??
L4Re::Event_buffer_t< PAYLOAD >	??
L4Re::Event_buffer_t< PAYLOAD >::Event	??
L4Re::Event_buffer_t< Default_event_payload >	??
L4Re::Util::Event_buffer_t< Default_event_payload >	??
L4Re::Util::Event_buffer_consumer_t< Default_event_payload >	??
L4Re::Event_buffer_t< PAYLOAD >	??
L4Re::Util::Event_buffer_t< PAYLOAD >	??
L4Re::Util::Event_buffer_consumer_t< PAYLOAD >	??
L4Re::Mem_alloc::Stats	??
L4Re::Rm::Area	??
L4Re::Rm::F	??
L4Re::Rm::Region	??
L4Re::Rm::Unique_region< T >	??
L4Re::Smart_cap_auto< Unmap_flags >	??
L4Re::Smart_count_cap< Unmap_flags >	??
L4Re::Util::Bitmap_base	??
cxx::Bitmap< BITS >	??
L4Re::Util::Bitmap_base::Bit	??
L4Re::Util::Bitmap_base::Char< BITS >	??
L4Re::Util::Bitmap_base::Word< BITS >	??
L4Re::Util::Cap_alloc_base	??
L4Re::Util::Counter< COUNTER >	??
L4Re::Util::Counter_atomic< COUNTER >	??
L4Re::Util::Counting_cap_alloc< COUNTERTYPE, Dbg >	??
L4Re::Util::Dataspace_svr	??
L4Re::Util::Event_svr< SVR >	??
L4Re::Util::Event_t< PAYLOAD >	??
L4Re::Util::Item_alloc_base	??
L4Re::Util::Names::Name_space	??
L4Re::Util::Ref_cap< T >	??
L4Re::Util::Ref_del_cap< T >	??
L4Re::Util::Smart_cap_auto< Unmap_flags >	??
L4Re::Util::Smart_count_cap< Unmap_flags >	??
L4Re::Util::Vcon_svr< SVR >	??
L4Re::Util::Video::Goos_svr	??
L4Re::Vfs::Directory	??
L4Re::Vfs::File	??

L4Re::Vfs::Be_file	??
L4Re::Vfs::File_system	??
L4Re::Vfs::Be_file_system	??
L4Re::Vfs::Fs	??
L4Re::Vfs::Ops	??
L4Re::Vfs::Generic_file	??
L4Re::Vfs::File	??
L4Re::Vfs::Mman	??
L4Re::Vfs::Ops	??
L4Re::Vfs::Regular_file	??
L4Re::Vfs::File	??
L4Re::Vfs::Special_file	??
L4Re::Vfs::File	??
L4Re::Video::Color_component	??
L4Re::Video::Goos::Info	??
L4Re::Video::Pixel_info	??
L4Re::Video::View	??
L4Re::Video::View::Info	??
l4re_aux_t	??
l4re_ds_stats_t	??
l4re_elf_aux_mword_t	??
l4re_elf_aux_t	??
l4re_elf_aux_vma_t	??
l4re_env_cap_entry_t	??
l4re_env_t	??
l4re_event_t	??
l4re_video_color_component_t	??
l4re_video_goos_info_t	??
l4re_video_pixel_info_t	??
l4re_video_view_info_t	??
l4re_video_view_t	??
l4shmc_ringbuf_head_t	??
l4shmc_ringbuf_t	??
l4util_l4mod_info	??
l4util_l4mod_mod	??
l4util_mb_addr_range_t	??
l4util_mb_apm_t	??
l4util_mb_drive_t	??
l4util_mb_info_t	??
l4util_mb_mod_t	??
l4util_mb_vbe_ctrl_t	??
l4util_mb_vbe_mode_t	??
L4vbus::Gpio_module::Pin_slice	??
L4vbus::Pm< DEC >	??
L4vbus::Pm< Device >	??
L4vbus::Device	??
L4vbus::Gpio_module	??
L4vbus::Gpio_pin	??
L4vbus::Icu	??
L4vbus::Pci_dev	??
L4vbus::Pci_host_bridge	??
l4vbus_device_t	??
l4vbus_resource_t	??
L4vcpu::State	??
L4virtio::Driver::Block_device::Handle	??
L4virtio::Driver::Device	??

L4virtio::Driver::Block_device	??
L4virtio::Driver::Virtio_net_device	??
L4virtio::Driver::Virtio_net_device::Packet	??
L4virtio::Ptr< T >	??
L4virtio::Svr::Bad_descriptor	??
L4virtio::Svr::Block_request< Ds_data >	??
L4virtio::Svr::Console::Control_message	??
L4virtio::Svr::Console::Control_request	??
L4virtio::Svr::Console::Port	??
L4virtio::Svr::Console::Device_port	??
L4virtio::Svr::Console::Port::Transition	??
L4virtio::Svr::Data_buffer	??
Buffer	??
L4virtio::Svr::Dev_config	??
L4virtio::Svr::Dev_features	??
L4virtio::Svr::Console::Features	??
L4virtio::Svr::Dev_status	??
L4virtio::Svr::Device_t< DATA >	??
L4virtio::Svr::Device_t< Ds_data >	??
L4virtio::Svr::Block_dev_base< Ds_data >	??
L4virtio::Svr::Device_t< Mem_region_info >	??
L4virtio::Svr::Block_dev_base< Mem_region_info >	??
L4virtio::Svr::Device_t< No_custom_data >	??
L4virtio::Svr::Console::Virtio_con	??
L4virtio::Svr::Console::Device	??
L4virtio::Svr::Scmi::Scmi_dev	??
L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >	??
L4virtio::Svr::Virtio_i2c< Request_handler, Epiface >	??
L4virtio::Svr::Virtio_rng< Rnd_state, Epiface >	??
Virtio_net	??
L4virtio::Svr::Driver_mem_list_t< DATA >	??
L4virtio::Svr::Driver_mem_region_t< DATA >	??
L4virtio::Svr::Driver_mem_region_t< Ds_data >	??
L4virtio::Svr::Driver_mem_region_t< Mem_region_info >	??
L4virtio::Svr::Driver_mem_region_t< No_custom_data >	??
L4virtio::Svr::Request_processor	??
L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::Request_processor	??
L4virtio::Svr::Virtio_i2c< Request_handler, Epiface >::Request_processor	??
L4virtio::Svr::Virtio_rng< Rnd_state, Epiface >::Request_processor	??
L4virtio::Svr::Scmi::Base_attr_t	??
L4virtio::Svr::Scmi::Performance_attr_t	??
L4virtio::Svr::Scmi::Performance_describe_level_t	??
L4virtio::Svr::Scmi::Performance_describe_levels_n_t	??
L4virtio::Svr::Scmi::Performance_domain_attr_t	??
L4virtio::Svr::Scmi::Proto< OBSERV >	??
L4virtio::Svr::Scmi::Proto< Scmi_dev >	??
L4virtio::Svr::Scmi::Base_proto	??
L4virtio::Svr::Scmi::Perf_proto	??
L4virtio::Svr::Scmi::Scmi_hdr_t	??
L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::Irq_handler	??
L4virtio::Svr::Virtqueue::Head_desc	??
L4virtio::Virtqueue	??
L4virtio::Driver::Virtqueue	??
L4virtio::Svr::Virtqueue	??
L4virtio::Virtqueue::Avail	??

L4virtio::Virtqueue::Avail::Flags	??
L4virtio::Virtqueue::Desc	??
L4virtio::Virtqueue::Desc::Flags	??
L4virtio::Virtqueue::Used	??
L4virtio::Virtqueue::Used::Flags	??
L4virtio::Virtqueue::Used_elem	??
l4virtio_block_config_t	??
l4virtio_block_discard_t	??
l4virtio_block_header_t	??
l4virtio_config_hdr_t	??
l4virtio_config_queue_t	??
l4virtio_input_absinfo_t	??
l4virtio_input_config_t	??
l4virtio_input_devids_t	??
l4virtio_input_event_t	??
l4virtio_net_config_t	??
l4virtio_net_header_t	??
Mac_addr	??
Mac_table< Size >	??
Net_transfer	??
Rm::Area	??
Rm::F	??
Rm::Region	??
Rm::Unique_region< T >	??
Virtio_net_request	??
Virtio_switch	??
Virtio_vlan_mangle	??

Chapter 12

Data Structure Index

12.1 Data Structures

Here are the data structures with brief descriptions:

Block_device::Device_discard_feature	Partial interface for devices that offer discard functionality	??
Block_device::Device_mgr< DEV, FACTORY, SCHEDULER >	Basic class that scans devices and handles client connections	??
Block_device::Device_with_notification_domain< DEV >	Device with a per-device notification domain	??
Block_device::Dma_region_info	Base class used by the driver implementation to derive its own DMA mapping tracking structure	??
Block_device::Errand::Errand	Wrapper for a small task executed asynchronously in the server loop	??
Block_device::Errand::Poll_errand	Wrapper for a regularly repeated task	??
Block_device::Impl::Partitioned_device_discard_mixin< PART_DEV, BASE_DEV, bool >	Dummy class used when the device class is not derived from Device_discard_feature	??
Block_device::Impl::Partitioned_device_discard_mixin< PART_DEV, BASE_DEV, true >	Mixin implementing discard for partition devices	??
Block_device::Inout_block	Description of an inout block to be sent to the device	??
Block_device::Inout_memory< DEV >	Helper class that temporarily allocates memory that can be used for in/out operations with the device	??
Block_device::Mem_region_info	Additional info stored in each L4virtio::Svr::Driver_mem_region_t used for tracking dataspace-wide DMA mappings	??
Block_device::Notification_domain	Opaque type for representing a notification domain	??
Block_device::Partition_info	Information about a single partition	??
Block_device::Partition_reader< DEV >	Partition table reader for block devices	??
Block_device::Partitioned_device< BASE_DEV >	A partition device for the given device interface	??
Block_device::Pending_request	Interface for pending requests	??
Block_device::Rr_scheduler< DEV >	Round Robin scheduler class	??

Block_device::Scheduler_base< DEV >	
Scheduler base class	??
Buffer	
Data buffer used to transfer packets	??
cxx::arith::Ld< V >	
Computes the binary logarithm of the given number at compile time	??
cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC >	
AVL tree based associative container	??
cxx::Avl_set< ITEM_TYPE, COMPARE, ALLOC >	
AVL set for simple comparable items	??
cxx::Avl_tree< Node, Get_key, Compare >	
A generic AVL tree	??
cxx::Avl_tree_node	
Node of an AVL tree	??
cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >	
Basic slab allocator	??
cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >::Slab_i	
Type of a slab	??
cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >	
Merged slab allocator (allocators for objects of the same size are merged together)	??
cxx::Bitfield< T, LSB, MSB >	
Definition for a member (part) of a bit field	??
cxx::Bitfield< T, LSB, MSB >::Value< TT >	
Internal helper type	??
cxx::Bitfield< T, LSB, MSB >::Value_base< TT >	
Internal helper type	??
cxx::Bitfield< T, LSB, MSB >::Value_unshifted< TT >	
Internal helper type	??
cxx::Bitmap< BITS >	
A static bitmap	??
cxx::Bitmap_base	
Basic bitmap abstraction	??
cxx::Bitmap_base::Bit	
A writable bit in a bitmap	??
cxx::Bitmap_base::Char< BITS >	
Helper abstraction for a byte contained in the bitmap	??
cxx::Bitmap_base::Word< BITS >	
Helper abstraction for a word contained in the bitmap	??
cxx::Bits::Avl_map_get_key< KEY_TYPE >	
Key-getter for Avl_map	??
cxx::Bits::Avl_set_get_key< KEY_TYPE >	
Internal, key-getter for Avl_set nodes	??
cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >	
Internal: AVL set with internally managed nodes	??
cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Node	
A smart pointer to a tree item	??
cxx::Bits::Basic_list< POLICY >	
Internal: Common functions for all head-based list implementations	??
cxx::Bits::Bst< Node, Get_key, Compare >	
Basic binary search tree (BST)	??
cxx::Bits::Bst_node	
Basic type of a node in a binary search tree (BST)	??
cxx::Bits::Direction	
The direction to go in a binary search tree	??
cxx::Bits::Smart_ptr_list< ITEM >	
List of smart-pointer-managed objects	??
cxx::Bits::Smart_ptr_list_item< T, STORE_T >	
List item for an arbitrary item in a Smart_ptr_list	??

cxx::H_list< T, POLICY >	General double-linked list of unspecified cxx::H_list_item elements	??
cxx::H_list_item_t< ELEM_TYPE >	Basic element type for a double-linked H_list	??
cxx::H_list_t< T >	Double-linked list of typed H_list_item_t elements	??
cxx::List< D, Alloc >	Doubly linked list, with internal allocation	??
cxx::List< D, Alloc >::Iter	Iterator	??
cxx::List_alloc	Standard list-based allocator	??
cxx::List_item	Basic list item	??
cxx::List_item::Iter	Iterator for a list of ListItem -s	??
cxx::List_item::T_iter< T, Poly >	Iterator for derived classes from ListItem	??
cxx::Lt_functor< Obj >	Generic comparator class that defaults to the less-than operator	??
cxx::New_allocator< _Type >	Standard allocator based on <code>operator new ()</code>	??
cxx::Nothrow	Helper type to distinguish the <code>operator new</code> version that does not throw exceptions . . .	??
cxx::Pair< First, Second >	Pair of two values	??
cxx::Pair_first_compare< Cmp, Typ >	Comparison functor for Pair	??
cxx::Ref_obj_list_item< T >	Item for list linked via cxx::Ref_ptr with default reference counting	??
cxx::Ref_ptr< T, CNT >	A reference-counting pointer with automatic cleanup	??
cxx::S_list< T, POLICY >	Simple single-linked list	??
cxx::Slab< Type, Slab_size, Max_free, Alloc >	Slab allocator for object of type <code>Type</code>	??
cxx::Slab_static< Type, Slab_size, Max_free, Alloc >	Merged slab allocator (allocators for objects of the same size are merged together)	??
cxx::static_vector< T, IDX >	Simple encapsulation for a dynamically allocated array	??
cxx::String	Allocation free string class with explicit length field	??
cxx::Weak_ref< T >	Typed weak reference to an object of type <code>T</code>	??
cxx::Weak_ref_base	Generic (base) weak reference to some object	??
cxx::Weak_ref_base::List	The list type for keeping all weak references to an object	??
Elf32_Auxv	Auxiliary vector (32-bit)	??
Elf32_Dyn	ELF32 dynamic entry	??
Elf32_Ehdr	ELF32 header	??
Elf32_Phdr	ELF32 program header	??
Elf32_Rel	ELF32 relocation entry w/o addend	??

Elf32_Rela	ELF32 relocation entry w/ addend	??
Elf32_Shdr	ELF32 section header	??
Elf32_Sym	ELF32 symbol table entry	??
Elf64_Auxv	Auxiliary vector (64-bit)	??
Elf64_Dyn	ELF64 dynamic entry	??
Elf64_Ehdr	ELF64 header	??
Elf64_Phdr	ELF64 program header	??
Elf64_Rel	ELF64 relocation entry w/o addend	??
Elf64_Rela	ELF64 relocation entry w/ addend	??
Elf64_Shdr	ELF64 section header	??
Elf64_Sym	ELF64 symbol table entry	??
gfxbitmap_offset	Offsets in pmap[] and bmap[]	??
L4::Alloc_list	A simple list-based allocator	??
L4::Arm_smccc	Wrapper for function calls that follow the ARM SMC/HVC calling convention	??
L4::Base_exception	Base class for all exceptions, thrown by the L4Re framework	??
L4::Basic_registry	This registry returns the corresponding server object based on the label of an lpc_gate	??
L4::Bounds_error	Access out of bounds	??
L4::Cap< T >	C++ interface for capabilities	??
L4::Cap_base	Base class for all kinds of capabilities	??
L4::Com_error	Error conditions during IPC	??
L4::Debugger	C++ kernel debugger API	??
L4::Element_already_exists	Exception for duplicate element insertions	??
L4::Element_not_found	Exception for a failed lookup (element not found)	??
L4::Epiface	Base class for interface implementations	??
L4::Epiface_t< Derived, IFACE, BASE, bool >	Epiface implementation for Kobject-based interface implementations	??
L4::Epiface_t0< RPC_IFACE, BASE >	Epiface mixin for generic Kobject-based interfaces	??
L4::Exception	Exception interface	??
L4::Exception_tracer	Back-trace support for exceptions	??
L4::Factory	C++ Factory interface, see Factory for the C interface	??

L4::Factory::Lstr	Special type to add a pascal string into the factory create stream	??
L4::Factory::Nil	Special type to add a void argument into the factory create stream	??
L4::Factory::S	Stream class for the create() argument stream	??
L4::lcu	C++ lcu interface, see Interrupt controller for the C interface	??
L4::lcu::Info	This class encapsulates information about an ICU	??
L4::Invalid_capability	Indicates that an invalid object was invoked	??
L4::lo_pager	lo_pager interface	??
L4::lommu	Interface for IO-MMUs used for DMA remapping	??
L4::IOModifier	Modifier class for the IO stream	??
L4::lpc::Array< ELEM_TYPE, LEN_TYPE >	Array data type for dynamically sized arrays in RPCs	??
L4::lpc::Array_in_buf< ELEM_TYPE, LEN_TYPE, MAX >	Server-side copy in buffer for Array	??
L4::lpc::Array_ref< ELEM_TYPE, LEN_TYPE >	Array reference data type for arrays located in the message	??
L4::lpc::As_value< T >	Pass the argument as plain data value	??
L4::lpc::Call	RPC attribute for a standard RPC call	??
L4::lpc::Call_t< RIGHTS >	RPC attribute for an RPC call with required rights	??
L4::lpc::Call_zero_send_timeout	RPC attribute for an RPC call, with zero send timeout	??
L4::lpc::Cap< T >	Capability type for RPC interfaces (see L4::Cap<T>)	??
L4::lpc::Gen_fpage	Generic RPC base for typed message items	??
L4::lpc::In_out< T >	Mark an argument as in-out argument	??
L4::lpc::lostream	Input/Output stream for IPC [un]marshalling	??
L4::lpc::lstream	Input stream for IPC unmarshalling	??
L4::lpc::Msg::Cls_buffer	Marker type for receive buffer values	??
L4::lpc::Msg::Cls_data	Marker type for data values	??
L4::lpc::Msg::Cls_item	Marker type for item values	??
L4::lpc::Msg::Dir_in	Marker type for input values	??
L4::lpc::Msg::Dir_out	Marker type for output values	??
L4::lpc::Msg::Do_in_data	Marker for Input data	??
L4::lpc::Msg::Do_in_items	Marker for Input items	??
L4::lpc::Msg::Do_out_data	Marker for Output data	??

L4::lpc::Msg::Do_out_items	
Marker for Output items	??
L4::lpc::Msg::Do_rcv_buffers	
Marker for receive buffers	??
L4::lpc::Msg::Elem< Array< A, LEN > & >	
Array as output argument	??
L4::lpc::Msg::Elem< Array< A, LEN > >	
Array as input arguments	??
L4::lpc::Msg::Elem< Array_ref< A, LEN > & >	
Array_ref as output argument	??
L4::lpc::Msg::False	
False meta value	??
L4::lpc::Msg::Is_valid_rpc_type< T >	
Type trait defining a valid RPC parameter type	??
L4::lpc::Msg::Svr_arg_pack< IPC_TYPE >	
Server-side RPC arguments data structure used to provide arguments to the server-side implementation of an RPC function	??
L4::lpc::Msg::Svr_val_ops< MTYPE, DIR, CLASS >	
Defines client-side handling of 'MTYPE' as RPC argument	??
L4::lpc::Msg::True	
True meta value	??
L4::lpc::Msg_ptr< T >	
Pointer to an element of type T in an lpc::lstream	??
L4::lpc::Opt< T >	
Attribute for defining an optional RPC argument	??
L4::lpc::Ostream	
Output stream for IPC marshalling	??
L4::lpc::Out< T >	
Mark an argument as a output value in an RPC signature	??
L4::lpc::Rcv_fpage	
Non-small receive item	??
L4::lpc::Ret_array< T >	
Dynamically sized output array of type T	??
L4::lpc::Send_only	
RPC attribute for a send-only RPC	??
L4::lpc::Small_buf	
A receive item for receiving a single object capability	??
L4::lpc::Snd_fpage	
Send item or return item	??
L4::lpc::Str_cp_in< T >	
Abstraction for extracting a zero-terminated string from an lpc::lstream	??
L4::lpc::Varg	
Variably sized RPC argument	??
L4::lpc::Varg_list< MAX >	
Self-contained list of variable-sized RPC parameters	??
L4::lpc::Varg_list_ref	
List of variable-sized RPC parameters as received by the server	??
L4::lpc::Varg_list_ref::Iterator	
Iterator for Valists	??
L4::lpc_gate	
The C++ IPC gate interface, see IPC-Gate API for the C interface	??
L4::lpc_svr::Br_manager_no_buffers	
Empty implementation of Server_iface	??
L4::lpc_svr::Compound_reply	
Mix in for LOOP_HOOKS to always use compound reply and wait	??
L4::lpc_svr::Dbg_dispatch< R, Exc, Printer >	
Dispatch helper that, in addition to what Exc_dispatch does, prints exception messages	??

L4::lpc_svr::Default_loop_hooks	
Default LOOP_HOOKS	??
L4::lpc_svr::Default_setup_wait	
Mix in for LOOP_HOOKS for setup_wait no op	??
L4::lpc_svr::Default_timeout	
Mix in for LOOP_HOOKS to use a 0 send and a infinite receive timeout	??
L4::lpc_svr::Direct_dispatch< R >	
Direct dispatch helper, for forwarding dispatch calls to a registry <i>R</i>	??
L4::lpc_svr::Direct_dispatch< R * >	
Direct dispatch helper, for forwarding dispatch calls via a pointer to a registry <i>R</i>	??
L4::lpc_svr::Exc_dispatch< R, Exc >	
Dispatch helper wrapping try {} catch {} around the dispatch call	??
L4::lpc_svr::Ignore_errors	
Mix in for LOOP_HOOKS to ignore IPC errors	??
L4::lpc_svr::Server_iface	
Interface for server-loop related functions	??
L4::lpc_svr::Timeout	
Callback interface for Timeout_queue	??
L4::lpc_svr::Timeout_queue	
Timeout queue to be used in L4re server loop	??
L4::lpc_svr::Timeout_queue_hooks< HOOKS, BR_MAN >	
Loop hooks mixin for integrating a timeout queue into the server loop	??
L4::lrq	
C++ lrq interface, see IRQs for the C interface	??
L4::lrq_eoi	
Interface for sending an unmask message to an object	??
L4::lrq_handler_object	
Server object base class for handling IRQ messages	??
L4::lrqep_t< Derived, BASE, bool >	
Epiface implementation for interrupt handlers	??
L4::Kip::Mem_desc	
Memory descriptors stored in the kernel interface page	??
L4::Kobject	
Base class for all kinds of kernel objects and remote objects, referenced by capabilities	??
L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND >	
Helper class to create an L4Re interface class that is derived from two base classes (see L4::Kobject_t)	??
L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND >	
Helper class to create an L4Re interface class that is derived from three base classes (see L4::Kobject_t)	??
L4::Kobject_demand< T >	
Get the combined server-side resource requirements for all type <i>T</i>	??
L4::Kobject_t< Derived, Base, PROTO, S_DEMAND >	
Helper class to create an L4Re interface class that is derived from a single base class	??
L4::Kobject_typeid< T >	
Meta object for handling access to type information of Kobjects	??
L4::Kobject_typeid< void >	
Minimalistic ID for <code>void</code> interface	??
L4::Kobject_x< Derived, ARGS >	
Generic Kobject inheritance template	??
L4::Lock_guard	
Basic lock guard implementation that prevents forgotten unlocks on exit paths from a method or a block of code	??
L4::Meta	
Meta interface that shall be implemented by each L4Re object and gives access to the dynamic type information for L4Re objects	??
L4::Out_of_memory	
Exception signalling insufficient memory	??

L4::Pager	Pager interface including the lo_pager interface	??
L4::Platform_control	L4 C++ interface for controlling platform-wide properties, see Platform Control C API for the C interface	??
L4::Poll_timeout_counter	Evaluate an expression for a maximum number of times	??
L4::Poll_timeout_kipclock	A polling timeout based on the L4Re clock	??
L4::Proto_t< P >	Data type for defining protocol numbers	??
L4::Rcv_endpoint	Interface for kernel objects that allow to receive IPC from them	??
L4::Registry_iface	Abstract interface for object registries	??
L4::Runtime_error	Exception for an abstract runtime error	??
L4::Scheduler	C++ interface of the Scheduler kernel object, see Scheduler for the C interface	??
L4::Semaphore	C++ Kernel-provided semaphore interface, see Kernel-provided semaphore for the C interface	??
L4::Server< LOOP_HOOKS >	Basic server loop for handling client requests	??
L4::Server_object	Abstract server object to be used with L4::Server and L4::Basic_registry	??
L4::Server_object_t< IFACE, BASE >	Base class (template) for server implementing server objects	??
L4::Server_object_x< Derived, IFACE, BASE >	Helper class to implement p_dispatch based server objects	??
L4::Smart_cap< T, SMART >	Smart capability class	??
L4::String	A null-terminated string container class	??
L4::Task	C++ interface of the Task kernel object, see Task for the C interface	??
L4::Thread	C++ L4 kernel thread interface, see Thread for the C interface	??
L4::Thread::Attr	Thread attributes used for control()	??
L4::Thread::Modify_senders	Class wrapping a list of rules which modify the sender label of IPC messages inbound to this thread	??
L4::Thread_group	C++ L4 kernel thread group interface, see Thread groups for the C interface	??
L4::Triggerable	Interface that allows an object to be triggered by some source	??
L4::Type_info	Dynamic Type Information for L4Re Interfaces	??
L4::Type_info::Demand	Data type for expressing the needed receive buffers at the server-side of an interface	??
L4::Type_info::Demand_t< CAPS, FLAGS, MEM, PORTS >	Template type statically describing demand of receive buffers	??
L4::Type_info::Demand_union_t< D1, D2 >	Template type statically describing the combination of two Demand object	??
L4::Typeid::Detail::_Rpc< OPCODE, O, X >	Empty list of RPCs	??
L4::Typeid::Detail::_Rpc< OPCODE, O, Default_op< R > >::Rpc< Y >	Find the given RPC in the list	??

L4::Typeid::Detail::_Rpc< OPCODE, O, R, X... >	
Non-empty list of RPCs	??
L4::Typeid::Detail::_Rpc< OPCODE, O, R, X... >::Rpc< Y >	
Find the given RPC in the list	??
L4::Typeid::Detail::Rpc_end	
Internal end-of-list marker	??
L4::Typeid::P_dispatch< LIST >	
Use for protocol based dispatch stage	??
L4::Typeid::Raw_ipc< CLASS >	
RPCs list for passing raw incoming IPC to the server object	??
L4::Typeid::Rpc_nocode< OPERATION >	
List of RPCs of an interface using a single operation without an opcode	??
L4::Typeid::Rpc< RPCS >	
Standard list of RPCs of an interface	??
L4::Typeid::Rpc_code< OPCODE_TYPE >	
List of RPCs of an interface using a special opcode type	??
L4::Typeid::Rpc_code< OPCODE_TYPE >::F< RPCS >	??
L4::Typeid::Rpc_sys< ARG >	
List of RPCs typically used for kernel interfaces	??
L4::Types::Bool< V >	
Boolean meta type	??
L4::Types::False	
False meta value	??
L4::Types::Flags< BITS_ENUM, UNDERLYING >	
Template for defining typical Flags bitmaps	??
L4::Types::Flags_ops_t< DT >	
Mixin class to define a set of friend bitwise operators on DT	??
L4::Types::Flags_t< DT, T >	
Template type to define a flags type with bitwise operations	??
L4::Types::Int_for_size< SIZE, bool >	
Metafunction to get an unsigned integral type for the given size	??
L4::Types::Int_for_type< T >	
Metafunction to get an integral type of the same size as T	??
L4::Types::Same< A, B >	
Compare two data types for equality	??
L4::Types::True	
True meta value	??
L4::Uart	
Uart driver abstraction	??
L4::Uart_apb	
Driver for the Advanced Peripheral Bus (APB) UART from the Cortex-M System Design Kit (CMSDK)	??
L4::Unknown_error	
Exception for an unknown condition	??
L4::Vcon	
C++ L4 Vcon interface, see Virtual Console for the C interface	??
L4::Vm	
Virtual machine host address space	??
l4_buf_regs_t	
Encapsulation of the buffer-registers block in the UTCB	??
l4_exc_regs_t	
UTCB structure for exceptions	??
l4_fpage_t	
L4 flexpage type	??
l4_icu_info_t	
Info structure for an ICU	??
l4_icu_msi_info_t	
Info to use for a specific MSI	??

l4_kernel_info_mem_desc_t	Memory descriptor data structure	??
l4_kernel_info_t	L4 Kernel Interface Page	??
l4_msg_regs_t	Encapsulation of the message-register block in the UTCB	??
l4_msgtag_t	Message tag data structure	??
l4_sched_cpu_set_t	CPU sets	??
l4_sched_param_t	Scheduler parameter set	??
l4_snd_fpage_t	Send-flexpage types	??
l4_thread_regs_t	Encapsulation of the thread-control-register block of the UTCB	??
l4_timeout_s	Basic timeout specification	??
l4_timeout_t	Timeout pair	??
l4_vcon_attr_t	Vcon attribute structure	??
l4_vcpu_arch_state_t	Architecture-specific vCPU state	??
l4_vcpu_ipc_regs_t	vCPU message registers	??
l4_vcpu_regs_t	vCPU registers	??
l4_vcpu_state_t	State of a vCPU	??
l4_vm_svm_vmcb_control_area	VMCB structure for SVM VMs	??
l4_vm_svm_vmcb_state_save_area	State save area structure for SVM VMs	??
l4_vm_svm_vmcb_state_save_area_seg	State save area segment selector struct	??
l4_vm_svm_vmcb_t	Control structure for SVM VMs	??
l4_vm_tz_state	State structure for TrustZone VMs	??
l4_vm_vmx_vcpu_infos_t	VMX information members	??
l4_vm_vmx_vcpu_state_t	VMX vCPU state	??
l4_vm_vmx_vcpu_vmcs_t	VMX software VMCS	??
l4_vmx_offset_table_t	Software VMCS field offset table	??
L4drivers::Mmio_register_block< MAX_BITS >	An MMIO block with up to 64-bit register access (32-bit default) and little endian byte order	??
L4drivers::Register_block< MAX_BITS, BLOCK >	Handles a reference to a register block of the given maximum access width	??
L4drivers::Register_block_base< MAX_BITS >	Abstract register block interface	??
L4drivers::Register_block_impl< BASE, MAX_BITS >	Implementation helper for register blocks	??
L4drivers::Register_block_tmpl< BLOCK >	Helper template that translates to the Register_block_base interface	??

L4drivers::Register_tmpl< BITS, BLOCK >	
Single hardware register inside a Register_block_base interface	??
L4drivers::Ro_register_block< MAX_BITS, BLOCK >	
Handles a reference to a read only register block of the given maximum access width	??
L4drivers::Ro_register_tmpl< BITS, BLOCK >	
Single read only register inside a Register_block_base interface	??
L4Re::Cap_alloc	
Capability allocator interface	??
L4Re::Console	
Console class	??
L4Re::Core::Ref_ptr< T, CNT >	
A reference-counting pointer with automatic cleanup	??
L4Re::Dataspace	
Interface for memory-like objects	??
L4Re::Dataspace::F	
Dataspace flags definitions	??
L4Re::Dataspace::Stats	
Information about the dataspace	??
L4Re::Debug_obj	
Debug interface	??
L4Re::Default_event_payload	
Default event stream payload	??
L4Re::Dma_space	
Managed DMA Address Space	??
L4Re::Env	
C++ interface of the initial environment that is provided to an L4 task	??
L4Re::Event	
Event class	??
L4Re::Event_buffer_t< PAYLOAD >	
Event buffer class	??
L4Re::Event_buffer_t< PAYLOAD >::Event	
Event structure used in buffer	??
L4Re::Inhibitor	
Set of inhibitor locks, which inhibit specific actions when held	??
L4Re::Itas	
Interface to the ITAS	??
L4Re::Log	
Log interface class	??
L4Re::Mem_alloc	
Memory allocation interface	??
L4Re::Mem_alloc::Stats	
Statistics about memory-allocator	??
L4Re::Mmio_space	
Interface for memory-like address space accessible via IPC	??
L4Re::Namespace	
Name-space interface	??
L4Re::Ned::Cmd_control	
Direct control interface for Ned	??
L4Re::Parent	
Parent interface	??
L4Re::Random	
Low-bandwidth interface for random number generators	??
L4Re::Rm	
Region map	??
L4Re::Rm::Area	
An area is a range of virtual addresses which is reserved, see L4Re::Rm::reserve_area() . . .	??
L4Re::Rm::F	
Rm flags definitions	??

L4Re::Rm::Region	
A region is a range of virtual addresses which is backed by content	??
L4Re::Rm::Unique_region< T >	
Unique region	??
L4Re::Smart_cap_auto< Unmap_flags >	
Helper for Unique_cap and Unique_del_cap	??
L4Re::Smart_count_cap< Unmap_flags >	
Helper for Ref_cap and Ref_del_cap	??
L4Re::Util::_Cap_alloc	
Adapter to expose the cap allocator implementation as L4Re::Cap_alloc compatible class . . .	??
L4Re::Util::Bitmap< BITS >	
A static bitmap	??
L4Re::Util::Bitmap_base	
Basic bitmap abstraction	??
L4Re::Util::Bitmap_base::Bit	
A writable bit in a bitmap	??
L4Re::Util::Bitmap_base::Char< BITS >	
Helper abstraction for a byte contained in the bitmap	??
L4Re::Util::Bitmap_base::Word< BITS >	
Helper abstraction for a word contained in the bitmap	??
L4Re::Util::Br_manager	
Buffer-register (BR) manager for L4::Server	??
L4Re::Util::Br_manager_hooks	
Predefined server-loop hooks for a server loop using the Br_manager	??
L4Re::Util::Br_manager_timeout_hooks	
Predefined server-loop hooks for a server with using the Br_manager and a timeout queue . .	??
L4Re::Util::Cap_alloc_base	
Capability allocator	??
L4Re::Util::Counter< COUNTER >	
Counter for Counting_cap_alloc with variable data width	??
L4Re::Util::Counter_atomic< COUNTER >	
Thread safe version of counter for Counting_cap_alloc	??
L4Re::Util::Counting_cap_alloc< COUNTERTYPE, Dbg >	
Internal reference-counting cap allocator	??
L4Re::Util::Dataspace_svr	
Dataspace server class	??
L4Re::Util::Event_buffer_consumer_t< PAYLOAD >	
An event buffer consumer	??
L4Re::Util::Event_buffer_t< PAYLOAD >	
Event_buffer utility class	??
L4Re::Util::Event_svr< SVR >	
Convenience wrapper for implementing an event server	??
L4Re::Util::Event_t< PAYLOAD >	
Convenience wrapper for getting access to an event object	??
L4Re::Util::Item_alloc_base	
Item allocator	??
L4Re::Util::Names::Name	
Name class	??
L4Re::Util::Names::Name_space	
Abstract server-side implementation of the L4::Namespace interface	??
L4Re::Util::Object_registry	
A registry that manages server objects and their attached IPC gates for a single server loop for a specific thread	??
L4Re::Util::Ref_cap< T >	
Automatic capability that implements automatic free and unmap of the capability selector . . .	??
L4Re::Util::Ref_del_cap< T >	
Automatic capability that implements automatic free and unmap+delete of the capability selector	??

L4Re::Util::Registry_server< LOOP_HOOKS >	
A server loop object which has a Object_registry included	??
L4Re::Util::Smart_cap_auto< Unmap_flags >	
Helper for Unique_cap and Unique_del_cap	??
L4Re::Util::Smart_count_cap< Unmap_flags >	
Helper for Ref_cap and Ref_del_cap	??
L4Re::Util::Vcon_svr< SVR >	
Console server template class	??
L4Re::Util::Video::Goos_svr	
Goos server class	??
L4Re::Vfs::Be_file	
Boiler plate class for implementing an open file for L4Re::Vfs	??
L4Re::Vfs::Be_file_system	
Boilerplate class for implementing a L4Re::Vfs::File_system	??
L4Re::Vfs::Directory	
Interface for a POSIX file that is a directory	??
L4Re::Vfs::File	
The basic interface for an open POSIX file	??
L4Re::Vfs::File_system	
Basic interface for an L4Re::Vfs file system	??
L4Re::Vfs::Fs	
POSIX File-system related functionality	??
L4Re::Vfs::Generic_file	
The common interface for an open POSIX file	??
L4Re::Vfs::Mman	
Interface for POSIX memory management	??
L4Re::Vfs::Ops	
Interface for the POSIX backends of an application	??
L4Re::Vfs::Regular_file	
Interface for a POSIX file that provides regular file semantics	??
L4Re::Vfs::Special_file	
Interface for a POSIX file that provides special file semantics	??
L4Re::Video::Color_component	
A color component	??
L4Re::Video::Goos	
Class that abstracts framebuffers	??
L4Re::Video::Goos::Info	
Information structure of a Goos	??
L4Re::Video::Pixel_info	
Pixel information	??
L4Re::Video::View	
View of a framebuffer	??
L4Re::Video::View::Info	
Information structure of a view	??
l4re_aux_t	
Auxiliary descriptor	??
l4re_ds_stats_t	
Information about the data space	??
l4re_elf_aux_mword_t	
Auxiliary vector element for a single unsigned data word	??
l4re_elf_aux_t	
Generic header for each auxiliary vector element	??
l4re_elf_aux_vma_t	
Auxiliary vector element for a reserved virtual memory area	??
l4re_env_cap_entry_t	
Entry in the L4Re environment array for the named initial objects	??
l4re_env_t	
Initial environment data structure	??

l4re_event_t	Event structure used in buffer	??
l4re_video_color_component_t	Color component structure	??
l4re_video_goos_info_t	Goos information structure	??
l4re_video_pixel_info_t	Pixel_info structure	??
l4re_video_view_info_t	View information structure	??
l4re_video_view_t	C representation of a goos view	??
l4shmc_ringbuf_head_t	Head field of a ring buffer	??
l4shmc_ringbuf_t	Ring buffer	??
l4util_l4mod_info	Base module structure	??
l4util_l4mod_mod	A single module	??
l4util_mb_addr_range_t	INT-15, AX=E820 style "AddressRangeDescriptor" ...with a "size" parameter on the front which is the structure size - 4, pointing to the next one, up until the full buffer length of the memory map has been reached	??
l4util_mb_apm_t	APM BIOS info	??
l4util_mb_drive_t	Drive Info structure	??
l4util_mb_info_t	MultiBoot Info description	??
l4util_mb_mod_t	The structure type "mod_list" is used by the multiboot_info structure	??
l4util_mb_vbe_ctrl_t	VBE controller information	??
l4util_mb_vbe_mode_t	VBE mode information	??
L4vbus::Device	Device on a L4vbus::Vbus	??
L4vbus::Gpio_module	A Gpio_module groups multiple GPIO pins together	??
L4vbus::Gpio_module::Pin_slice	A slice of the pins provided by this module	??
L4vbus::Gpio_pin	A GPIO pin	??
L4vbus::Icu	Vbus Interrupt controller API	??
L4vbus::Pci_dev	A PCI device	??
L4vbus::Pci_host_bridge	A Pci host bridge	??
L4vbus::Pm< DEC >	Power-management API mixin	??
L4vbus::Vbus	The virtual bus (Vbus) interface	??
l4vbus_device_t	Detailed information about a vbus device	??
l4vbus_resource_t	Description of a single vbus resource	??

L4vcpu::State	
C++ implementation of state word in the vCPU area	??
L4vcpu::Vcpu	
C++ implementation of the vCPU save state area	??
L4virtio::Device	
IPC interface for virtio over L4 IPC	??
L4virtio::Driver::Block_device	
Simple class for accessing a virtio block device synchronously	??
L4virtio::Driver::Block_device::Handle	
Handle to an ongoing request	??
L4virtio::Driver::Device	
Client-side implementation for a general virtio device	??
L4virtio::Driver::Virtio_net_device	
Simple class for accessing a virtio net device	??
L4virtio::Driver::Virtio_net_device::Packet	
Structure for a network packet (header including data) with maximum size, assuming that no extra features have been negotiated	??
L4virtio::Driver::Virtqueue	
Driver-side implementation of a Virtqueue	??
L4virtio::Ptr< T >	
Pointer used in virtio descriptors	??
L4virtio::Svr::Bad_descriptor	
Exception used by Queue to indicate descriptor errors	??
L4virtio::Svr::Block_dev_base< Ds_data >	
Base class for virtio block devices	??
L4virtio::Svr::Block_request< Ds_data >	
A request to read or write data	??
L4virtio::Svr::Console::Control_message	
Virtio console control message	??
L4virtio::Svr::Console::Control_request	
Specialised Virtqueue::Request providing access to control message payload	??
L4virtio::Svr::Console::Device	
Base class implementing a virtio console device with L4Re-based notification handling	??
L4virtio::Svr::Console::Device_port	
A console port with associated read/write state	??
L4virtio::Svr::Console::Features	
Virtio console specific feature bits	??
L4virtio::Svr::Console::Port	
Representation of a Virtio console port	??
L4virtio::Svr::Console::Port::Transition	
State transition from last report state to current state	??
L4virtio::Svr::Console::Virtio_con	
Base class implementing a virtio console functionality	??
L4virtio::Svr::Data_buffer	
Abstract data buffer	??
L4virtio::Svr::Dev_config	
Abstraction for L4-Virtio device config memory	??
L4virtio::Svr::Dev_features	
Type for device feature bitmap	??
L4virtio::Svr::Dev_status	
Type of the device status register	??
L4virtio::Svr::Device_t< DATA >	
Server-side L4-VIRTIO device stub	??
L4virtio::Svr::Driver_mem_list_t< DATA >	
List of driver memory regions assigned to a single L4-VIRTIO transport instance	??
L4virtio::Svr::Driver_mem_region_t< DATA >	
Region of driver memory, that shall be managed locally	??

L4virtio::Svr::Request_processor	Encapsulate the state for processing a VIRTIO request	??
L4virtio::Svr::Scmi::Base_attr_t	SCMI base protocol attributes	??
L4virtio::Svr::Scmi::Base_proto	Base class for the SCMI base protocol	??
L4virtio::Svr::Scmi::Perf_proto	Base class for the SCMI performance protocol	??
L4virtio::Svr::Scmi::Performance_attr_t	SCMI performance protocol attributes	??
L4virtio::Svr::Scmi::Performance_describe_level_t	SCMI performance describe level	??
L4virtio::Svr::Scmi::Performance_describe_levels_n_t	SCMI performance describe levels numbers	??
L4virtio::Svr::Scmi::Performance_domain_attr_t	SCMI performance domain protocol attributes	??
L4virtio::Svr::Scmi::Proto< OBSERV >	Base class for all protocols	??
L4virtio::Svr::Scmi::Scmi_dev	A server implementation of the virtio-scmi protocol	??
L4virtio::Svr::Scmi::Scmi_hdr_t	SCMI header	??
L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >	A server implementation of the virtio-gpio protocol	??
L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::Host_irq	Handler for the host irq	??
L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::Irq_handler	Handler for an gpio pin irq	??
L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::Request_processor	Generic handler for the Virtio requests	??
L4virtio::Svr::Virtio_i2c< Request_handler, Epiface >	A server implementation of the virtio-i2c protocol	??
L4virtio::Svr::Virtio_i2c< Request_handler, Epiface >::Host_irq	Handler for the host irq	??
L4virtio::Svr::Virtio_i2c< Request_handler, Epiface >::Request_processor	Handler for the Virtio requests	??
L4virtio::Svr::Virtio_rng< Rnd_state, Epiface >	A server implementation of the virtio-rng protocol	??
L4virtio::Svr::Virtio_rng< Rnd_state, Epiface >::Host_irq	Handler for the host irq	??
L4virtio::Svr::Virtio_rng< Rnd_state, Epiface >::Request_processor	Handler for the Virtio requests	??
L4virtio::Svr::Virtqueue	Virtqueue implementation for the device	??
L4virtio::Svr::Virtqueue::Head_desc	VIRTIO request, essentially a descriptor from the available ring	??
L4virtio::Virtqueue	Low-level Virtqueue	??
L4virtio::Virtqueue::Avail	Type of available ring, this is read-only for the host	??
L4virtio::Virtqueue::Avail::Flags	Flags of the available ring	??
L4virtio::Virtqueue::Desc	Descriptor in the descriptor table	??
L4virtio::Virtqueue::Desc::Flags	Type for descriptor flags	??
L4virtio::Virtqueue::Used	Used ring	??

L4virtio::Virtqueue::Used::Flags	
Flags for the used ring	??
L4virtio::Virtqueue::Used_elem	
Type of an element of the used ring	??
l4virtio_block_config_t	
Device configuration for block devices	??
l4virtio_block_discard_t	
Structure used for the write zeroes and discard commands	??
l4virtio_block_header_t	
Header structure of a request for a block device	??
l4virtio_config_hdr_t	
L4-VIRTIO config header, provided in shared data space	??
l4virtio_config_queue_t	
Queue configuration entry	??
l4virtio_input_absinfo_t	
Information about the absolute axis in the underlying evdev implementation	??
l4virtio_input_config_t	
Device configuration for input devices	??
l4virtio_input_devids_t	
Device ID information for the device	??
l4virtio_input_event_t	
Single event in event or status queue	??
l4virtio_net_config_t	
Device configuration for network devices	??
l4virtio_net_header_t	
Header structure of a request for a network device	??
L4virtio_port	
A Port on the Virtio Net Switch	??
Mac_addr	
A wrapper class around the value of a MAC address	??
Mac_table< Size >	
Mac_table manages a 1:n association between ports and MAC addresses	??
Net_transfer	
A network request to only a single destination	??
Rm	
Region map	??
Rm::Area	
An area is a range of virtual addresses which is reserved, see L4Re::Rm::reserve_area()	??
Rm::F	
Rm flags definitions	??
Rm::Region	
A region is a range of virtual addresses which is backed by content	??
Rm::Unique_region< T >	
Unique region	??
Switch_factory	
The IPC interface for creating ports	??
Virtio_net	
The Base class of a Port	??
Virtio_net_request	
Abstraction for a network request	??
Virtio_switch	
The Virtio switch contains all ports and processes network requests	??
Virtio_vlan_mangle	
Class for VLAN packet rewriting	??

Chapter 13

File Index

13.1 File List

Here is a list of all documented files with brief descriptions:

pkg/drivers-frst/include/asm_access_gen.h	??
pkg/drivers-frst/include/hw_mmio_register_block	??
pkg/drivers-frst/include/hw_register_block	??
pkg/drivers-frst/include/io_regblock.h	??
pkg/drivers-frst/include/io_regblock_port.h	??
pkg/drivers-frst/include/Makefile	??
pkg/drivers-frst/include/poll_timeout_counter.h	??
pkg/drivers-frst/include/ARCH-amd64/asm_access.h	??
pkg/drivers-frst/include/ARCH-arm/asm_access.h	??
pkg/drivers-frst/include/ARCH-arm64/asm_access.h	??
pkg/drivers-frst/include/ARCH-mips/asm_access.h	??
pkg/drivers-frst/include/ARCH-ppc32/asm_access.h	??
pkg/drivers-frst/include/ARCH-riscv/asm_access.h	??
pkg/drivers-frst/include/ARCH-sparc/asm_access.h	??
pkg/drivers-frst/include/ARCH-x86/asm_access.h	??
pkg/drivers-frst/uart/include/Makefile	??
pkg/drivers-frst/uart/include/uart_16550.h	??
pkg/drivers-frst/uart/include/uart_16550_dw.h	??
pkg/drivers-frst/uart/include/uart_apb.h	??
pkg/drivers-frst/uart/include/uart_base.h	??
pkg/drivers-frst/uart/include/uart_cadence.h	??
pkg/drivers-frst/uart/include/uart_dcc-v6.h	??
pkg/drivers-frst/uart/include/uart_dm.h	??
pkg/drivers-frst/uart/include/uart_dummy.h	??
pkg/drivers-frst/uart/include/uart_geni.h	??
pkg/drivers-frst/uart/include/uart_imx.h	??
pkg/drivers-frst/uart/include/uart_leon3.h	??
pkg/drivers-frst/uart/include/uart_linflex.h	??
pkg/drivers-frst/uart/include/uart_lpuart.h	??
pkg/drivers-frst/uart/include/uart_mvebu.h	??
pkg/drivers-frst/uart/include/uart_of.h	??
pkg/drivers-frst/uart/include/uart_omap35x.h	??
pkg/drivers-frst/uart/include/uart_pl011.h	??
pkg/drivers-frst/uart/include/uart_s3c2410.h	??
pkg/drivers-frst/uart/include/uart_sa1000.h	??

pkg/drivers-frst/uart/include/uart_sbi.h	??
pkg/drivers-frst/uart/include/uart_sh.h	??
pkg/drivers-frst/uart/include/uart_sifive.h	??
pkg/drivers-frst/uart/include/uart_tegra-tcu.h	??
pkg/l4re-core/ned/doc/tutorial.lua	??
pkg/l4re-core/ned/lib/include/cmd_control	??
pkg/l4re-core/ned/lib/include/Makefile	??
pkg/virtio-net-switch/server/switch/debug.h	??
pkg/virtio-net-switch/server/switch/filter.cc	??
pkg/virtio-net-switch/server/switch/filter.h	??
pkg/virtio-net-switch/server/switch/mac_addr.h	??
pkg/virtio-net-switch/server/switch/mac_table.h	??
pkg/virtio-net-switch/server/switch/main.cc	??
pkg/virtio-net-switch/server/switch/Makefile	??
pkg/virtio-net-switch/server/switch/options.cc	??
pkg/virtio-net-switch/server/switch/options.h	??
pkg/virtio-net-switch/server/switch/port.h	??
pkg/virtio-net-switch/server/switch/port_ixl.h	??
pkg/virtio-net-switch/server/switch/port_l4virtio.h	??
pkg/virtio-net-switch/server/switch/request.h	??
pkg/virtio-net-switch/server/switch/request_ixl.h	??
pkg/virtio-net-switch/server/switch/request_l4virtio.h	??
pkg/virtio-net-switch/server/switch/stats.h	??
pkg/virtio-net-switch/server/switch/switch.cc	??
pkg/virtio-net-switch/server/switch/switch.h	??
pkg/virtio-net-switch/server/switch/virtio_net.h	??
pkg/virtio-net-switch/server/switch/virtio_net_buffer.h	??
pkg/virtio-net-switch/server/switch/vlan.h	??
amd64/l4/sys/__kip-arch.h	??
amd64/l4/sys/__vcpu-arch.h	
AMD64-specific vCPU interface	??
amd64/l4/sys/cache.h	
Cache functions	??
amd64/l4/sys/consts.h	
Common L4 constants, AMD64 version	??
amd64/l4/sys/ktrace_events.h	??
amd64/l4/sys/l4int.h	
Fixed sized integer types, AMD64 version	??
amd64/l4/sys/linkage.h	
Linkage	??
amd64/l4/sys/segment.h	
Segment handling (AMD64)	??
amd64/l4/sys/utcb.h	
UTCB definitions for AMD64	??
amd64/l4/sys/vm.h	??
amd64/l4/util/bitops_arch.h	
Amd64 bit manipulation functions	??
amd64/l4/util/cpu.h	
CPU related functions	??
amd64/l4/util/irq.h	
Some PIC and hardware interrupt related functions	??
amd64/l4/util/l4_macros.h	
Main function	??
amd64/l4/util/mbi_argv.h	
Command line handling	??
amd64/l4/util/perform.h	
Performance Monitoring using P5/P6 Measurement Counters	??

amd64/l4/util/ port_io.h	
Port I/O functions	??
amd64/l4/util/ rdtsc.h	
Timestamp counter related functions	??
amd64/l4/util/ spin.h	
Spinning for amd64	??
amd64/l4f/l4/sys/ ipc.h	??
amd64/l4f/l4/sys/ segment.h	
L4f-specific fs/gs manipulation (AMD64)	??
amd64/l4f/l4/util/ port_io.h	
Port I/O functions	??
arm/l4/sys/ __kip-arch.h	??
arm/l4/sys/ __vcpu-arch.h	
ARM-specific vCPU interface	??
arm/l4/sys/ atomic.h	??
arm/l4/sys/ cache.h	
Cache functions	??
arm/l4/sys/ consts.h	
Common L4 constants, arm version	??
arm/l4/sys/ ktrace_events.h	??
arm/l4/sys/ l4int.h	
Fixed sized integer types, arm version	??
arm/l4/sys/ linkage.h	
Linkage	??
arm/l4/sys/ mem_op.h	
Memory access functions (ARM specific)	??
arm/l4/sys/ platform_control.h	??
arm/l4/sys/ task.h	??
arm/l4/sys/ thread.h	
ARM-specific thread related definitions	??
arm/l4/sys/ utcb.h	
UTCB definitions for ARM	??
arm/l4/sys/ vm	??
arm/l4/sys/ vm.h	
ARM virtualization interface	??
arm/l4/util/ bitops_arch.h	
ARM specific implementation of bitops functions	??
arm/l4/util/ cpu.h	
CPU related functions	??
arm/l4/util/ irq.h	
ARM specific implementation of irq functions	??
arm/l4/util/ l4_macros.h	
Main function	??
arm/l4/util/ mbi_argv.h	
Multiboot	??
arm/l4f/l4/sys/ ipc.h	??
arm/l4f/l4/sys/ syscall_defs.h	
Syscall entry definitions	??
amd64/l4/sys/ __kip-arch.h	??
amd64/l4/sys/ __vcpu-arch.h	
ARM64-specific vCPU interface	??
amd64/l4/sys/ cache.h	
Cache functions	??
amd64/l4/sys/ consts.h	
Common L4 constants, arm version	??
amd64/l4/sys/ ktrace_events.h	??
amd64/l4/sys/ l4int.h	
Fixed sized integer types, arm version	??

arm64/l4/sys/linkage.h	??
arm64/l4/sys/platform_control.h	??
arm64/l4/sys/task.h	??
arm64/l4/sys/thread.h	
ARM64-specific thread related definitions	??
arm64/l4/sys/utcb.h	
UTCB definitions for ARM64	??
arm64/l4/sys/vm	??
arm64/l4/sys/vm.h	??
arm64/l4f/l4/sys/ipc.h	??
contrib/libio-io/l4/io/io.h	??
contrib/libio-io/l4/io/types.h	??
l4/cxx/alloc.h	
Alloc list	??
l4/cxx/arith	??
l4/cxx/atomic.h	
Atomic template	??
l4/cxx/avl_map	
AVL map	??
l4/cxx/avl_set	
AVL set	??
l4/cxx/avl_tree	
AVL tree	??
l4/cxx/basic_ostream	
Basic IO stream	??
l4/cxx/basic_vector.h	
Basic vector	??
l4/cxx/bitfield	??
l4/cxx/bitmap	??
l4/cxx/dlist	??
l4/cxx/exceptions	
Base exceptions	??
l4/cxx/hlist	??
l4/cxx/iostream	
IO Stream	??
l4/cxx/ipc_helper	
IPC helper	??
l4/cxx/ipc_server	
IPC server loop	??
l4/cxx/ipc_stream	
IPC stream	??
l4/cxx/ipc_timeout_queue	??
l4/cxx/l4iostream	
L4 IO stream	??
l4/cxx/l4types.h	
L4 Types	??
l4/cxx/list	??
l4/cxx/list_alloc	??
l4/cxx/lock_guard.h	
Lock guard implementation	??
l4/cxx/main_thread	
Main thread	??
l4/cxx/minmax	??
l4/cxx/numeric	??
l4/cxx/observer	??
l4/cxx/pair	
Pair implementation	??
l4/cxx/ref_ptr	??

l4/cxx/ref_ptr_list		
Implementation of a list of ref-ptr-managed objects	..	??
l4/cxx/slab_alloc	..	??
l4/cxx/slist	..	??
l4/cxx/static_container	..	??
l4/cxx/static_vector	..	??
l4/cxx/std_alloc	..	??
l4/cxx/std_ops	..	??
l4/cxx/string	..	??
l4/cxx/string.h		
String	..	??
l4/cxx/thread		
Thread implementation	..	??
l4/cxx/type_list	..	??
l4/cxx/type_traits	..	??
l4/cxx/unique_ptr	..	??
l4/cxx/unique_ptr_list		
Implementation of a list of unique-ptr-managed objects	..	??
l4/cxx/utils	..	??
l4/cxx/weak_ref	..	??
l4/cxx/bits/bst.h		
AVL tree	..	??
l4/cxx/bits/bst_base.h		
AVL tree	..	??
l4/cxx/bits/bst_iter.h		
AVL tree	..	??
l4/cxx/bits/list_basics.h	..	??
l4/cxx/bits/smart_ptr_list.h		
Implementation of a list of smart-pointer-managed objects	..	??
l4/cxx/bits/type_traits.h	..	??
l4/irq/irq.h		
IRQ handling routines	..	??
l4/l4re_vfs/backend	..	??
l4/l4re_vfs/vfs.h	..	??
l4/l4re_vfs/impl/default_ops_impl.h	..	??
l4/l4re_vfs/impl/fd_store.h	..	??
l4/l4re_vfs/impl/fd_store_impl.h	..	??
l4/l4re_vfs/impl/ns_fs.h	..	??
l4/l4re_vfs/impl/ns_fs_impl.h	..	??
l4/l4re_vfs/impl/ro_file.h	..	??
l4/l4re_vfs/impl/ro_file_impl.h	..	??
l4/l4re_vfs/impl/vcon_stream.h	..	??
l4/l4re_vfs/impl/vcon_stream_impl.h	..	??
l4/l4re_vfs/impl/vfs_impl.h	..	??
l4/l4virtio/l4virtio	..	??
l4/l4virtio/virtio.h	..	??
l4/l4virtio/virtio_block.h	..	??
l4/l4virtio/virtio_input.h	..	??
l4/l4virtio/virtio_net.h	..	??
l4/l4virtio/virtqueue	..	??
l4/l4virtio/client/l4virtio	..	??
l4/l4virtio/client/virtio-block	..	??
l4/l4virtio/client/virtio-net	..	??
l4/l4virtio/server/l4virtio	..	??
l4/l4virtio/server/virtio	..	??
l4/l4virtio/server/virtio-block	..	??
l4/l4virtio/server/virtio-console	..	??
l4/l4virtio/server/virtio-console-device	..	??

I4/I4virtio/server/virtio-gpio-device	??
I4/I4virtio/server/virtio-i2c-device	??
I4/I4virtio/server/virtio-rng-device	??
I4/I4virtio/server/virtio-scmi-device	??
I4/libblock-device/block_device_mgr.h	??
I4/libblock-device/debug.h	??
I4/libblock-device/device.h	??
I4/libblock-device/errand.h	??
I4/libblock-device/gpt.h	??
I4/libblock-device/inout_memory.h	??
I4/libblock-device/part_device.h	??
I4/libblock-device/partition.h	??
I4/libblock-device/request.h	??
I4/libblock-device/scheduler.h	??
I4/libblock-device/types.h	??
I4/libblock-device/virtio_client.h	??
I4/libedid/edid.h	??
I4/libgfxbitmap/bitmap.h	
Bitmap renderer header file	??
I4/libgfxbitmap/font.h	
Bitmap font renderer header file	??
I4/libgfxbitmap/support	
Terminal support functionality	??
I4/re/cap_alloc	
Abstract capability-allocator interface	??
I4/re/console	??
I4/re/consts	
Constants	??
I4/re/consts.h	
Constants	??
I4/re/dataspace	
Dataspace interface	??
I4/re/dataspace-sys.h	
Dataspace protocol definition	??
I4/re/dbg_events	??
I4/re/debug	
Debug interface	??
I4/re/dma_space	??
I4/re/elf_aux.h	
Auxiliary information for binaries	??
I4/re/env	
Environment interface	??
I4/re/env.h	
Environment interface	??
I4/re/error_helper	
Error helper	??
I4/re/event	??
I4/re/event-sys.h	??
I4/re/event.h	
Events	??
I4/re/event_enums.h	??
I4/re/inhibitor	??
I4/re/inhibitor-sys.h	??
I4/re/itas	??
I4/re/I4aux.h	
Auxiliary definitions	??
I4/re/log	
Log interface	??

l4/re/log-sys.h	
Log protocol definition	??
l4/re/mem_alloc	
Memory allocator interface	??
l4/re/mem_alloc-sys.h	
Memory allocator protocol definitions	??
l4/re/mmio_space	
Interface definition to emit MMIO-like accesses via IPC	??
l4/re/namespace	
Namespace interface	??
l4/re/namespace-sys.h	
Namespace protocol definitions	??
l4/re/parent	
Parent interface	??
l4/re/parent-sys.h	
Parent protocol definition	??
l4/re/protocols.h	
L4Re Protocol Constants (C version)	??
l4/re/random	
Random number generator interface definition	??
l4/re/remote_access	
	??
l4/re/rm	
Region mapper interface	??
l4/re/rm-sys.h	
Region mapper protocol definitions	??
l4/re/shared_cap	
Shared_cap / Shared_del_cap	??
l4/re/unique_cap	
Unique_cap / Unique_del_cap	??
l4/re/c/dataspace.h	
Data space C interface	??
l4/re/c/debug.h	
Debug C interface	??
l4/re/c/dma_space.h	
DMA space C interface	??
l4/re/c/event.h	
Event C interface	??
l4/re/c/event_buffer.h	
	??
l4/re/c/inhibitor.h	
Inhibitor C interface	??
l4/re/c/log.h	
Log C interface	??
l4/re/c/mem_alloc.h	
Memory allocator C interface	??
l4/re/c/namespace.h	
Namespace functions, C interface	??
l4/re/c/parent.h	
Parent C interface	??
l4/re/c/rm.h	
Region map interface, C interface	??
l4/re/c/util/cap_alloc.h	
Capability allocator C interface	??
l4/re/c/util/kumem_alloc.h	
Kumem allocator utility C interface	??
l4/re/c/util/video/goos_fb.h	
Framebuffer utility functionality	??
l4/re/c/video/colors.h	
	??
l4/re/c/video/goos.h	
	??

I4/re/c/video/view.h	??
I4/re/impl/dataspace_impl.h	
Dataspace client stub implementation	??
I4/re/impl/mem_alloc_impl.h	
Memory allocator client stub implementation	??
I4/re/impl/namespace_impl.h	
Namespace client stub implementation	??
I4/re/impl/rm_impl.h	
Region map client stub implementation	??
I4/re/util/bitmap_cap_alloc	
Bitmap capability allocator	??
I4/re/util/br_manager	??
I4/re/util/cap	
Capability utility functions	??
I4/re/util/cap_alloc	
Capability allocator	??
I4/re/util/cap_alloc_impl.h	
Capability allocator implementation	??
I4/re/util/counting_cap_alloc	
Reference-counting capability allocator	??
I4/re/util/dataspace_svr	??
I4/re/util/debug	??
I4/re/util/env_ns	??
I4/re/util/event	??
I4/re/util/event_buffer	??
I4/re/util/event_svr	??
I4/re/util/icu_svr	??
I4/re/util/item_alloc	
Item allocator	??
I4/re/util/kumem_alloc	
Kumem allocator helper	??
I4/re/util/meta	??
I4/re/util/name_space_svr	??
I4/re/util/object_registry	??
I4/re/util/poll_timeout_kipclock	??
I4/re/util/region_mapping	
Region handling	??
I4/re/util/region_mapping_svr_2	??
I4/re/util/shared_cap	
Shared_cap / Shared_del_cap	??
I4/re/util/unique_cap	
Unique_cap / Unique_del_cap	??
I4/re/util/vcon_svr	??
I4/re/util/video/goos_fb	??
I4/re/util/video/goos_svr	??
I4/re/video/colors	??
I4/re/video/goos	??
I4/re/video/goos-sys.h	
Goos protocol definition	??
I4/re/video/view	??
I4/shmc/ringbuf.h	??
I4/shmc/shmc.h	
Shared memory library header file	??
I4/sigma0/sigma0.h	
Sigma0 interface	??
I4/sys/_kernel_object_impl.h	??
I4/sys/_ktrace-impl.h	
L4 kernel event tracing	??

l4/sys/_l4_fpage.h	??
l4/sys/_platform_control-arm.h	??
l4/sys/_task-arm.h	??
l4/sys/_timeout.h	??
l4/sys/_typeinfo.h	
Type information handling	??
l4/sys/_vcpu-arm.h	??
l4/sys/_vm-arm.h	
Virtualization interface	??
l4/sys/_vm-svm.h	??
l4/sys/_vm-vmx.h	??
l4/sys/arm_smccc	
ARM secure monitor call functions	??
l4/sys/arm_smccc.h	
ARM secure monitor call functions	??
l4/sys/assert.h	
Low-level assert implementation	??
l4/sys/cache.h	
Cache-consistency functions	??
l4/sys/capability	
L4::Cap related definitions	??
l4/sys/compiler.h	
L4 compiler related defines	??
l4/sys/consts.h	
Common constants	??
l4/sys/debugger	
The debugger interface specifies common debugging related definitions	??
l4/sys/debugger.h	
Debugger related definitions	??
l4/sys/err.h	
Error codes	??
l4/sys/exception	
Exception C++ interface	??
l4/sys/factory	
Common factory related definitions	??
l4/sys/factory.h	
Common factory related definitions	??
l4/sys/icu	
Interrupt controller	??
l4/sys/icu.h	
Interrupt controller	??
l4/sys/iommu	??
l4/sys/ipc.h	
Common IPC interface	??
l4/sys/ipc_gate	
The C++ IPC gate interface	??
l4/sys/ipc_gate.h	
The C IPC gate interface, see L4::lpc_gate for the C++ interface	??
l4/sys/irq	
C++ Irq interface	??
l4/sys/irq.h	
C Irq interface	??
l4/sys/kdebug.h	
Functionality for invoking the kernel debugger	??
l4/sys/kdump.h	
Functionality for dumping kernel information	??
l4/sys/kernel_object.h	
Kernel object system calls	??

l4/sys/kip	??
l4/sys/kip.h	
Kernel Info Page access functions	??
l4/sys/kobject	??
l4/sys/ktrace.h	
L4 kernel event tracing	??
l4/sys/l4int.h	
Fixed sized integer types, generic version	??
l4/sys/memdesc.h	
Memory description functions	??
l4/sys/meta	
Meta interface for getting dynamic type information about objects behind capabilities	??
l4/sys/obj_info.h	
Debugger related functions	??
l4/sys/pager	
Pager and lo_pager C++ interface	??
l4/sys/platform_control	
Platform control object	??
l4/sys/platform_control.h	
Platform control object	??
l4/sys/rcv_endpoint	
The C++ Receive endpoint interface	??
l4/sys/rcv_endpoint.h	
Receive endpoint C interface	??
l4/sys/scheduler	
Scheduler object functions	??
l4/sys/scheduler.h	
Scheduler object functions	??
l4/sys/semaphore	
Semaphore class definition	??
l4/sys/semaphore.h	
C semaphore interface	??
l4/sys/smart_capability	
L4::Capability class	??
l4/sys/snd_destination	
The C++ Sender destination interface	??
l4/sys/snd_destination.h	
Sender destination endpoint C interface	??
l4/sys/task	
Common task related definitions	??
l4/sys/task.h	
Common task related definitions	??
l4/sys/thread	
Common thread related definitions	??
l4/sys/thread.h	
Common thread related definitions	??
l4/sys/thread_group	??
l4/sys/thread_group.h	??
l4/sys/typeinfo_svr	
Type information server template	??
l4/sys/types.h	
Common L4 ABI Data Types	??
l4/sys/utcb.h	
UTCB definitions	??
l4/sys/vcon	
C++ Virtual console interface	??
l4/sys/vcon.h	
Virtual console interface	??

l4/sys/vcpu.h	
VCPU API	??
l4/sys/vcpu_context	
Hardware vCPU context interface	??
l4/sys/vcpu_context.h	??
l4/sys/vm	
Virtualization interface	??
l4/sys/cxx/capability.h	??
l4/sys/cxx/consts	??
l4/sys/cxx/ipc_array	??
l4/sys/cxx/ipc_basics	??
l4/sys/cxx/ipc_client	??
l4/sys/cxx/ipc_epiface	??
l4/sys/cxx/ipc_iface	
Interface Definition Language	??
l4/sys/cxx/ipc_legacy	??
l4/sys/cxx/ipc_ret_array	??
l4/sys/cxx/ipc_server	??
l4/sys/cxx/ipc_server_loop	??
l4/sys/cxx/ipc_string	??
l4/sys/cxx/ipc_types	??
l4/sys/cxx/ipc_varg	??
l4/sys/cxx/smart_capability_1x	??
l4/sys/cxx/types	??
l4/util/assert.h	
Some useful assert-style macros	??
l4/util/atomic.h	
Atomic operations header and generic implementations	??
l4/util/backtrace.h	
Backtrace	??
l4/util/base64.h	
Base 64 encoding and decoding functions adapted from Bob Trower 08/04/01	??
l4/util/bitops.h	
Bit manipulation functions	??
l4/util/elf.h	
ELF definition	??
l4/util/getopt.h	
Getopt	??
l4/util/keymap.h	
Event to ASCII key mapping	??
l4/util/kip.h	??
l4/util/kprintf.h	
Printf using the kernel debugger	??
l4/util/l4_macros.h	
Some useful generic macros, L4f version	??
l4/util/l4mod.h	
L4mod structures and constants	??
l4/util/list_alloc.h	
Simple list-based allocator	??
l4/util/lock.h	
Simple lock implementation	??
l4/util/mb_info.h	
Multiboot info structure as defined by GRUB	??
l4/util/parse_cmd.h	
Comfortable command-line parsing	??
l4/util/printf_helpers.h	??
l4/util/rand.h	
Simple Pseudo-Random Number Generator	??

l4/util/splitlog2.h	
Split a range in log2 aligned and size-aligned chunks	??
l4/util/thread.h	
Low-level Thread Functions	??
l4/util/util.h	??
l4/vbus/vbus	??
l4/vbus/vbus.h	
Description of the vbus C API	??
l4/vbus/vbus_generic	??
l4/vbus/vbus_gpio	??
l4/vbus/vbus_gpio-ops.h	??
l4/vbus/vbus_gpio.h	??
l4/vbus/vbus_i2c.h	??
l4/vbus/vbus_inhibitor.h	??
l4/vbus/vbus_interfaces.h	
This header contains the definition of VBUS sub-interfaces and convenience functions to work with the interface IDs	??
l4/vbus/vbus_mcspi.h	??
l4/vbus/vbus_pci	??
l4/vbus/vbus_pci-ops.h	??
l4/vbus/vbus_pci.h	??
l4/vbus/vbus_pm-ops.h	??
l4/vbus/vbus_pm.h	??
l4/vbus/vbus_types.h	
This header file contains descriptions of vbus related data types and constants	??
l4/vbus/vdevice-ops.h	??
l4/vcpu/vcpu	
VCPU support library (C++ interface)	??
l4/vcpu/vcpu.h	
VCPU support library (C interface)	??
x86/l4/sys/__kip-arch.h	??
x86/l4/sys/__vcpu-arch.h	
X86-specific vCPU interface	??
x86/l4/sys/cache.h	
Cache functions	??
x86/l4/sys/consts.h	
Common L4 constants, x86 version	??
x86/l4/sys/ipc-invoke.h	??
x86/l4/sys/ktrace_events.h	??
x86/l4/sys/l4int.h	
Fixed sized integer types, x86 version	??
x86/l4/sys/linkage.h	
Linkage	??
x86/l4/sys/segment.h	
Segment handling (x86)	??
x86/l4/sys/utcb.h	
UTCB definitions for x86	??
x86/l4/sys/vm.h	??
x86/l4/util/bitops_arch.h	
X86 bit manipulation functions	??
x86/l4/util/cpu.h	
CPU related functions	??
x86/l4/util/irq.h	
Some PIC and hardware interrupt related functions	??
x86/l4/util/l4_macros.h	
Main function	??
x86/l4/util/mbi_argv.h	
Command line handling	??

x86/l4/util/ perform.h	
Performance Monitoring using P5/P6 Measurement Counters	??
x86/l4/util/ port_io.h	
X86 port I/O	??
x86/l4/util/ rdtsc.h	
Timestamp counter related functions	??
x86/l4/util/ spin.h	
Spinning for x86	??
x86/l4f/l4/sys/ ipc-l42-gcc3-nopic.h	??
x86/l4f/l4/sys/ ipc.h	
L4 IPC System Calls, x86	??
x86/l4f/l4/sys/ segment.h	
L4f-specific segment manipulation (x86)	??
x86/l4f/l4/util/ port_io.h	
Port I/O functions	??

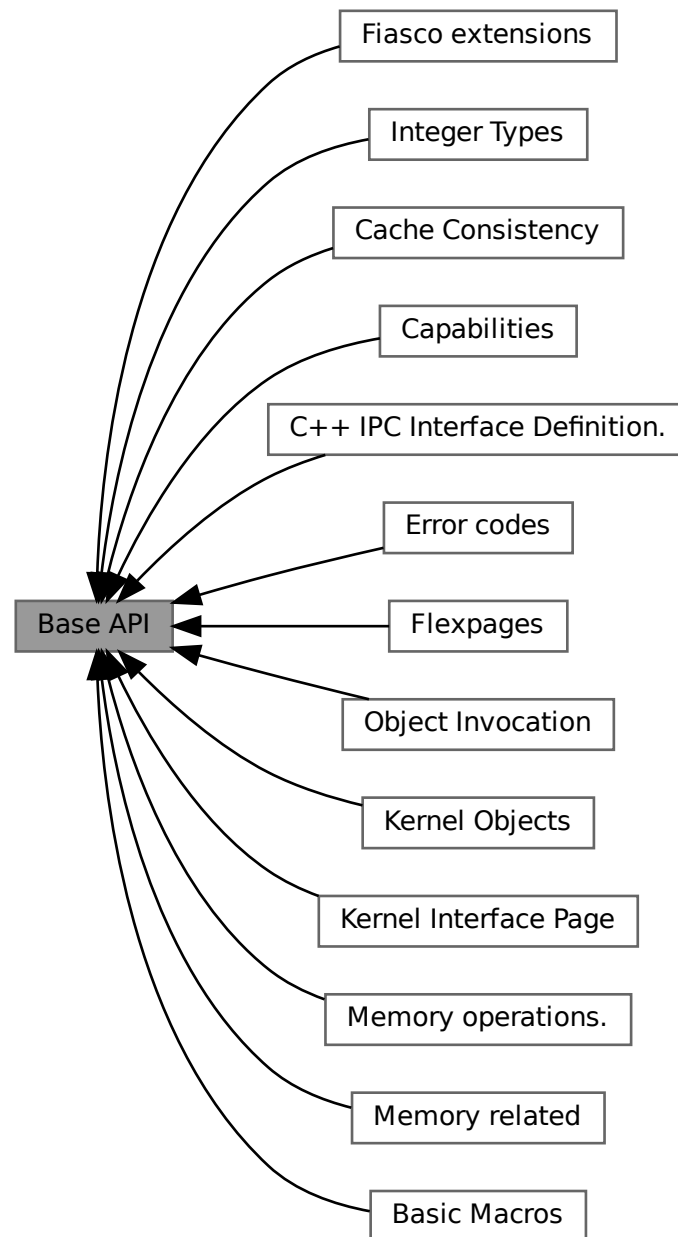
Chapter 14

Topic Documentation

14.1 Base API

Interfaces for all kinds of base functionality.

Collaboration diagram for Base API:



Topics

- Basic Macros ??
 L4 standard macros for header files, function definitions, and public APIs etc.
- Fiasco extensions ??

<i>Extensions of the Fiasco L4 implementation.</i>	
• Flexpages	??
<i>Flexpage-related API.</i>	
• C++ IPC Interface Definition.	??
<i>APIs for defining IPC interfaces using C++ as language.</i>	
• Cache Consistency	??
<i>Various functions for cache consistency.</i>	
• Memory related	??
<i>Memory related constants, data types and functions.</i>	
• Error codes	??
<i>Common error codes.</i>	
• Object Invocation	??
<i>API for L4 object invocation.</i>	
• Kernel Objects	??
<i>API of kernel objects.</i>	
• Kernel Interface Page	??
<i>Kernel Interface Page.</i>	
• Capabilities	??
<i>C interface for capabilities.</i>	
• Memory operations.	??
<i>Operations for memory access.</i>	
• Integer Types	??

Files

- file [cache.h](#)
Cache-consistency functions.
- file [compiler.h](#)
L4 compiler related defines.
- file [consts.h](#)
Common constants.
- file [debugger.h](#)
Debugger related definitions.
- file [factory.h](#)
Common factory related definitions.
- file [icu](#)
Interrupt controller.
- file [icu.h](#)
Interrupt controller.

- file [ipc.h](#)
Common IPC interface.
- file [irq.h](#)
C Irq interface.
- file [kip](#)
- file [kip.h](#)
Kernel Info Page access functions.
- file [memdesc.h](#)
Memory description functions.
- file [semaphore.h](#)
C semaphore interface.
- file [types.h](#)
Common L4 ABI Data Types.
- file [consts.h](#)
Common L4 constants, arm version.
- file [consts.h](#)
Common L4 constants, arm version.
- file [consts.h](#)
Common L4 constants, AMD64 version.
- file [ipc.h](#)
L4 IPC System Calls, x86.
- file [consts.h](#)
Common L4 constants, x86 version.

14.1.1 Detailed Description

Interfaces for all kinds of base functionality.

Some notes on Inter Process Communication (IPC)

IPC in L4 is always synchronous and unbuffered: a message is transferred from the sender to the recipient if and only if the recipient has invoked a corresponding IPC operation. The sender blocks until this happens or a timeout specified by the sender elapsed without the destination becoming ready to receive.

14.1.2 Basic Macros

L4 standard macros for header files, function definitions, and public APIs etc.

Collaboration diagram for Basic Macros:



Files

- file [linkage.h](#)
Linkage.
- file [linkage.h](#)
Linkage.
- file [linkage.h](#)
Linkage.

Macros

- **#define L4_INLINE**
L4 Inline function attribute.
- **#define L4_ALWAYS_INLINE**
Always inline a function.
- **#define L4_BEGIN_DECLS**
Start section with C types and functions.
- **#define L4_END_DECLS**
End section with C types and functions.
- **#define L4_NOTHROW**
Mark a function declaration and definition as never throwing an exception.
- **#define L4_EXPORT**
Attribute to mark functions, variables, and data types as being exported from a library.
- **#define L4_HIDDEN**
Attribute to mark functions, variables, and data types as being explicitly hidden from users of a library.
- **#define L4_CONSTEXPR**
Constexpr function attribute.
- **#define L4_NORETURN**
Noreturn function attribute.
- **#define L4_NOINSTRUMENT**
No instrumentation function attribute.
- **#define L4_LIKELY(x)**
Expression is likely to execute.
- **#define L4_UNLIKELY(x)**
Expression is unlikely to execute.
- **#define L4_STICKY(x)**
Mark symbol sticky (even not there).
- **#define L4_DEPRECATED(s)**
Mark symbol deprecated.
- **#define L4_stringify_helper(x)**
stringify helper.
- **#define L4_stringify(x)**
stringify.
- **#define L4_CV**
Define calling convention.
- **#define L4_CV**
Define calling convention.
- **#define L4_CV**
Define calling convention.
- **#define L4_CV**
Define calling convention.

Functions

- unsigned long [l4_align_stack_for_direct_fncall](#) (unsigned long stack)
Specify the desired alignment of the stack pointer.
- void [l4_barrier](#) (void)
Memory barrier.
- void [l4_mb](#) (void)
Memory barrier.
- void [l4_wmb](#) (void)
Write memory barrier.
- [L4_NORETURN](#) void [l4_infinite_loop](#) (void)
Infinite loop.

14.1.2.1 Detailed Description

[L4](#) standard macros for header files, function definitions, and public APIs etc.

Include File

```
#include <l4/sys/compiler.h>
```

14.1.2.2 Macro Definition Documentation

14.1.2.2.1 L4_EXPORT

```
#define L4_EXPORT
```

Attribute to mark functions, variables, and data types as being exported from a library.

All data types, functions, and global variables that shall be exported from a library shall be marked with this attribute. The default may become to hide everything that is not marked as `L4_EXPORT` from the users of a library and provide the possibility for aggressive optimization of all those internal functionality of a library.

Usage:

```
class L4_EXPORT My_class
{
    ...
};

int L4_EXPORT function(void);

int L4_EXPORT global_data; // global data is not recommended
```

Definition at line 214 of file [compiler.h](#).

Referenced by [l4io_iterate_devices\(\)](#), [l4io_lookup_device\(\)](#), [l4io_lookup_resource\(\)](#), [l4io_release_iomem\(\)](#), [l4io_release_ioport\(\)](#), [l4io_request_all_ioports\(\)](#), [l4io_request_icu\(\)](#), [l4io_request_iomem\(\)](#), [l4io_request_iomem_region\(\)](#), [l4io_request_ioport\(\)](#), [l4io_request_resource_iomem\(\)](#), [l4re_inhibitor_acquire\(\)](#), and [l4re_inhibitor_release\(\)](#).

14.1.2.2.2 L4_HIDDEN

```
#define L4_HIDDEN
```

Attribute to mark functions, variables, and data types as being explicitly hidden from users of a library.

This attribute is intended for functions, data, and data types that shall never be visible outside of a library. In particular, for shared libraries this may result in much faster code within the library and short linking times.

```
class L4_HIDDEN My_class
{
    ...
};

int L4_HIDDEN function(void);

int L4_HIDDEN global_data; // global data is not recommended
```

Definition at line 211 of file [compiler.h](#).

14.1.2.2.3 L4_NOTHROW

```
#define L4_NOTHROW
```

Mark a function declaration and definition as never throwing an exception.

(Also for C code).

This macro shall be used to mark C and C++ functions that never throw any exception. Note that also C functions may throw exceptions according to the compilers ABI and shall be marked with L4_NOTHROW if they never do. In C++ this is equivalent to `throw()`.

```
int foo() L4_NOTHROW;
...
int foo() L4_NOTHROW
{
    ...
    return result;
}
```

Definition at line 161 of file [compiler.h](#).

Referenced by [__kdebug_3_text\(\)](#), [__kdebug_op\(\)](#), [__kdebug_op_1\(\)](#), [__kdebug_text\(\)](#), [enter_kdebug\(\)](#), [l4_msgtag_t::flags\(\)](#), [l4_msgtag_t::has_error\(\)](#), [l4_msgtag_t::is_exception\(\)](#), [l4_msgtag_t::is_io_page_fault\(\)](#), [l4_msgtag_t::is_page_fault\(\)](#), [l4_msgtag_t::is_sigma0\(\)](#), [l4_msgtag_t::items\(\)](#), [l4_arm_smccc_call\(\)](#), [l4_bytes_to_mwords\(\)](#), [l4_cache_clean_data\(\)](#), [l4_cache_coherent\(\)](#), [l4_cache_dma_coherent\(\)](#), [l4_cache_dma_coherent_full\(\)](#), [l4_cache_flush_data\(\)](#), [l4_cache_inv_data\(\)](#), [l4_capability_equal\(\)](#), [l4_capability_next\(\)](#), [l4_debugger_add_image_info\(\)](#), [l4_debugger_get_object_name\(\)](#), [l4_debugger_global_id\(\)](#), [l4_debugger_kobj_to_id\(\)](#), [l4_debugger_query_log_name\(\)](#), [l4_debugger_query_log_typeid\(\)](#), [l4_debugger_query_obj_infos\(\)](#), [l4_debugger_set_object_name\(\)](#), [l4_debugger_switch_log\(\)](#), [l4_error\(\)](#), [l4_factory_create\(\)](#), [l4_factory_create_factory\(\)](#), [l4_factory_create_gate\(\)](#), [l4_factory_create_irq\(\)](#), [l4_factory_create_task\(\)](#), [l4_factory_create_thread\(\)](#), [l4_factory_create_thread_group\(\)](#), [l4_factory_create_vcpu_context\(\)](#), [l4_factory_create_vm\(\)](#), [l4_fpage\(\)](#), [l4_fpage_all\(\)](#), [l4_fpage_contains\(\)](#), [l4_fpage_invalid\(\)](#), [l4_fpage_ioport\(\)](#), [l4_fpage_memaddr\(\)](#), [l4_fpage_obj\(\)](#), [l4_fpage_page\(\)](#), [l4_fpage_rights\(\)](#), [l4_fpage_set_rights\(\)](#), [l4_fpage_size\(\)](#), [l4_fpage_type\(\)](#), [l4_icu_bind\(\)](#), [l4_icu_bind_u\(\)](#), [l4_icu_info\(\)](#), [l4_icu_info_u\(\)](#), [l4_icu_mask\(\)](#), [l4_icu_mask_u\(\)](#), [l4_icu_msi_info\(\)](#), [l4_icu_msi_info_u\(\)](#), [l4_icu_set_mode\(\)](#), [l4_icu_set_mode_u\(\)](#), [l4_icu_unbind\(\)](#), [l4_icu_unbind_u\(\)](#), [l4_icu_unmask\(\)](#), [l4_icu_unmask_u\(\)](#), [l4_iofpage\(\)](#), [l4_ipc\(\)](#), [l4_ipc_call\(\)](#), [l4_ipc_error\(\)](#), [l4_ipc_error_code\(\)](#), [l4_ipc_is_rcv_error\(\)](#), [l4_ipc_is_snd_error\(\)](#), [l4_ipc_receive\(\)](#), [l4_ipc_reply_and_wait\(\)](#), [l4_ipc_send\(\)](#), [l4_ipc_send_and_wait\(\)](#), [l4_ipc_sleep\(\)](#), [l4_ipc_sleep_ms\(\)](#), [l4_ipc_sleep_us\(\)](#), [l4_ipc_timeout\(\)](#), [l4_ipc_to_errno\(\)](#), [l4_ipc_wait\(\)](#), [l4_irq_bind_vcpu\(\)](#), [l4_irq_bind_vcpu_u\(\)](#), [l4_irq_detach\(\)](#), [l4_irq_detach_u\(\)](#), [l4_irq_receive\(\)](#), [l4_irq_receive_u\(\)](#), [l4_irq_trigger\(\)](#), [l4_irq_trigger_u\(\)](#), [l4_irq_unmask\(\)](#), [l4_irq_unmask_u\(\)](#), [l4_irq_wait\(\)](#), [l4_irq_wait_u\(\)](#), [l4_is_fpage_valid\(\)](#), [l4_is_fpage_writable\(\)](#), [l4_is_invalid_cap\(\)](#), [l4_is_valid_cap\(\)](#), [l4_kernel_info_get_mem_desc_end\(\)](#), [l4_kernel_info_get_mem_desc_info\(\)](#), [l4_kernel_info_get_mem_desc_start\(\)](#), [l4_kernel_info_get_mem_desc_subtype\(\)](#), [l4_kernel_info_get_mem_desc_type\(\)](#), [l4_kernel_info_get_mem_descs\(\)](#), [l4_kernel_info_get_num_mem_descs\(\)](#), [l4_kernel_info_set_mem_desc\(\)](#),

[l4_kernel_info_version_offset\(\)](#), [l4_kip\(\)](#), [l4_kip_clock\(\)](#), [l4_kip_clock_lw\(\)](#), [l4_kip_clock_ns\(\)](#), [l4_kip_version\(\)](#),
[l4_kip_version_string\(\)](#), [l4_map_control\(\)](#), [l4_map_obj_control\(\)](#), [l4_msgtag\(\)](#), [l4_msgtag_flags\(\)](#), [l4_msgtag_has_error\(\)](#),
[l4_msgtag_is_exception\(\)](#), [l4_msgtag_is_io_page_fault\(\)](#), [l4_msgtag_is_page_fault\(\)](#), [l4_msgtag_is_sigma0\(\)](#),
[l4_msgtag_items\(\)](#), [l4_msgtag_label\(\)](#), [l4_msgtag_words\(\)](#), [l4_obj_fpage\(\)](#), [l4_platform_ctl_cpu_allow_shutdown\(\)](#),
[l4_platform_ctl_cpu_disable\(\)](#), [l4_platform_ctl_cpu_enable\(\)](#), [l4_platform_ctl_set_task_asid\(\)](#), [l4_platform_ctl_system_shutdown\(\)](#),
[l4_platform_ctl_system_suspend\(\)](#), [l4_rcv_timeout\(\)](#), [l4_round_page\(\)](#), [l4_round_size\(\)](#), [l4_sched_cpu_set\(\)](#),
[l4_sched_param\(\)](#), [l4_scheduler_idle_time\(\)](#), [l4_scheduler_info\(\)](#), [l4_scheduler_info_with_classes\(\)](#), [l4_scheduler_is_online\(\)](#),
[l4_scheduler_run_thread\(\)](#), [l4_semaphore_down\(\)](#), [l4_semaphore_up\(\)](#), [l4_sleep\(\)](#), [l4_sleep_forever\(\)](#), [l4_snd_timeout\(\)](#),
[l4_sndpage_add\(\)](#), [l4_task_add_ku_mem\(\)](#), [l4_task_cap_equal\(\)](#), [l4_task_cap_valid\(\)](#), [l4_task_delete_obj\(\)](#),
[l4_task_map\(\)](#), [l4_task_release_cap\(\)](#), [l4_task_unmap\(\)](#), [l4_task_unmap_batch\(\)](#), [l4_task_vgicc_map\(\)](#),
[l4_thread_arm_set_tpidruro\(\)](#), [l4_thread_control_alien\(\)](#), [l4_thread_control_bind\(\)](#), [l4_thread_control_commit\(\)](#),
[l4_thread_control_exc_handler\(\)](#), [l4_thread_control_pager\(\)](#), [l4_thread_control_start\(\)](#), [l4_thread_ex_regs\(\)](#),
[l4_thread_ex_regs_ret\(\)](#), [l4_thread_ex_regs_ret_u\(\)](#), [l4_thread_ex_regs_u\(\)](#), [l4_thread_group_add\(\)](#), [l4_thread_group_remove\(\)](#),
[l4_thread_modify_sender_add\(\)](#), [l4_thread_modify_sender_commit\(\)](#), [l4_thread_modify_sender_start\(\)](#), [l4_thread_register_del_irq\(\)](#),
[l4_thread_register_doorbell_irq\(\)](#), [l4_thread_stats_time\(\)](#), [l4_thread_switch\(\)](#), [l4_thread_vcpu_control\(\)](#), [l4_thread_vcpu_control_ext\(\)](#),
[l4_thread_vcpu_control_ext_u\(\)](#), [l4_thread_vcpu_control_u\(\)](#), [l4_thread_vcpu_resume_commit\(\)](#), [l4_thread_vcpu_resume_start\(\)](#),
[l4_thread_yield\(\)](#), [l4_timeout\(\)](#), [l4_timeout_abs\(\)](#), [l4_timeout_get\(\)](#), [l4_timeout_is_absolute\(\)](#), [l4_timeout_rel\(\)](#),
[l4_timeout_rel_get\(\)](#), [l4_touch_ro\(\)](#), [l4_touch_rw\(\)](#), [l4_trunc_page\(\)](#), [l4_trunc_size\(\)](#), [l4_usleep\(\)](#), [l4_utcb\(\)](#),
[l4_utcb_br\(\)](#), [l4_utcb_exc\(\)](#), [l4_utcb_exc_is_ex_regs_exception\(\)](#), [l4_utcb_exc_is_ex_regs_exception\(\)](#), [l4_utcb_exc_is_pf\(\)](#),
[l4_utcb_exc_is_pf\(\)](#), [l4_utcb_exc_pc\(\)](#), [l4_utcb_exc_pc\(\)](#), [l4_utcb_exc_pc_set\(\)](#), [l4_utcb_exc_pfa\(\)](#), [l4_utcb_exc_pfa\(\)](#),
[l4_utcb_exc_typeval\(\)](#), [l4_utcb_exc_typeval\(\)](#), [l4_utcb_inherit_fpu\(\)](#), [l4_utcb_mr\(\)](#), [l4_utcb_mr64_idx\(\)](#), [l4_utcb_tcr\(\)](#),
[l4_vcon_get_attr\(\)](#), [l4_vcon_get_attr_u\(\)](#), [l4_vcon_read\(\)](#), [l4_vcon_read_u\(\)](#), [l4_vcon_read_with_flags\(\)](#),
[l4_vcon_send\(\)](#), [l4_vcon_send_u\(\)](#), [l4_vcon_set_attr\(\)](#), [l4_vcon_set_attr_raw\(\)](#), [l4_vcon_set_attr_u\(\)](#), [l4_vcon_write\(\)](#),
[l4_vcon_write_u\(\)](#), [l4_vcpu_check_version\(\)](#), [l4_vm_vmx_clear\(\)](#), [l4_vm_vmx_field_len\(\)](#), [l4_vm_vmx_field_order\(\)](#),
[l4_vm_vmx_get_caps\(\)](#), [l4_vm_vmx_get_caps_default1\(\)](#), [l4_vm_vmx_get_cr2_index\(\)](#), [l4_vm_vmx_get_hw_vmcs\(\)](#),
[l4_vm_vmx_ptr_load\(\)](#), [l4_vm_vmx_read\(\)](#), [l4_vm_vmx_read_16\(\)](#), [l4_vm_vmx_read_32\(\)](#), [l4_vm_vmx_read_64\(\)](#),
[l4_vm_vmx_read_nat\(\)](#), [l4_vm_vmx_set_hw_vmcs\(\)](#), [l4_vm_vmx_write\(\)](#), [l4_vm_vmx_write_16\(\)](#), [l4_vm_vmx_write_32\(\)](#),
[l4_vm_vmx_write_64\(\)](#), [l4_vm_vmx_write_nat\(\)](#), [l4re_debug_obj_debug\(\)](#), [l4re_dma_space_associate\(\)](#), [l4re_dma_space_map\(\)](#),
[l4re_dma_space_unmap\(\)](#), [l4re_ds_allocate\(\)](#), [l4re_ds_clear\(\)](#), [l4re_ds_copy_in\(\)](#), [l4re_ds_flags\(\)](#), [l4re_ds_info\(\)](#),
[l4re_ds_map_info\(\)](#), [l4re_ds_size\(\)](#), [l4re_env\(\)](#), [l4re_env_cap_entry_t::l4re_env_cap_entry_t\(\)](#), [l4re_env_cap_entry_t::l4re_env_cap](#),
[l4re_env_get_cap\(\)](#), [l4re_env_get_cap_e\(\)](#), [l4re_env_get_cap_l\(\)](#), [l4re_event_get_axis_info\(\)](#), [l4re_event_get_buffer\(\)](#),
[l4re_event_get_num_streams\(\)](#), [l4re_event_get_stream_info\(\)](#), [l4re_event_get_stream_info_for_id\(\)](#), [l4re_kip\(\)](#),
[l4re_log_print\(\)](#), [l4re_log_print_srv\(\)](#), [l4re_log_printn\(\)](#), [l4re_log_printn_srv\(\)](#), [l4re_ma_alloc\(\)](#), [l4re_ma_alloc_align\(\)](#),
[l4re_ma_alloc_align_srv\(\)](#), [l4re_ns_query_srv\(\)](#), [l4re_ns_query_to_srv\(\)](#), [l4re_ns_register_obj_srv\(\)](#), [l4re_rm_attach\(\)](#),
[l4re_rm_attach_srv\(\)](#), [l4re_rm_detach\(\)](#), [l4re_rm_detach_ds\(\)](#), [l4re_rm_detach_ds_unmap\(\)](#), [l4re_rm_detach_srv\(\)](#),
[l4re_rm_detach_unmap\(\)](#), [l4re_rm_find\(\)](#), [l4re_rm_find_srv\(\)](#), [l4re_rm_free_area\(\)](#), [l4re_rm_free_area_srv\(\)](#),
[l4re_rm_get_info\(\)](#), [l4re_rm_get_info_srv\(\)](#), [l4re_rm_reserve_area\(\)](#), [l4re_rm_reserve_area_srv\(\)](#), [l4re_rm_show_lists\(\)](#),
[l4re_rm_show_lists_srv\(\)](#), [l4re_util_cap_alloc\(\)](#), [l4re_util_cap_free\(\)](#), [l4re_util_cap_free_um\(\)](#), [l4re_util_cap_last\(\)](#),
[l4re_util_kumem_alloc\(\)](#), [l4re_video_goos_create_buffer\(\)](#), [l4re_video_goos_create_view\(\)](#), [l4re_video_goos_delete_buffer\(\)](#),
[l4re_video_goos_delete_view\(\)](#), [l4re_video_goos_get_static_buffer\(\)](#), [l4re_video_goos_get_view\(\)](#), [l4re_video_goos_info\(\)](#),
[l4re_video_goos_refresh\(\)](#), [l4re_video_view_get_info\(\)](#), [l4re_video_view_refresh\(\)](#), [l4re_video_view_set_info\(\)](#),
[l4re_video_view_set_viewport\(\)](#), [l4re_video_view_stack\(\)](#), [l4util_micros2l4to\(\)](#), [l4vcpu_ext_alloc\(\)](#), [l4vcpu_irq_disable\(\)](#),
[l4vcpu_irq_disable_save\(\)](#), [l4vcpu_irq_enable\(\)](#), [l4vcpu_irq_restore\(\)](#), [l4vcpu_is_irq_entry\(\)](#), [l4vcpu_is_page_fault_entry\(\)](#),
[l4vcpu_print_state\(\)](#), [l4vcpu_wait_for_event\(\)](#), [l4virtio_config_queue\(\)](#), [l4virtio_device_config_ds\(\)](#), [l4virtio_device_notification_irq\(\)](#),
[l4virtio_register_ds\(\)](#), [l4virtio_set_status\(\)](#), [l4_msgtag_t::label\(\)](#), [l4_msgtag_t::label\(\)](#), and [l4_msgtag_t::words\(\)](#).

14.1.2.3 Function Documentation

14.1.2.3.1 l4_align_stack_for_direct_fncall()

```

unsigned long l4_align_stack_for_direct_fncall (
    unsigned long stack) [inline]
  
```

Specify the desired alignment of the stack pointer.

BIGGEST_ALIGNMENT provides the largest alignment ever used for any data type on the target machine. This is normally identical to desired stack alignment. Align stack pointer for directly invoked functions.

The stack needs to be aligned to `L4_STACK_ALIGN` for being able to access certain data on the stack. On x86/AMD64, a function call is performed using the 'call' instruction decrementing the stack pointer and writing the return address onto the stack. The called function considers this when adapting the stack pointer after function entry. If the called function was not invoked by a 'call' instruction, the stack pointer is actually off by a machine word leading to stack alignment issues when executing SSE instructions.

This function fixes the stack pointer for directly invoked functions. For architectures not automatically pushing the stack pointer during a function call, just enforce the `L4_STACK_ALIGN` alignment.

Definition at line 273 of file `compiler.h`.

References `L4_INLINE`.

14.1.2.3.2 `l4_infinite_loop()`

```
L4_NORETURN void l4_infinite_loop (
    void ) [inline]
```

Infinite loop.

Will never return. Use `l4_sleep_forever()` if at all possible.

Definition at line 347 of file `compiler.h`.

References `l4_barrier()`, `L4_INLINE`, and `L4_NORETURN`.

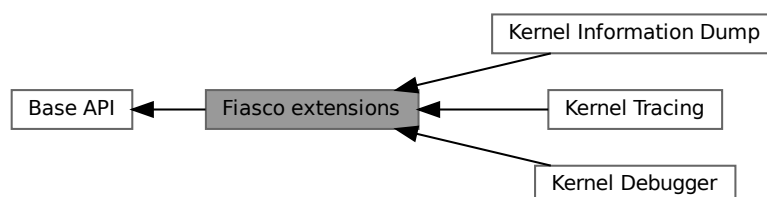
Here is the call graph for this function:



14.1.3 Fiasco extensions

Extensions of the Fiasco `L4` implementation.

Collaboration diagram for Fiasco extensions:



Topics

- Kernel Debugger ??
Kernel debugger related functionality.
- Kernel Information Dump ??
Kernel information dumping related functionality.
- Kernel Tracing ??
Kernel tracing related functionality.

Files

- file [__ktrace-impl.h](#)
L4 kernel event tracing.
- file [ktrace.h](#)
L4 kernel event tracing.
- file [obj_info.h](#)
Debugger related functions.
- file [segment.h](#)
I4f-specific fs/gs manipulation (AMD64).
- file [segment.h](#)
Segment handling (AMD64).
- file [segment.h](#)
I4f-specific segment manipulation (x86).
- file [segment.h](#)
Segment handling (x86).

Functions

- long [fiasco_ldt_set](#) ([l4_cap_idx_t](#) task, void *ldt, unsigned int num_desc, unsigned int entry_number_start, [l4_utcb_t](#) *utcb)
Set LDT segments descriptors.
- long [fiasco_gdt_set](#) ([l4_cap_idx_t](#) thread, void *desc, unsigned int size, unsigned int entry_number_start, [l4_utcb_t](#) *utcb)
Set GDT segment descriptors.
- unsigned [fiasco_gdt_get_entry_offset](#) ([l4_cap_idx_t](#) thread, [l4_utcb_t](#) *utcb)
Return the offset of the entry in the GDT.

14.1.3.1 Detailed Description

Extensions of the Fiasco [L4](#) implementation.

14.1.3.2 Function Documentation

14.1.3.2.1 fiasco_gdt_get_entry_offset()

```
unsigned fiasco_gdt_get_entry_offset (
    l4\_cap\_idx\_t thread,
    l4\_utcb\_t * utcb) [inline]
```

Return the offset of the entry in the GDT.

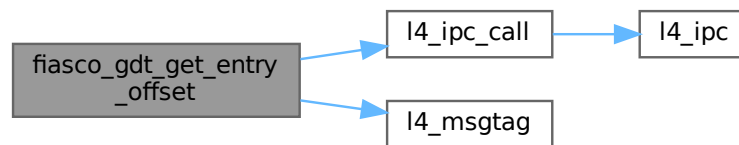
Parameters

<i>thread</i>	Thread to get info from.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Definition at line 166 of file [segment.h](#).

References [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), [l4_msgtag\(\)](#), [L4_PROTO_THREAD](#), and [L4_THREAD_X86_GDT_OP](#).

Here is the call graph for this function:



14.1.3.2.2 fiasco_gdt_set()

```

long fiasco_gdt_set (
    l4_cap_idx_t thread,
    void * desc,
    unsigned int size,
    unsigned int entry_number_start,
    l4_utcb_t * utcb) [inline]

```

Set GDT segment descriptors.

Fiasco supports 4 consecutive entries, starting at the value returned by [fiasco_gdt_get_entry_offset\(\)](#).

Parameters

<i>thread</i>	Thread to set the GDT entry for.
<i>desc</i>	Pointer to GDT descriptors.
<i>size</i>	Size of the descriptors in bytes (multiple of 8).
<i>entry_number_start</i>	Entry number to start (valid values: 0-3).
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

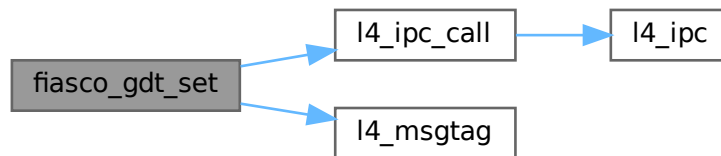
Return values

<0	At least one provided GDT descriptor is considered unsafe by the kernel, and not all selected GDT descriptors have been updated.
<i>L4_EOK</i>	Success.

Definition at line 41 of file [segment.h](#).

References [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), [l4_msgtag\(\)](#), [L4_PROTO_THREAD](#), and [L4_THREAD_X86_GDT_OP](#).

Here is the call graph for this function:



14.1.3.2.3 fiasco_ldt_set()

```

long fiasco_ldt_set (
    l4_cap_idx_t task,
    void * ldt,
    unsigned int num_desc,
    unsigned int entry_number_start,
    l4_utcb_t * utcb) [inline]

```

Set LDT segments descriptors.

Parameters

<i>task</i>	Task to set the segment for.
<i>ldt</i>	Pointer to LDT hardware descriptors.
<i>num_desc</i>	Number of descriptors.
<i>entry_number_start</i>	Entry number to start.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Return values

<code>-L4_ENOSYS</code>	The kernel configuration doesn't support this feature.
<code>-L4_EINVAL</code>	Invalid descriptor or invalid entry number.
<code>L4_EOK</code>	Success.

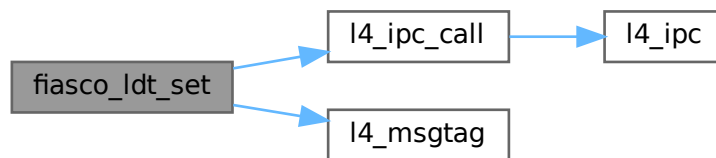
Note

This feature is not available if the kernel is configured with page table isolation.

Definition at line 153 of file [segment.h](#).

References [L4_EINVAL](#), [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), [l4_msgtag\(\)](#), [L4_PROTO_TASK](#), [L4_TASK_LDT_SET_X86_OP](#), [L4_TASK_LDT_X86_ENTRY_SIZE](#), and [L4_TASK_LDT_X86_MAX_ENTRIES](#).

Here is the call graph for this function:

**14.1.3.3 Kernel Debugger**

Kernel debugger related functionality.

Collaboration diagram for Kernel Debugger:

**Files**

- file [kdebug.h](#)

Functionality for invoking the kernel debugger.

Functions

- `l4_msgtag_t l4_debugger_set_object_name (l4_cap_idx_t cap, const char *name) L4_NOTHROW`
Set the name of a kernel object.
- `l4_msgtag_t l4_debugger_get_object_name (l4_cap_idx_t cap, unsigned id, char *name, unsigned size) L4_NOTHROW`
Get name of the kernel object with id `id`.
- `unsigned long l4_debugger_global_id (l4_cap_idx_t cap) L4_NOTHROW`
Get the globally unique ID of the object behind a capability.
- `unsigned long l4_debugger_kobj_to_id (l4_cap_idx_t cap, l4_addr_t kobjp) L4_NOTHROW`
Get the globally unique ID of the object behind the kobject pointer.
- `long l4_debugger_query_log_typeid (l4_cap_idx_t cap, const char *name, unsigned idx) L4_NOTHROW`
Query the log-id for a log type.
- `long l4_debugger_query_log_name (l4_cap_idx_t cap, unsigned idx, char *name, unsigned namelen, char *shortname, unsigned shortnamelen) L4_NOTHROW`
Query the name of a log type given the ID.
- `l4_msgtag_t l4_debugger_switch_log (l4_cap_idx_t cap, const char *name, int on_off) L4_NOTHROW`
Set or unset log.
- `l4_msgtag_t l4_debugger_add_image_info (l4_cap_idx_t cap, l4_addr_t base, const char *name) L4_NOTHROW`
Add loaded image information for a task.

14.1.3.3.1 Detailed Description

Kernel debugger related functionality.

Attention

This API is subject to change!

This is a debugging facility, any call to any function might be invalid. Do not rely on it in any real code.

Include File

```
#include <l4/sys/debugger.h>
```

14.1.3.3.2 Function Documentation

14.1.3.3.2.1 l4_debugger_add_image_info()

```
l4_msgtag_t l4_debugger_add_image_info (
    l4_cap_idx_t cap,
    l4_addr_t base,
    const char * name) [inline]
```

Add loaded image information for a task.

Parameters

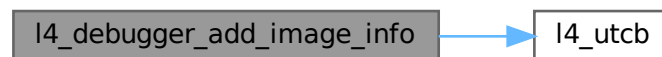
<i>cap</i>	Capability which refers to the task object.
<i>base</i>	Load base address of image.
<i>name</i>	Image base name.

This is a debugging facility, the call might be invalid.

Definition at line 417 of file [debugger.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.3.3.2.2 l4_debugger_get_object_name()

```

l4_msgtag_t l4_debugger_get_object_name (
    l4_cap_idx_t cap,
    unsigned id,
    char * name,
    unsigned size) [inline]
  
```

Get name of the kernel object with Id *id*.

Parameters

	<i>cap</i>	Capability of the debugger object.
	<i>id</i>	Global id of the object whose name is asked.
out	<i>name</i>	Buffer to copy the name into. The buffer must be allocated by the caller.
	<i>size</i>	Length of the <i>name</i> buffer.

Returns

Syscall return tag

Definition at line 410 of file [debugger.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.3.3.2.3 l4_debugger_global_id()

```
unsigned long l4_debugger_global_id (
    l4_cap_idx_t cap) [inline]
```

Get the globally unique ID of the object behind a capability.

Parameters

<i>cap</i>	Capability
------------	------------

Return values

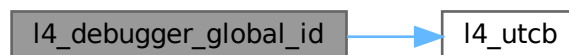
<i>~0UL</i>	Capability is not valid.
<i>otherwise</i>	Global debugger id.

This is a debugging facility, the call might be invalid.

Definition at line 375 of file [debugger.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.3.3.2.4 l4_debugger_kobj_to_id()

```
unsigned long l4_debugger_kobj_to_id (  
    l4_cap_idx_t cap,  
    l4_addr_t kobjp) [inline]
```

Get the globally unique ID of the object behind the kobject pointer.

Parameters

<i>cap</i>	Capability
<i>kobjp</i>	Kobject pointer

Return values

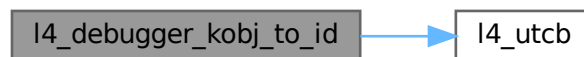
<i>~0UL</i>	The capability or the kobject pointer are invalid.
<i>otherwise</i>	The globally unique id.

This is a debugging facility, the call might be invalid.

Definition at line 381 of file [debugger.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.3.3.2.5 l4_debugger_query_log_name()

```

long l4_debugger_query_log_name (
    l4_cap_idx_t cap,
    unsigned idx,
    char * name,
    unsigned namelen,
    char * shortname,
    unsigned shortnamelen) [inline]
  
```

Query the name of a log type given the ID.

Parameters

<i>cap</i>	Debugger capability.
<i>idx</i>	ID to query.
<i>name</i>	Buffer to copy name to.
<i>namelen</i>	Buffer length of name.
<i>shortname</i>	Buffer to copy shortname to.
<i>shortnamelen</i>	Buffer length of shortname.

Return values

0	Success
<0	Error

This is a debugging facility, the call might be invalid.

Definition at line 394 of file [debugger.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.3.3.2.6 l4_debugger_query_log_typeid()

```
long l4_debugger_query_log_typeid (  
    l4_cap_idx_t cap,  
    const char * name,  
    unsigned idx) [inline]
```

Query the log-id for a log type.

Parameters

<i>cap</i>	Debugger capability
<i>name</i>	Name to query for.
<i>idx</i>	Idx to start searching, start with 0

Returns

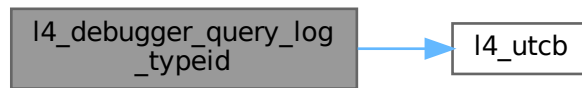
positive ID, or negative error code

This is a debugging facility, the call might be invalid.

Definition at line 387 of file [debugger.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.3.3.2.7 l4_debugger_set_object_name()

```
l4_msgtag_t l4_debugger_set_object_name (  
    l4_cap_idx_t cap,  
    const char * name) [inline]
```

Set the name of a kernel object.

Parameters

<i>cap</i>	Capability which refers to the kernel object.
<i>name</i>	Name of the kernel object that is e.g. displayed in the kernel debugger.

This is a debugging facility, the call might be invalid.

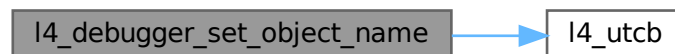
Examples

[examples/libs/shmc/prodcons.c](#), and [examples/sys/aliens/main.c](#).

Definition at line 368 of file [debugger.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.3.3.2.8 l4_debugger_switch_log()

```
l4_msgtag_t l4_debugger_switch_log (  
    l4_cap_idx_t cap,  
    const char * name,  
    int on_off) [inline]
```

Set or unset log.

Parameters

<i>cap</i>	Debugger object.
<i>name</i>	Name of the log type.
<i>on_off</i>	1: turn log on, 0: turn log off

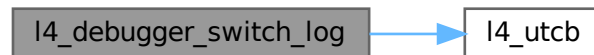
Returns

Syscall return tag

Definition at line 403 of file [debugger.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.3.4 Kernel Information Dump

Kernel information dumping related functionality.

Collaboration diagram for Kernel Information Dump:



Kernel information dumping related functionality.

Functions that dump various kernel internal information to the console. Probably only present in kernel debug builds.

Include File

```
#include <l4/sys/kdump.h>
```

14.1.3.5 Kernel Tracing

Kernel tracing related functionality.

Collaboration diagram for Kernel Tracing:



Functions

- `l4_umword_t fiasco_tbuf_log` (const char *text)
Create new trace-buffer entry with describing <text>.
- `l4_umword_t fiasco_tbuf_log_3val` (const char *text, `l4_umword_t` v1, `l4_umword_t` v2, `l4_umword_t` v3)
Create new trace-buffer entry with describing <text> and three additional values.
- `l4_umword_t fiasco_tbuf_log_binary` (const unsigned char *data)
Create new trace-buffer entry with binary data.
- void `fiasco_tbuf_clear` (void)
Clear trace-buffer.
- void `fiasco_tbuf_dump` (void)
Dump trace-buffer to kernel console.

14.1.3.5.1 Detailed Description

Kernel tracing related functionality.

Attention

This API is subject to change!

This is a tracing facility for the Fiasco kernel trace buffer. Any call to any function might be invalid. Do not rely on it in any real code.

Include File

```
#include <l4/sys/ktrace.h>
```

14.1.3.5.2 Function Documentation

14.1.3.5.2.1 fiasco_tbuf_log()

```
l4_umword_t fiasco_tbuf_log (
    const char * text) [inline]
```

Create new trace-buffer entry with describing <text>.

Parameters

<i>text</i>	Logging text
-------------	--------------

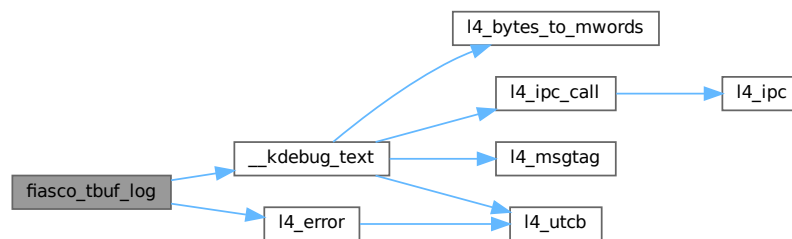
Returns

Pointer to trace-buffer entry

Definition at line 24 of file [__ktrace-impl.h](#).

References [__kdebug_text\(\)](#), and [l4_error\(\)](#).

Here is the call graph for this function:



14.1.3.5.2.2 fiasco_tbuf_log_3val()

```

l4_umword_t fiasco_tbuf_log_3val (
    const char * text,
    l4_umword_t v1,
    l4_umword_t v2,
    l4_umword_t v3) [inline]
  
```

Create new trace-buffer entry with describing <text> and three additional values.

Parameters

<i>text</i>	Logging text
<i>v1</i>	first value
<i>v2</i>	second value
<i>v3</i>	third value

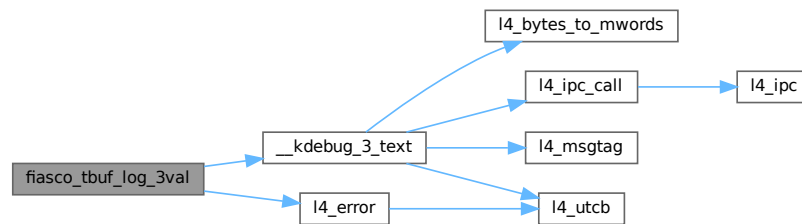
Returns

Pointer to trace-buffer entry

Definition at line 31 of file [__ktrace-impl.h](#).

References [__kdebug_3_text\(\)](#), and [l4_error\(\)](#).

Here is the call graph for this function:

**14.1.3.5.2.3 fiasco_tbuf_log_binary()**

```
l4_umword_t fiasco_tbuf_log_binary (
    const unsigned char * data) [inline]
```

Create new trace-buffer entry with binary data.

Parameters

<i>data</i>	binary data
-------------	-------------

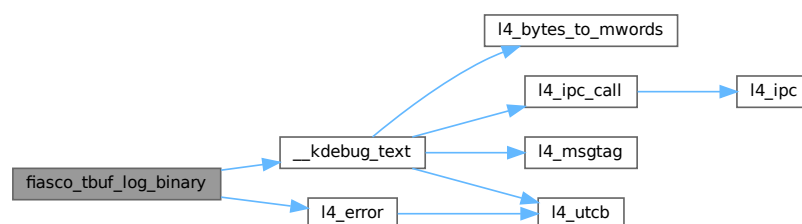
Returns

Pointer to trace-buffer entry

Definition at line 54 of file [__ktrace-impl.h](#).

References [__kdebug_text\(\)](#), and [l4_error\(\)](#).

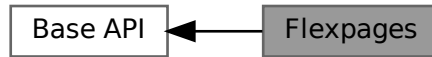
Here is the call graph for this function:



14.1.4 Flexpages

Flexpage-related API.

Collaboration diagram for Flexpages:



Data Structures

- union `l4_fpage_t`
L4 flexpage type.

Enumerations

- enum `L4_fpage_consts` {
`L4_FPAGE_RIGHTS_SHIFT` = 0 , `L4_FPAGE_TYPE_SHIFT` = 4 , `L4_FPAGE_SIZE_SHIFT` = 6 ,
`L4_FPAGE_ADDR_SHIFT` = 12 ,
`L4_FPAGE_RIGHTS_BITS` = 4 , `L4_FPAGE_TYPE_BITS` = 2 , `L4_FPAGE_SIZE_BITS` = 6 , `L4_FPAGE_ADDR_BITS`
= `L4_MWORD_BITS` - `L4_FPAGE_ADDR_SHIFT` ,
`L4_FPAGE_RIGHTS_MASK` , `L4_FPAGE_TYPE_MASK` , `L4_FPAGE_SIZE_MASK` , `L4_FPAGE_ADDR_`
`_MASK` = `~0UL << L4_FPAGE_ADDR_SHIFT` ,
`L4_FPAGE_RIGHTS_ALL` = `L4_FPAGE_RIGHTS_MASK` }
L4 flexpage structure.
- enum { `L4_WHOLE_ADDRESS_SPACE` = 63 }
Constants for flexpages.
- enum `L4_fpage_rights` {
`L4_FPAGE_X` = 1 , `L4_FPAGE_W` = 2 , `L4_FPAGE_RO` = 4 , `L4_FPAGE_RW` = `L4_FPAGE_RO` | `L4_`
`FPAGE_W` ,
`L4_FPAGE_RX` = `L4_FPAGE_RO` | `L4_FPAGE_X` , `L4_FPAGE_RWX` = `L4_FPAGE_RW` | `L4_FPAGE_X` }
Memory and IO port flexpage rights.
- enum `L4_cap_fpage_rights` {
`L4_CAP_FPAGE_W` = 0x1 , `L4_CAP_FPAGE_S` = 0x2 , `L4_CAP_FPAGE_R` = 0x4 , `L4_CAP_FPAGE_RO` =
0x4 ,
`L4_CAP_FPAGE_D` = 0x8 , `L4_CAP_FPAGE_RW` = `L4_CAP_FPAGE_R` | `L4_CAP_FPAGE_W` ,
`L4_CAP_FPAGE_RS` = `L4_CAP_FPAGE_R` | `L4_CAP_FPAGE_S` , `L4_CAP_FPAGE_RWS` = `L4_CAP_`
`_FPAGE_RW` | `L4_CAP_FPAGE_S` ,
`L4_CAP_FPAGE_RWSD` = `L4_CAP_FPAGE_RWS` | `L4_CAP_FPAGE_D` , `L4_CAP_FPAGE_RWD` = `L4_`
`_CAP_FPAGE_RW` | `L4_CAP_FPAGE_D` , `L4_CAP_FPAGE_RSD` = `L4_CAP_FPAGE_RS` | `L4_CAP_`
`FPAGE_D` }
Object flexpage rights.
- enum `L4_fpage_type` { `L4_FPAGE_SPECIAL` = 0 , `L4_FPAGE_MEMORY` = 1 , `L4_FPAGE_IO` = 2 ,
`L4_FPAGE_OBJ` = 3 }
Flexpage type.

- enum `L4_fpage_control` { `L4_FPAGE_CONTROL_OFFSET_SHIFT` = 12 , `L4_FPAGE_CONTROL_MASK` = `~0UL << L4_FPAGE_CONTROL_OFFSET_SHIFT` }
Flexpage map control flags.
- enum { `L4_WHOLE_IOADDRESS_SPACE` = 16 , `L4_IOPORT_MAX` = (1L << `L4_WHOLE_IOADDRESS_SPACE`) }
Special constants for IO flexpages.

Functions

- `l4_fpage_t l4_fpage` (`l4_addr_t` address, unsigned int order, unsigned char rights) `L4_NOTHROW`
Create a memory flexpage.
- `l4_fpage_t l4_fpage_all` (void) `L4_NOTHROW`
Get a flexpage, describing all address spaces at once.
- `l4_fpage_t l4_fpage_invalid` (void) `L4_NOTHROW`
Get an invalid flexpage.
- `l4_fpage_t l4_iofpage` (unsigned long port, unsigned int order) `L4_NOTHROW`
Create an IO-port flexpage.
- `l4_fpage_t l4_obj_fpage` (`l4_cap_idx_t` obj, unsigned int order, unsigned char rights) `L4_NOTHROW`
Create a kernel-object flexpage.
- int `l4_is_fpage_writable` (`l4_fpage_t` fp) `L4_NOTHROW`
Test if the flexpage is writable.
- unsigned `l4_fpage_rights` (`l4_fpage_t` f) `L4_NOTHROW`
Return rights from a flexpage.
- unsigned `l4_fpage_type` (`l4_fpage_t` f) `L4_NOTHROW`
Return type from a flexpage.
- unsigned `l4_fpage_size` (`l4_fpage_t` f) `L4_NOTHROW`
Return size (log2) from a flexpage.
- unsigned long `l4_fpage_page` (`l4_fpage_t` f) `L4_NOTHROW`
Return the page part from a flexpage.
- `l4_addr_t l4_fpage_memaddr` (`l4_fpage_t` f) `L4_NOTHROW`
Return the memory address from the memory flexpage.
- `l4_cap_idx_t l4_fpage_obj` (`l4_fpage_t` f) `L4_NOTHROW`
Return the capability index from the object flexpage.
- unsigned long `l4_fpage_ioport` (`l4_fpage_t` f) `L4_NOTHROW`
Return the IO port number from the IO flexpage.
- `l4_fpage_t l4_fpage_set_rights` (`l4_fpage_t` src, unsigned char new_rights) `L4_NOTHROW`
Set new right in a flexpage.
- int `l4_fpage_contains` (`l4_fpage_t` fpage, `l4_addr_t` addr, unsigned order) `L4_NOTHROW`
Test whether a given range is completely within an fpage.
- unsigned char `l4_fpage_max_order` (unsigned char order, `l4_addr_t` addr, `l4_addr_t` min_addr, `l4_addr_t` max_addr, `l4_addr_t` hotspot=0)
Determine maximum flexpage size of a region.
- int `l4_is_fpage_valid` (`l4_fpage_t` fp) `L4_NOTHROW`
Test if the flexpage is valid.

14.1.4.1 Detailed Description

Flexpage-related API.

A flexpage is a page with a variable size, that can describe memory, IO-Ports (IA32 only), and sets of kernel objects.

A flexpage describes an always size aligned region of an address space. The size is given in a log2 scale. This means the size in elements (bytes for memory, ports for IO-Ports, and capabilities for kernel objects) is always a power of two.

A flexpage also carries type and access right information for the described region. The type information selects the address space in which the flexpage is valid. Access rights have a meaning depending on the specific address space (type).

There exists a special type for defining *receive windows* or for the [l4_task_unmap\(\)](#) method, that can be used to describe all address spaces (all types) with a single flexpage.

Include File

```
#include <l4/sys/types.h>
```

14.1.4.2 Enumeration Type Documentation

14.1.4.2.1 anonymous enum

anonymous enum

Constants for flexpages.

Enumerator

L4_WHOLE_ADDRESS_SPACE	Whole address space size. This value does not only specify the log2 size of the biggest possible memory flexpage. It can be also used as size for a special flexpage to define a flexpage which completely covers all spaces.
------------------------	---

Definition at line 84 of file [__l4_fpage.h](#).

14.1.4.2.2 anonymous enum

anonymous enum

Special constants for IO flexpages.

Enumerator

L4_WHOLE_IOADDRESS_SPACE	Whole I/O address space size. In contrast to L4_WHOLE_ADDRESS_SPACE , this value forms the log2 size of the biggest possible I/O flexpage.
--------------------------	--

L4_IOPORT_MAX	Maximum I/O port address plus 1.
---------------	----------------------------------

Definition at line 314 of file [__l4_fpage.h](#).

14.1.4.2.3 L4_cap_fpage_rights

```
enum L4_cap_fpage_rights
```

Object flexpage rights.

Capabilities are modified or transferred with map and unmap operations. For that, capabilities are wrapped into flexpage objects. The flexpage carries a set of rights the sender wants to hand over to the receiver along with the capability.

For the user only the 'S' and the 'W' right are visible. Other rights such as the 'D' right are internal to the corresponding kernel object and cannot be evaluated by the receiver.

Note that additional object attributes and permissions can be specified in a send item, see [L4_obj_fpage_ctl](#).

Note

A thread can also map a capability from its task's capability table with a reduced set of rights into another slot of its own capability table.

Enumerator

L4_CAP_FPAGE_W	Interface specific 'W' right for capability flexpages. The semantics of the 'W' right is defined by the protocol. For example in case of a dataspace cap, the 'W' right is needed to get a writable dataspace.
L4_CAP_FPAGE_S	Interface specific 'S' right for capability flexpages. The semantics of the 'S' right is defined by the interface. When transferring object capabilities via IPC, the kernel masks this right with the 'S' right of the capability used to address the IPC partner. Thus, the 'S' right of sent capabilities is only transferred if both the flexpage and the IPC gate or thread capability specifying the IPC partner have the 'S' right. For L4::Task::map() , the 'S' right is only transferred if the flexpage, the source and destination task capabilities have the 'S' right.
L4_CAP_FPAGE_R	Read right for capability flexpages. This is always required, otherwise no capability is mapped.
L4_CAP_FPAGE_RO	Read right for capability flexpages. This is always required, otherwise no capability is mapped.
L4_CAP_FPAGE_D	Delete right for capability flexpages. This allows the receiver to delete the corresponding kernel object using unmap() regardless of other tasks still holding a capability to the kernel object. Such capabilities are set to an empty capability if the object is deleted.
L4_CAP_FPAGE_RW	Read and interface specific 'W' right for capability flexpages. The semantics of the 'W' right is defined by the interface. See also L4_CAP_FPAGE_W

L4_CAP_FPAGE_RS	Read and interface specific 'S' right for capability flexpages. The semantics of the 'S' right is defined by the interface. See also L4_CAP_FPAGE_S
L4_CAP_FPAGE_RWS	Read, interface specific 'W', and 'S' rights for capability flexpages. The semantics of the 'W' and 'S' right are defined by the interface. See also L4_CAP_FPAGE_R , L4_CAP_FPAGE_W , and L4_CAP_FPAGE_S
L4_CAP_FPAGE_RWSD	Full rights for capability flexpages. See also L4_CAP_FPAGE_R , L4_CAP_FPAGE_W , L4_CAP_FPAGE_S , and L4_CAP_FPAGE_D
L4_CAP_FPAGE_RWD	Read, write, and delete right for capability flexpages. See also L4_CAP_FPAGE_R , L4_CAP_FPAGE_W , and L4_CAP_FPAGE_D
L4_CAP_FPAGE_RSD	Read, 'S', and delete right for capability flexpages. See also L4_CAP_FPAGE_R , L4_CAP_FPAGE_S , and L4_CAP_FPAGE_D

Definition at line 148 of file [__l4_fpage.h](#).

14.1.4.2.4 L4_fpage_consts

enum [L4_fpage_consts](#)

[L4](#) flexpage structure.

Enumerator

L4_FPAGE_RIGHTS_SHIFT	Access permissions shift.
L4_FPAGE_TYPE_SHIFT	Flexpage type shift (memory, IO port, obj...).
L4_FPAGE_SIZE_SHIFT	Flexpage size shift (log2-based).
L4_FPAGE_ADDR_SHIFT	Page address shift.
L4_FPAGE_RIGHTS_BITS	Access permissions size.
L4_FPAGE_TYPE_BITS	Flexpage type size (memory, IO port, obj...).
L4_FPAGE_SIZE_BITS	Flexpage size size (log2-based).
L4_FPAGE_ADDR_BITS	Page address size.
L4_FPAGE_RIGHTS_MASK	Mask to get the flexpage rights.

L4_FPAGE_RIGHTS_ALL	Specify as flexpage rights during grant.
---------------------	--

Definition at line 48 of file [__l4_fpage.h](#).

14.1.4.2.5 L4_fpage_control

```
enum L4_fpage_control
```

Flexpage map control flags.

Enumerator

L4_FPAGE_CONTROL_OFFSET_SHIFT	Number of bits an index must be shifted or an address must be aligned to in the control word.
L4_FPAGE_CONTROL_MASK	Mask for truncating the lower bits of the send base or the index of the control word.

Definition at line 243 of file [__l4_fpage.h](#).

14.1.4.2.6 L4_fpage_rights

```
enum L4_fpage_rights
```

Memory and IO port flexpage rights.

For IO flexpages, bit 1 and bit 2 are a combined read/write right. In a map operation, the receiver receives the IO port capability when the sender possesses it and at least one of these bits is present. For an unmap operation, the absence of one of those bits is sufficient to unmap the IO port capability.

Note that more memory attributes can be specified in a send item, see [l4_fpage_cacheability_opt_t](#).

Enumerator

L4_FPAGE_X	Executable flexpage.
L4_FPAGE_W	Writable flexpage.
L4_FPAGE_RO	Read-only flexpage.
L4_FPAGE_RW	Read-write flexpage.
L4_FPAGE_RX	Read-execute flexpage.
L4_FPAGE_RWX	Read-write-execute flexpage.

Definition at line 118 of file [__l4_fpage.h](#).

14.1.4.2.7 L4_fpage_type

```
enum L4_fpage_type
```

Flexpage type.

Enumerator

L4_FPAGE_SPECIAL	Special flexpage, either l4_fpage_invalid() or l4_fpage_all() ; only supported by selected interfaces.
L4_FPAGE_MEMORY	Flexpage for memory spaces.
L4_FPAGE_IO	Flexpage for I/O port spaces.
L4_FPAGE_OBJ	Flexpage for object spaces.

Definition at line 230 of file [__l4_fpage.h](#).

14.1.4.3 Function Documentation

14.1.4.3.1 l4_fpage()

```
l4_fpage_t l4_fpage (
    l4_addr_t address,
    unsigned int order,
    unsigned char rights) [inline]
```

Create a memory flexpage.

Parameters

<i>address</i>	Flexpage start address
<i>order</i>	Flexpage size (log2), L4_WHOLE_ADDRESS_SPACE to specify the whole address space (with <i>address</i> 0). The minimum log2 size of a memory flexpage is defined by L4_LOG2_PAGESIZE according to the size of the smallest virtual page supported by the MMU.
<i>rights</i>	Access rights, see L4_fpage_rights

Returns

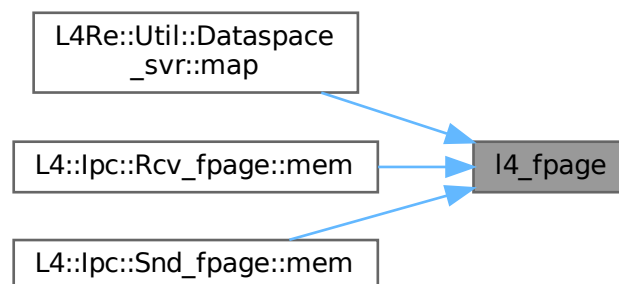
Memory flexpage

Definition at line 703 of file [__l4_fpage.h](#).

References [L4_FPAGE_MEMORY](#), and [L4_NOTHROW](#).

Referenced by [L4Re::Util::Dataspace_svr::map\(\)](#), [L4::lpc::Rcv_fpage::mem\(\)](#), and [L4::lpc::Snd_fpage::mem\(\)](#).

Here is the caller graph for this function:



14.1.4.3.2 l4_fpage_all()

```
l4_fpage_t l4_fpage_all (
    void ) [inline]
```

Get a flexpage, describing all address spaces at once.

Returns

Special *all-spaces* flexpage.

Note

This flexpage can be used to define a receive window where the sender can send objects of any type, or for an unmap item completely covering all spaces of the target task. It does not make sense to use this flexpage as send item.

Definition at line 723 of file [__l4_fpage.h](#).

References [L4_FPAGE_SPECIAL](#), [L4_NOTHROW](#), and [L4_WHOLE_ADDRESS_SPACE](#).

14.1.4.3.3 l4_fpage_contains()

```
int l4_fpage_contains (
    l4_fpage_t fpage,
    l4_addr_t addr,
    unsigned order) [inline]
```

Test whether a given range is completely within an fpage.

Parameters

<i>fpage</i>	Flexpage
<i>addr</i>	Address
<i>order</i>	Size of range in log2.

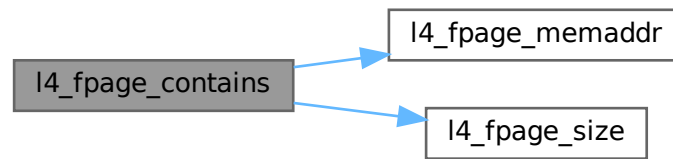
Return values

<i>==0</i>	The range is not completely in the fpage.
<i>!=0</i>	The range is within the fpage.

Definition at line 755 of file [__l4_fpage.h](#).

References [l4_fpage_memaddr\(\)](#), [l4_fpage_size\(\)](#), and [L4_NOTHROW](#).

Here is the call graph for this function:



14.1.4.3.4 l4_fpage_invalid()

```
l4_fpage_t l4_fpage_invalid (
    void ) [inline]
```

Get an invalid flexpage.

Returns

Special *invalid* flexpage.

Definition at line 729 of file [__l4_fpage.h](#).

References [L4_FPAGE_SPECIAL](#), and [L4_NOTHROW](#).

14.1.4.3.5 l4_fpage_ioport()

```
unsigned long l4_fpage_ioport (
    l4_fpage_t f) [inline]
```

Return the IO port number from the IO flexpage.

Parameters

<i>f</i>	Flexpage
----------	----------

Returns

IO port number from the given IO flexpage.

Precondition

f must be an IO flexpage (`l4_fpage_type(f) == L4_FPAGE_IO`) and

The function does not enforce size alignment of the read memory address. The caller must ensure the input fpage is correct.

Definition at line 659 of file [__l4_fpage.h](#).

References [L4_FPAGE_ADDR_SHIFT](#), and [L4_NOTHROW](#).

14.1.4.3.6 l4_fpage_max_order()

```
unsigned char l4_fpage_max_order (
    unsigned char order,
    l4_addr_t addr,
    l4_addr_t min_addr,
    l4_addr_t max_addr,
    l4_addr_t hotspot = 0) [inline]
```

Determine maximum flexpage size of a region.

Parameters

<i>order</i>	Order value to start with (e.g. for memory L4_LOG2_PAGESIZE would be used)
<i>addr</i>	Address to be covered by the flexpage.
<i>min_addr</i>	Start of region / minimal address (including).
<i>max_addr</i>	End of region / maximal address (excluding).
<i>hotspot</i>	(Optional) hot spot.

Returns

Maximum order (log2-size) possible.

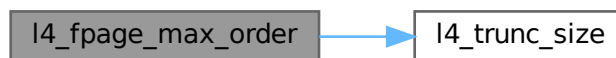
Note

The start address of the flexpage can be determined with `l4_trunc_size(addr, returnvalue)`

Definition at line 763 of file `__l4_fpage.h`.

References `l4_trunc_size()`.

Here is the call graph for this function:

**14.1.4.3.7 l4_fpage_memaddr()**

```
l4_addr_t l4_fpage_memaddr (
    l4_fpage_t f) [inline]
```

Return the memory address from the memory flexpage.

Parameters

<i>f</i>	Flexpage
----------	----------

Returns

Page address from the given memory flexpage.

Precondition

f must be a memory flexpage (`l4_fpage_type(f) == L4_FPAGE_MEMORY`).

The function does not enforce size alignment of the read memory address. The caller must ensure the input fpage is correct.

Definition at line 665 of file `__l4_fpage.h`.

References [L4_NOTHROW](#).

Referenced by [l4_fpage_contains\(\)](#).

Here is the caller graph for this function:

**14.1.4.3.8 l4_fpage_obj()**

```

l4_cap_idx_t l4_fpage_obj (
    l4_fpage_t f) [inline]
  
```

Return the capability index from the object flexpage.

Parameters

<i>f</i>	Flexpage
----------	----------

Returns

Capability index from the given object flexpage.

Precondition

f must be an object flexpage (`l4_fpage_type(f) == L4_FPAGE_OBJ`).

The function does not enforce size alignment of the read memory address. The caller must ensure the input fpage is correct.

Definition at line 671 of file `__l4_fpage.h`.

References [L4_NOTHROW](#).

14.1.4.3.9 l4_fpage_page()

```
unsigned long l4_fpage_page (  
    l4_fpage_t f) [inline]
```

Return the page part from a flexpage.

Parameters

<i>f</i>	Flexpage
----------	----------

Returns

Page part of the given flexpage.

Note

The meaning of the page part depends on the flexpage type.

Definition at line 653 of file [__l4_fpage.h](#).

References [L4_FPAGE_ADDR_SHIFT](#), and [L4_NOTHROW](#).

14.1.4.3.10 l4_fpage_rights()

```
unsigned l4_fpage_rights (  
    l4_fpage_t f) [inline]
```

Return rights from a flexpage.

Parameters

<i>f</i>	Flexpage
----------	----------

Returns

Size part of the given flexpage.

Definition at line 635 of file [__l4_fpage.h](#).

References [L4_FPAGE_RIGHTS_MASK](#), [L4_FPAGE_RIGHTS_SHIFT](#), and [L4_NOTHROW](#).

Referenced by [l4_is_fpage_writable\(\)](#).

Here is the caller graph for this function:



14.1.4.3.11 l4_fpage_set_rights()

```
l4_fpage_t l4_fpage_set_rights (
    l4_fpage_t src,
    unsigned char new_rights) [inline]
```

Set new right in a flexpage.

Parameters

<i>src</i>	Flexpage
<i>new_rights</i>	New rights

Returns

Modified flexpage with new rights.

Definition at line 694 of file [__l4_fpage.h](#).

References [L4_FPAGE_RIGHTS_MASK](#), [L4_FPAGE_RIGHTS_SHIFT](#), [L4_NOTHROW](#), and [l4_fpage_t::raw](#).

Referenced by [L4::lpc::Snd_fpage::io\(\)](#).

Here is the caller graph for this function:



14.1.4.3.12 l4_fpage_size()

```
unsigned l4_fpage_size (
    l4_fpage_t f) [inline]
```

Return size (log2) from a flexpage.

Parameters

<i>f</i>	Flexpage
----------	----------

Returns

Size part of the given flexpage.

See also

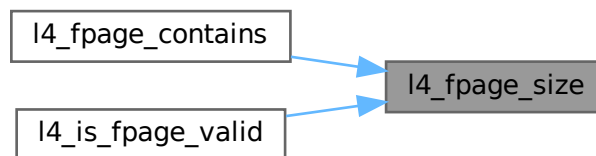
[l4_fpage_memaddr\(\)](#), [l4_fpage_obj\(\)](#), [l4_fpage_ioport\(\)](#)

Definition at line 647 of file [__l4_fpage.h](#).

References [L4_FPAGE_SIZE_SHIFT](#), and [L4_NOTHROW](#).

Referenced by [l4_fpage_contains\(\)](#), and [l4_is_fpage_valid\(\)](#).

Here is the caller graph for this function:



14.1.4.3.13 l4_fpage_type()

```

unsigned l4_fpage_type (
    l4_fpage_t f) [inline]
  
```

Return type from a flexpage.

Parameters

<i>f</i>	Flexpage
----------	----------

Returns

Type part of the given flexpage.

Definition at line 641 of file [__l4_fpage.h](#).

References [L4_FPAGE_TYPE_SHIFT](#), and [L4_NOTHROW](#).

Referenced by [l4_is_fpage_valid\(\)](#).

Here is the caller graph for this function:



14.1.4.3.14 l4_iofpage()

```
l4_fpage_t l4_iofpage (
    unsigned long port,
    unsigned int order) [inline]
```

Create an IO-port flexpage.

Parameters

<i>port</i>	I/O-flexpage port base
<i>order</i>	I/O-flexpage size (log2), L4_WHOLE_IOADDRESS_SPACE to specify the whole I/O address space (with <code>port 0</code>)

Returns

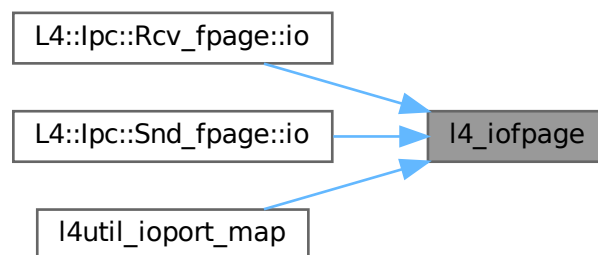
I/O flexpage

Definition at line 709 of file [__l4_fpage.h](#).

References [L4_FPAGE_ADDR_SHIFT](#), [L4_FPAGE_IO](#), [L4_FPAGE_RW](#), and [L4_NOTHROW](#).

Referenced by [L4::lpc::Rcv_fpage::io\(\)](#), [L4::lpc::Snd_fpage::io\(\)](#), and [l4util_ioport_map\(\)](#).

Here is the caller graph for this function:



14.1.4.3.15 l4_is_fpage_valid()

```
int l4_is_fpage_valid (
    l4_fpage_t fp) [inline]
```

Test if the flexpage is valid.

Parameters

<i>fp</i>	Flexpage.
-----------	-----------

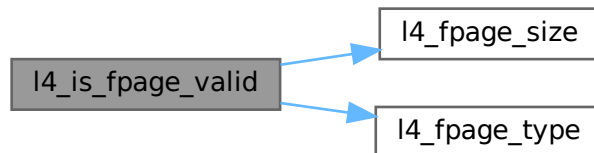
Return values

<i>!=0</i>	if flexpage is valid.
<i>==0</i>	if flexpage is not valid.

Definition at line 788 of file [__l4_fpage.h](#).

References [l4_fpage_size\(\)](#), [L4_FPAGE_SPECIAL](#), [l4_fpage_type\(\)](#), and [L4_NOTHROW](#).

Here is the call graph for this function:

**14.1.4.3.16 l4_is_fpage_writable()**

```
int l4_is_fpage_writable (
    l4_fpage_t fp) [inline]
```

Test if the flexpage is writable.

Parameters

<i>fp</i>	Flexpage.
-----------	-----------

Return values

<i>!=0</i>	if flexpage is writable.
<i>==0</i>	if flexpage is not writable.

Definition at line 736 of file [__l4_fpage.h](#).

References [l4_fpage_rights\(\)](#), [L4_FPAGE_W](#), and [L4_NOTHROW](#).

Here is the call graph for this function:



14.1.4.3.17 l4_obj_fpage()

```

l4_fpage_t l4_obj_fpage (
    l4_cap_idx_t obj,
    unsigned int order,
    unsigned char rights) [inline]
  
```

Create a kernel-object flexpage.

Parameters

<i>obj</i>	Base capability selector.
<i>order</i>	Log2 size (number of capabilities).
<i>rights</i>	Access rights, see L4_cap_fpage_rights

Returns

Flexpage for a set of kernel objects.

Note

[L4_CAP_FPAGE_R](#) is always required, otherwise no capability is mapped.

Examples

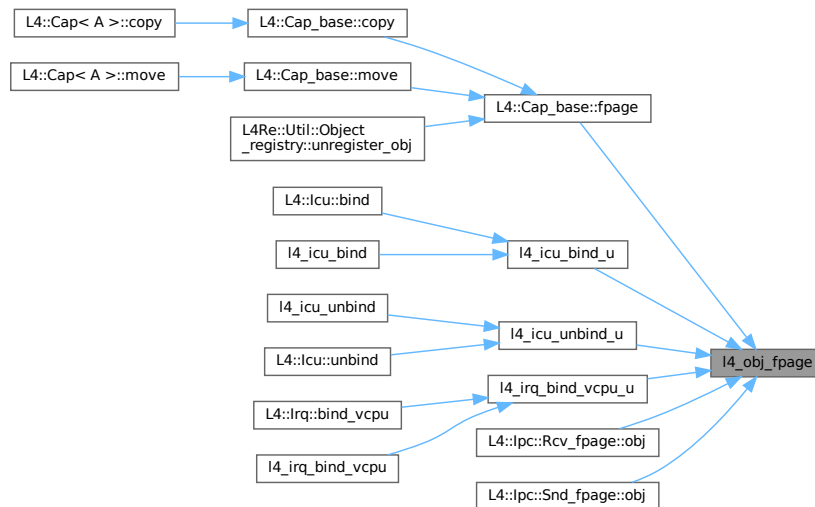
[examples/sys/utcb-ipc/main.c](#).

Definition at line 715 of file [__l4_fpage.h](#).

References [L4_CAP_SHIFT](#), [L4_FPAGE_ADDR_SHIFT](#), [L4_FPAGE_OBJ](#), and [L4_NOTHROW](#).

Referenced by [L4::Cap_base::fpage\(\)](#), [l4_icu_bind_u\(\)](#), [l4_icu_unbind_u\(\)](#), [l4_irq_bind_vcpu_u\(\)](#), [L4::lpc::Rcv_fpage::obj\(\)](#), and [L4::lpc::Snd_fpage::obj\(\)](#).

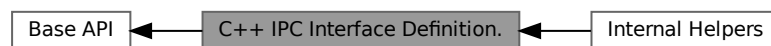
Here is the caller graph for this function:



14.1.5 C++ IPC Interface Definition.

APIs for defining IPC interfaces using C++ as language.

Collaboration diagram for C++ IPC Interface Definition.:



Topics

- Internal Helpers ??

Namespaces

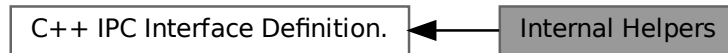
- namespace [L4::Typeid](#)
Definition of interface data-type helpers.

14.1.5.1 Detailed Description

APIs for defining IPC interfaces using C++ as language.

14.1.5.2 Internal Helpers

Collaboration diagram for Internal Helpers:



Data Structures

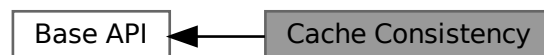
- struct [L4::Types::Bool< V >](#)
Boolean meta type.
- struct [L4::Types::False](#)
False meta value.
- struct [L4::Types::True](#)
True meta value.
- struct [L4::Types::Same< A, B >](#)
Compare two data types for equality.

14.1.5.2.1 Detailed Description

14.1.6 Cache Consistency

Various functions for cache consistency.

Collaboration diagram for Cache Consistency:



Functions

- [L4_BEGIN_DECLS](#) int [l4_cache_clean_data](#) (unsigned long start, unsigned long end) [L4_NOTHROW](#)
Cache clean a range in D-cache; writes back to PoC.
- int [l4_cache_flush_data](#) (unsigned long start, unsigned long end) [L4_NOTHROW](#)
Cache flush a range; writes back to PoC.
- int [l4_cache_inv_data](#) (unsigned long start, unsigned long end) [L4_NOTHROW](#)
Cache invalidate a range; might write back to PoC.
- int [l4_cache_coherent](#) (unsigned long start, unsigned long end) [L4_NOTHROW](#)
Make memory coherent between I-cache and D-cache; writes back to PoU.
- int [l4_cache_dma_coherent](#) (unsigned long start, unsigned long end) [L4_NOTHROW](#)
Make memory coherent for use with external memory; writes back to PoC.
- int [l4_cache_dma_coherent_full](#) (void) [L4_NOTHROW](#)
Make memory coherent for use with external memory; writes back to PoC.

14.1.6.1 Detailed Description

Various functions for cache consistency.

These functions shall be used to ensure that

- all blocks (e.g. CPU cores, devices, DMA engines) are guaranteed to see the same copy of a memory location (Point of Coherency – PoC),
- instruction and data caches of a core are guaranteed to see the same copy of a memory location (Point of Unification – PoU).

Certain functions are NOPs on certain architectures, for example on Intel it's not necessary to explicitly make caches coherent to PoU.

14.1.6.2 Function Documentation

14.1.6.2.1 `l4_cache_clean_data()`

```
L4_BEGIN_DECLS int l4_cache_clean_data (
    unsigned long start,
    unsigned long end) [inline]
```

Cache clean a range in D-cache; writes back to PoC.

Parameters

<i>start</i>	Start of range (inclusive)
<i>end</i>	End of range (exclusive)

Return values

<i>0</i>	on success
<i>-EFAULT</i>	in the case of an unresolved page fault in the given area

Writes back any dirty cache lines in the range but leaves them in the cache and marks the cached copies clean.

Examples

[examples/libs/l4re/c++/shared_ds/ds_clnt.cc](#).

Definition at line 70 of file [cache.h](#).

References [L4_NOTHROW](#).

14.1.6.2.2 `l4_cache_coherent()`

```
int l4_cache_coherent (
    unsigned long start,
    unsigned long end) [inline]
```

Make memory coherent between I-cache and D-cache; writes back to PoU.

Parameters

<i>start</i>	Start of range (inclusive)
<i>end</i>	End of range (exclusive)

Return values

<i>0</i>	on success
<i>-EFAULT</i>	in the case of an unresolved page fault in the given area

Definition at line 94 of file [cache.h](#).

References [L4_NOTHROW](#).

14.1.6.2.3 l4_cache_dma_coherent()

```
int l4_cache_dma_coherent (
    unsigned long start,
    unsigned long end) [inline]
```

Make memory coherent for use with external memory; writes back to PoC.

Parameters

<i>start</i>	Start of range (inclusive)
<i>end</i>	End of range (exclusive)

Return values

<i>0</i>	on success
<i>-EFAULT</i>	in the case of an unresolved page fault in the given area

Definition at line 102 of file [cache.h](#).

References [L4_NOTHROW](#).

14.1.6.2.4 l4_cache_flush_data()

```
int l4_cache_flush_data (
    unsigned long start,
    unsigned long end) [inline]
```

Cache flush a range; writes back to PoC.

Parameters

<i>start</i>	Start of range (inclusive)
<i>end</i>	End of range (exclusive)

Return values

0	on success
-EFAULT	in the case of an unresolved page fault in the given area

Writes back any dirty cache lines and invalidates all cache entries in the range.

Definition at line 78 of file [cache.h](#).

References [L4_NOTHROW](#).

14.1.6.2.5 l4_cache_inv_data()

```
int l4_cache_inv_data (
    unsigned long start,
    unsigned long end) [inline]
```

Cache invalidate a range; might write back to PoC.

Parameters

<i>start</i>	Start of range (inclusive)
<i>end</i>	End of range (exclusive)

Return values

0	on success
-EFAULT	in the case of an unresolved page fault in the given area

Invalidates all cache entries in the range but does not necessarily write back dirty cache lines.

Note

Implementations may choose to write back dirty lines nonetheless if this is more efficient.

Definition at line 86 of file [cache.h](#).

References [L4_NOTHROW](#).

14.1.7 Memory related

Memory related constants, data types and functions.

Collaboration diagram for Memory related:



Macros

- **#define L4_PAGESIZE**
Minimal page size (in bytes).
- **#define L4_PAGEMASK**
Mask for the page number.
- **#define L4_LOG2_PAGESIZE**
Number of bits used for page offset.
- **#define L4_SUPERPAGESIZE**
Size of a large page.
- **#define L4_SUPERPAGEMASK**
Mask for the number of a large page.
- **#define L4_LOG2_SUPERPAGESIZE**
Number of bits used as offset for a large page.
- **#define L4_INVALID_PTR** ((void *)L4_INVALID_ADDR)
Invalid address as pointer type.
- **#define L4_PAGESHIFT** 12
Size of a page, log2-based.
- **#define L4_SUPERPAGESHIFT** 21
Size of a large page, log2-based.
- **#define L4_PAGESHIFT** 12
Size of a page, log2-based.
- **#define L4_SUPERPAGESHIFT** 21
Size of a large page, log2-based.
- **#define L4_PAGESHIFT** 12
Size of a page, log2-based.
- **#define L4_SUPERPAGESHIFT** 21
Size of a large page, log2-based.
- **#define L4_PAGESHIFT** 12
Size of a page log2-based.
- **#define L4_SUPERPAGESHIFT** 22
Size of a large page log2-based.

Enumerations

- enum [l4_addr_consts_t](#) { [L4_INVALID_ADDR](#) = ~0UL }
- Address related constants.*

Functions

- [l4_addr_t l4_trunc_page](#) ([l4_addr_t](#) address) [L4_NOTHROW](#)
Round an address down to the next lower page boundary.
- [l4_addr_t l4_trunc_size](#) ([l4_addr_t](#) address, unsigned char bits) [L4_NOTHROW](#)
Round an address down to the next lower flexpage with size bits.
- [l4_addr_t l4_round_page](#) ([l4_addr_t](#) address) [L4_NOTHROW](#)
Round address up to the next page.
- [l4_addr_t l4_round_size](#) ([l4_addr_t](#) value, unsigned char bits) [L4_NOTHROW](#)
Round value up to the next alignment with bits size.
- unsigned [l4_bytes_to_mwords](#) (unsigned size) [L4_NOTHROW](#)
Determine how many machine words ([l4_umword_t](#)) are required to store a buffer of 'size' bytes.

14.1.7.1 Detailed Description

Memory related constants, data types and functions.

14.1.7.2 Macro Definition Documentation

14.1.7.2.1 L4_LOG2_PAGESIZE

```
#define L4_LOG2_PAGESIZE
```

Number of bits used for page offset.

Size of page in log2.

Definition at line 409 of file [consts.h](#).

Referenced by [L4Re::Dataspace::map\(\)](#), [L4Re::Dataspace::map_region\(\)](#), and [L4Re::Util::Dataspace_svr::page_shift\(\)](#).

14.1.7.2.2 L4_LOG2_SUPERPAGESIZE

```
#define L4_LOG2_SUPERPAGESIZE
```

Number of bits used as offset for a large page.

Size of large page in log2

Definition at line 435 of file [consts.h](#).

14.1.7.2.3 L4_PAGEMASK

```
#define L4_PAGEMASK
```

Mask for the page number.

Note

The most significant bits are set.

Definition at line 400 of file [consts.h](#).

Referenced by [L4virtio::Driver::Device::driver_connect\(\)](#), [l4_round_page\(\)](#), and [l4_trunc_page\(\)](#).

14.1.7.2.4 L4_PAGESHIFT [1/3]

```
#define L4_PAGESHIFT 12
```

Size of a page, log2-based.

Size of a page log2-based.

Definition at line 24 of file [consts.h](#).

14.1.7.2.5 L4_PAGESHIFT [2/3]

```
#define L4_PAGESHIFT 12
```

Size of a page, log2-based.

Size of a page log2-based.

Examples

[examples/libs/l4re/c++/mem_alloc/ma+rm.cc](#), [examples/libs/l4re/c/ma+rm.c](#), [examples/libs/l4re/streammap/client.cc](#),
and [examples/libs/l4re/streammap/server.cc](#).

Definition at line 26 of file [consts.h](#).

Referenced by [L4Re::Rm::attach\(\)](#), [L4virtio::Driver::Device::driver_connect\(\)](#), [L4Re::Rm::free_area\(\)](#), [L4Re::Util::Dataspace_svr::map](#)
[L4Re::Rm::reserve_area\(\)](#), [L4Re::Rm::reserve_area\(\)](#), [L4virtio::Driver::Block_device::setup_device\(\)](#), and
[L4virtio::Driver::Virtio_net_device::setup_device\(\)](#).

14.1.7.2.6 L4_PAGESHIFT [3/3]

```
#define L4_PAGESHIFT 12
```

Size of a page, log2-based.

Size of a page log2-based.

Definition at line 26 of file [consts.h](#).

14.1.7.2.7 L4_SUPERPAGEMASK

```
#define L4_SUPERPAGEMASK
```

Mask for the number of a large page.

Note

The most significant bits are set.

Definition at line 427 of file [consts.h](#).

14.1.7.2.8 L4_SUPERPAGESHIFT [1/3]

```
#define L4_SUPERPAGESHIFT 21
```

Size of a large page, log2-based.

Size of a large page log2-based.

Definition at line 30 of file [consts.h](#).

14.1.7.2.9 L4_SUPERPAGESHIFT [2/3]

```
#define L4_SUPERPAGESHIFT 21
```

Size of a large page, log2-based.

Size of a large page log2-based.

Examples

[examples/libs/l4re/c++/mem_alloc/ma+rm.cc](#), and [examples/libs/l4re/c/ma+rm.c](#).

Definition at line 31 of file [consts.h](#).

Referenced by [L4virtio::Svr::Driver_mem_region_t< Mem_region_info >::Driver_mem_region_t\(\)](#).

14.1.7.2.10 L4_SUPERPAGESHIFT [3/3]

```
#define L4_SUPERPAGESHIFT 21
```

Size of a large page, log2-based.

Size of a large page log2-based.

Definition at line 31 of file [consts.h](#).

14.1.7.2.11 L4_SUPERPAGESIZE

```
#define L4_SUPERPAGESIZE
```

Size of a large page.

A large page is a *super page* on IA32 or a *section* on ARM.

Definition at line 418 of file [consts.h](#).

Referenced by [L4virtio::Driver::Device::driver_connect\(\)](#).

14.1.7.3 Enumeration Type Documentation

14.1.7.3.1 l4_addr_consts_t

```
enum l4_addr_consts_t
```

Address related constants.

Enumerator

L4_INVALID_ADDR	Invalid address.
-----------------	------------------

Definition at line 503 of file [consts.h](#).

14.1.7.4 Function Documentation

14.1.7.4.1 l4_bytes_to_mwords()

```
unsigned l4_bytes_to_mwords (
    unsigned size) [inline]
```

Determine how many machine words ([l4_umword_t](#)) are required to store a buffer of 'size' bytes.

Parameters

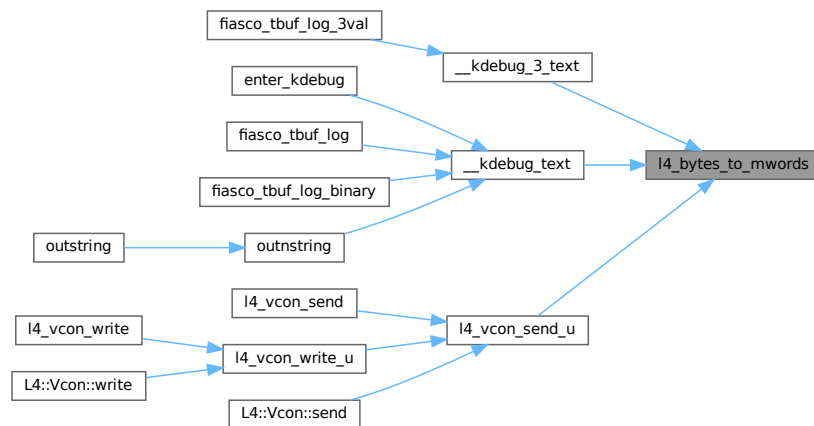
size	The number of bytes to be translated into machine words.
------	--

Definition at line 496 of file [consts.h](#).

References [L4_INLINE](#), and [L4_NOTHROW](#).

Referenced by [__kdebug_3_text\(\)](#), [__kdebug_text\(\)](#), and [l4_vcon_send_u\(\)](#).

Here is the caller graph for this function:



14.1.7.4.2 l4_round_page()

```
l4_addr_t l4_round_page (
    l4_addr_t address) [inline]
```

Round address up to the next page.

The address is rounded up to the next minimal page boundary. On most architectures this is a 4k page. Check [L4_PAGESIZE](#) for the minimal page size.

Parameters

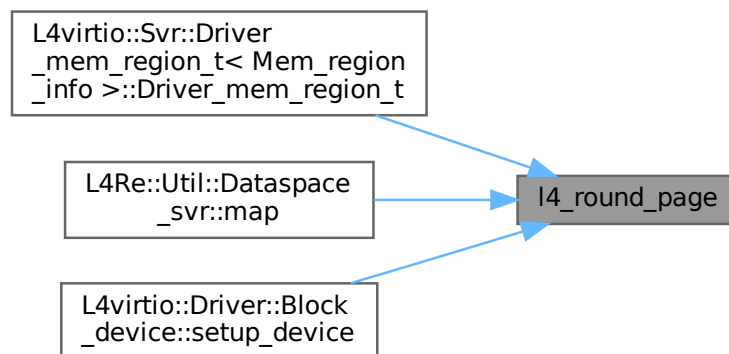
<i>address</i>	The address to round up.
----------------	--------------------------

Definition at line 473 of file [consts.h](#).

References [L4_INLINE](#), [L4_NOTHROW](#), [L4_PAGEMASK](#), and [L4_PAGESIZE](#).

Referenced by [L4virtio::Svr::Driver_mem_region_t< Mem_region_info >::Driver_mem_region_t\(\)](#), [L4Re::Util::Dataspace_svr::map\(\)](#), and [L4virtio::Driver::Block_device::setup_device\(\)](#).

Here is the caller graph for this function:



14.1.7.4.3 l4_round_size()

```
l4_addr_t l4_round_size (
    l4_addr_t value,
    unsigned char bits) [inline]
```

Round value up to the next alignment with *bits* size.

Parameters

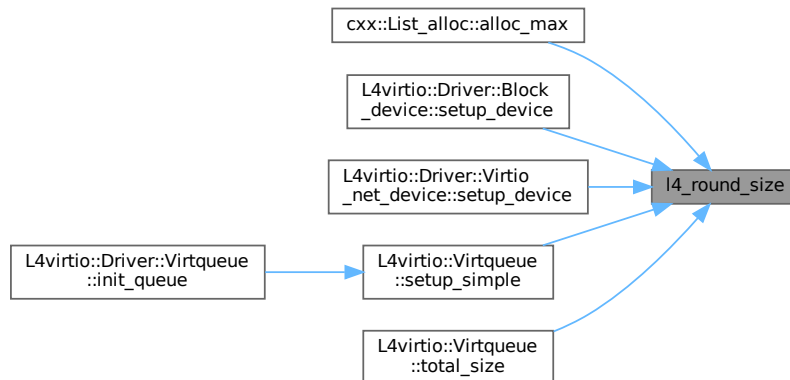
<i>value</i>	The value to round up to the next size-alignment.
<i>bits</i>	The size of the alignment (log2).

Definition at line 484 of file [consts.h](#).

References [L4_INLINE](#), and [L4_NOTHROW](#).

Referenced by [cxx::List_alloc::alloc_max\(\)](#), [L4virtio::Driver::Block_device::setup_device\(\)](#), [L4virtio::Driver::Virtio_net_device::setup_device\(\)](#), [L4virtio::Virtqueue::setup_simple\(\)](#), and [L4virtio::Virtqueue::total_size\(\)](#).

Here is the caller graph for this function:



14.1.7.4.4 l4_trunc_page()

```
l4_addr_t l4_trunc_page (
    l4_addr_t address) [inline]
```

Round an address down to the next lower page boundary.

The address is rounded down to the next lower minimal page boundary. On most architectures this is a 4k page. Check [L4_PAGESIZE](#) for the minimal page size.

Parameters

<i>address</i>	The address to round.
----------------	-----------------------

Examples

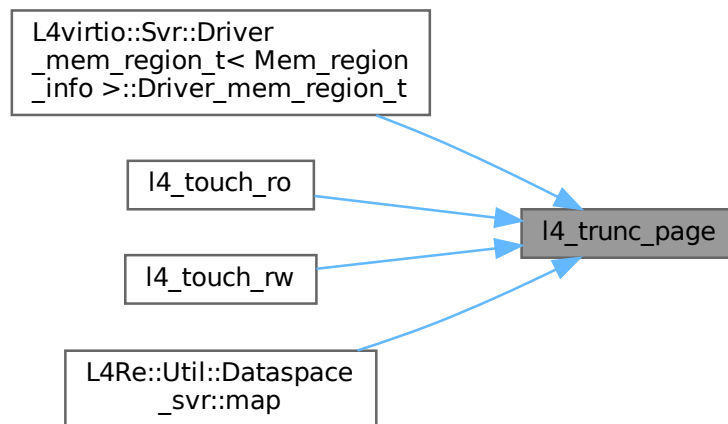
[examples/libs/l4re/c++/mem_alloc/ma+rm.cc](#), and [examples/libs/l4re/c/ma+rm.c](#).

Definition at line 448 of file [consts.h](#).

References [L4_INLINE](#), [L4_NOTHROW](#), and [L4_PAGEMASK](#).

Referenced by [L4virtio::Svr::Driver_mem_region_t< Mem_region_info >::Driver_mem_region_t\(\)](#), [l4_touch_ro\(\)](#), [l4_touch_rw\(\)](#), and [L4Re::Util::Dataspace_svr::map\(\)](#).

Here is the caller graph for this function:



14.1.7.4.5 l4_trunc_size()

```
l4_addr_t l4_trunc_size (
    l4_addr_t address,
    unsigned char bits) [inline]
```

Round an address down to the next lower flexpage with size *bits*.

Parameters

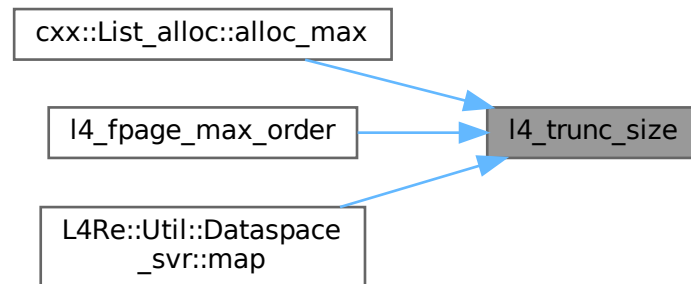
<i>address</i>	The address to round.
<i>bits</i>	The size of the flexpage (log2).

Definition at line 459 of file `consts.h`.

References `L4_INLINE`, and `L4_NOTHROW`.

Referenced by `cxx::List_alloc::alloc_max()`, `l4_fpage_max_order()`, and `L4Re::Util::Dataspace_svr::map()`.

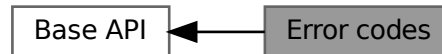
Here is the caller graph for this function:



14.1.8 Error codes

Common error codes.

Collaboration diagram for Error codes:



Enumerations

- enum `l4_error_code_t` {
`L4_EOK` = 0 , `L4_EPERM` = 1 , `L4_ENOENT` = 2 , `L4_EIO` = 5 ,
`L4_ENXIO` = 6 , `L4_E2BIG` = 7 , `L4_EAGAIN` = 11 , `L4_ENOMEM` = 12 ,
`L4_EACCESS` = 13 , `L4_EFAULT` = 14 , `L4_EBUSY` = 16 , `L4_EEXIST` = 17 ,
`L4_ENODEV` = 19 , `L4_ENOTDIR` = 20 , `L4_EINVAL` = 22 , `L4_ENOSPC` = 28 ,
`L4_ERANGE` = 34 , `L4_ENAMETOOLONG` = 36 , `L4_ENOSYS` = 38 , `L4_EBADPROTO` = 39 ,
`L4_EADDRNOTAVAIL` = 99 , `L4_ERRNOMAX` = 100 , `L4_ENOREPLY` = 1000 , `L4_MSGTOOSHORT` =
1001 ,
`L4_MSGTOOLONG` = 1002 , `L4_MSGMISSARG` = 1003 , `L4_EIPC_LO` = 2000 , `L4_EIPC_HI` = 2000 +
0x1f }

L4 error codes.

14.1.8.1 Detailed Description

Common error codes.

Include File

```
#include <l4/sys/err.h>
```

14.1.8.2 Enumeration Type Documentation

14.1.8.2.1 l4_error_code_t

enum [l4_error_code_t](#)

[L4](#) error codes.

Those error codes are used by both the kernel and the user programs.

Enumerator

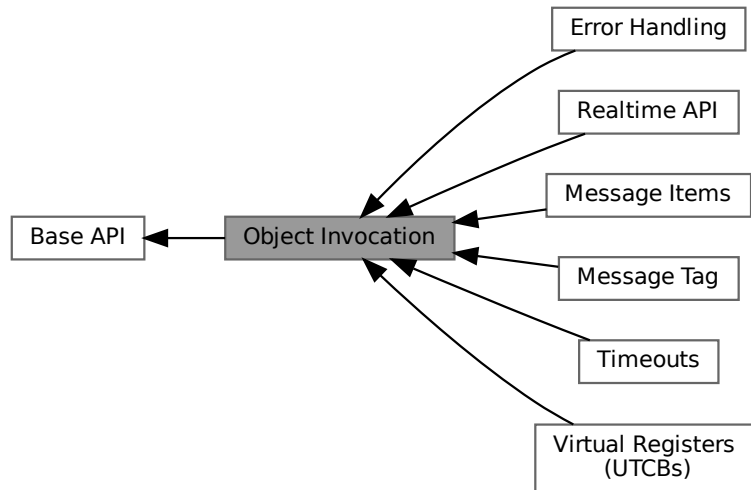
L4_EOK	Ok.
L4_EPERM	No permission.
L4_ENOENT	No such entity.
L4_EIO	I/O error.
L4_ENXIO	No such device or address.
L4_E2BIG	Argument value too big.
L4_EAGAIN	Try again.
L4_ENOMEM	No memory.
L4_EACCESS	Permission denied.
L4_EFAULT	Invalid memory address.
L4_EBUSY	Object currently busy, try later.
L4_EEXIST	Already exists.
L4_ENODEV	No such thing.
L4_ENOTDIR	Not a directory.
L4_EINVAL	Invalid argument.
L4_ENOSPC	No space left on device.
L4_ERANGE	Range error.
L4_ENAMETOOLONG	Name too long.
L4_ENOSYS	No sys.
L4_EBADPROTO	Unsupported protocol.
L4_EADDRNOTAVAIL	Address not available.
L4_ERRNOMAX	Maximum error value.
L4_ENOREPLY	No reply.
L4_MSGTOOSHORT	Message too short.
L4_MSGTOOLONG	Message too long.
L4_MSGMISSARG	Message has invalid capability.
L4_EIPC_LO	Communication error-range low.
L4_EIPC_HI	Communication error-range high.

Definition at line 30 of file [err.h](#).

14.1.9 Object Invocation

API for [L4](#) object invocation.

Collaboration diagram for Object Invocation:



Topics

- Message Items ??
Message-item-related functionality.
- Timeouts ??
All kinds of timeouts and time related functions.
- Error Handling ??
Error handling for [L4](#) object invocation.
- Realtime API ??
- Message Tag ??
API related to the message tag data type.
- Virtual Registers (UTCBS) ??
[L4](#) Virtual Registers (UTCB).

Files

- file [utcb.h](#)
UTCB definitions.

Enumerations

- enum `l4_syscall_flags_t` {
`L4_SYSF_NONE`, `L4_SYSF_SEND`, `L4_SYSF_RECV`, `L4_SYSF_OPEN_WAIT`,
`L4_SYSF_REPLY`, `L4_SYSF_CALL`, `L4_SYSF_WAIT`, `L4_SYSF_SEND_AND_WAIT`,
`L4_SYSF_REPLY_AND_WAIT` }

Capability selector flags.

Functions

- `l4_msgtag_t l4_ipc_send (l4_cap_idx_t dest, l4_utcb_t *utcb, l4_msgtag_t tag, l4_timeout_t timeout) L4_NOTHROW`
*Send a message to an object (do **not** wait for a reply).*
- `l4_msgtag_t l4_ipc_wait (l4_utcb_t *utcb, l4_umword_t *label, l4_timeout_t timeout) L4_NOTHROW`
Wait for an incoming message from any possible sender.
- `l4_msgtag_t l4_ipc_receive (l4_cap_idx_t object, l4_utcb_t *utcb, l4_timeout_t timeout) L4_NOTHROW`
Wait for a message from a specific source.
- `l4_msgtag_t l4_ipc_call (l4_cap_idx_t object, l4_utcb_t *utcb, l4_msgtag_t tag, l4_timeout_t timeout) L4_NOTHROW`
Object call (usual invocation).
- `l4_msgtag_t l4_ipc_reply_and_wait (l4_utcb_t *utcb, l4_msgtag_t tag, l4_umword_t *label, l4_timeout_t timeout) L4_NOTHROW`
Reply and wait operation (uses the reply capability).
- `l4_msgtag_t l4_ipc_send_and_wait (l4_cap_idx_t dest, l4_utcb_t *utcb, l4_msgtag_t tag, l4_umword_t *label, l4_timeout_t timeout) L4_NOTHROW`
Send a message and do an open wait.
- `l4_msgtag_t l4_ipc (l4_cap_idx_t dest, l4_utcb_t *utcb, l4_umword_t flags, l4_umword_t label, l4_msgtag_t tag, l4_umword_t *rlabel, l4_timeout_t timeout) L4_NOTHROW`
Generic L4 object invocation.
- `l4_msgtag_t l4_ipc_sleep (l4_timeout_t timeout) L4_NOTHROW`
Sleep for an amount of time.
- `l4_msgtag_t l4_ipc_sleep_ms (l4_uint32_t ms) L4_NOTHROW`
Sleep for a certain amount of milliseconds.
- `l4_msgtag_t l4_ipc_sleep_us (l4_uint64_t us) L4_NOTHROW`
Sleep for a certain amount of microseconds.
- int `l4_sndfpage_add (l4_fpage_t const snd_fpage, unsigned long snd_base, l4_msgtag_t *tag) L4_NOTHROW`
Add a flexpage to be sent to the UTCB.

14.1.9.1 Detailed Description

API for L4 object invocation.

Include File

```
#include <l4/sys/ipc.h>
```

General abstractions for L4 object invocation. The basic principle is that all objects are denoted by a capability that is accessed via a capability selector (see [Capabilities](#)).

This set of functions is common to all kinds of objects provided by the L4 micro kernel. The concrete semantics of an invocation depends on the object that shall be invoked.

Objects may be invoked in various ways, the most common way is to use a *call* operation (`l4_ipc_call()`). However, there are a lot more flavours available that have a semantics depending on the object.

See also

[IPC-Gate API](#)

[L4 Inter-Process Communication \(IPC\)](#)

14.1.9.2 Timeouts during IPC

IPC operation between two communication partners may consist of up to two phases (send phase and receive phase). For both phases, a timeout may be specified (send timeout and receive timeout).

Note

When IPC communication happens across CPU cores and a timeout is specified, then the counting of the timeout only begins after the target thread has been scheduled at least once. In particular, this means that an IPC timeout, including a timeout of zero, may be delayed depending on the scheduling on the target CPU core. If a higher priority thread on the target core is executing a busy loop, that delay may even be indefinitely.

See also

[Timeouts](#)

14.1.9.3 Enumeration Type Documentation

14.1.9.3.1 `l4_syscall_flags_t`

```
enum l4_syscall_flags_t
```

Capability selector flags.

These flags determine the concrete operation when a kernel object is invoked.

The following combinations of flags are supported when invoking IPC (see [l4_ipc\(\)](#)); with other combinations, behavior is undefined:

- [L4_SYSF_SEND](#): send to specified partner
- [L4_SYSF_RECV](#): receive from specified partner
- [L4_SYSF_RECV](#) | [L4_SYSF_OPEN_WAIT](#): receive from any sending partner; see [L4_SYSF_WAIT](#)
- [L4_SYSF_SEND](#) | [L4_SYSF_RECV](#): call specified partner; see [L4_SYSF_CALL](#)
- [L4_SYSF_SEND](#) | [L4_SYSF_RECV](#) | [L4_SYSF_OPEN_WAIT](#): send to specified partner and receive from any sending partner; see [L4_SYSF_SEND_AND_WAIT](#)
- [L4_SYSF_REPLY](#) | [L4_SYSF_SEND](#): reply to caller
- [L4_SYSF_REPLY](#) | [L4_SYSF_SEND](#) | [L4_SYSF_RECV](#): call the caller
- [L4_SYSF_REPLY](#) | [L4_SYSF_SEND](#) | [L4_SYSF_RECV](#) | [L4_SYSF_OPEN_WAIT](#): reply to caller and receive from any sending partner; see [L4_SYSF_REPLY_AND_WAIT](#)

Enumerator

L4_SYSF_NONE	Empty set of flags.
L4_SYSF_SEND	Send-phase flag. Setting this flag in a capability selector induces a send phase, this means a message is sent to the object denoted by the capability. For receive phase see L4_SYSF_RECV . In l4_vcpu_state_t::user_task this flag means that the kernel has cached the user task capability internally, see l4_thread_vcpu_resume_commit() .
L4_SYSF_RECV	Receive-phase flag. Setting this flag in a capability selector induces a receive phase, this means the invoking thread waits for a message from the object denoted by the capability. For a send phase see L4_SYSF_SEND .
L4_SYSF_OPEN_WAIT	Open-wait flag. This flag indicates that the receive operation (see L4_SYSF_RECV) shall be an <i>open wait</i> . <i>Open wait</i> means that the invoking thread shall wait for a message from any possible sender and <i>not</i> from the sender denoted by the capability.
L4_SYSF_REPLY	Reply flag. This flag indicates that the send phase shall use the in-kernel reply capability instead of the capability denoted by the selector index.
L4_SYSF_CALL	Call flags (combines send and receive). Combines L4_SYSF_SEND and L4_SYSF_RECV .
L4_SYSF_WAIT	Wait flags (combines receive and open wait). Combines L4_SYSF_RECV and L4_SYSF_OPEN_WAIT .
L4_SYSF_SEND_AND_WAIT	Send-and-wait flags. Combines L4_SYSF_SEND and L4_SYSF_WAIT .
L4_SYSF_REPLY_AND_WAIT	Reply-and-wait flags. Combines L4_SYSF_SEND , L4_SYSF_REPLY , and L4_SYSF_WAIT .

Definition at line 50 of file [consts.h](#).

14.1.9.4 Function Documentation

14.1.9.4.1 l4_ipc()

```
l4_msgtag_t l4_ipc (
    l4_cap_idx_t dest,
    l4_utcb_t * utcb,
    l4_umword_t flags,
    l4_umword_t slabel,
    l4_msgtag_t tag,
    l4_umword_t * rlabel,
    l4_timeout_t timeout) [inline]
```

Generic [L4](#) object invocation.

Parameters

<i>dest</i>	Destination object. L4_INVALID_CAP denotes the current thread. An IPC to the current thread will always abort after the specified timeout and can be used for sleeping without busy waiting.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .
<i>flags</i>	Invocation flags (see l4_syscall_flags_t).
<i>slabel</i>	Send label if applicable (may be seen by the receiver).
<i>tag</i>	Sending message tag.

out	<i>rlabel</i>	Receiving label.
	<i>timeout</i>	Timeout pair (see l4_timeout_t).

Returns

return tag

See also

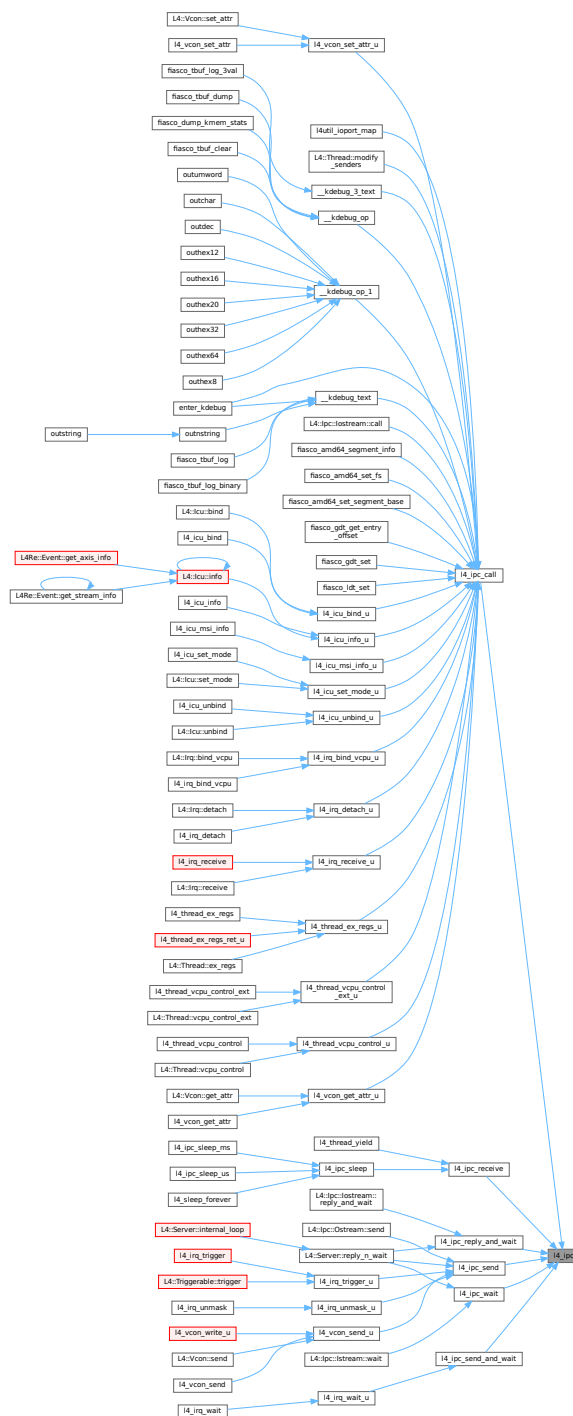
[L4 Inter-Process Communication \(IPC\)](#)

Definition at line 23 of file [ipc.h](#).

References [L4_NOTHROW](#).

Referenced by [l4_ipc_call\(\)](#), [l4_ipc_receive\(\)](#), [l4_ipc_reply_and_wait\(\)](#), [l4_ipc_send\(\)](#), [l4_ipc_send_and_wait\(\)](#), and [l4_ipc_wait\(\)](#).

```
l4_msgtag_t l4_ipc_call (
    l4_cap_idx_t object,
    l4_utcb_t * utcb,
```



14.1.9.4.2 l4_ipc_call()

```
l4_msgtag_t l4_ipc_call (
    l4_cap_idx_t object,
    l4_utcb_t * utcb,
```

```
l4_msgtag_t tag,
l4_timeout_t timeout) [inline]
```

Object call (usual invocation).

Parameters

<i>object</i>	Capability selector for the object to call. A value of L4_INVALID_CAP denotes the current thread and will abort the IPC after the time specified in the <code>snd</code> part of the <code>timeout</code> parameter has expired.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .
<i>tag</i>	Message tag to describe the message to be sent.
<i>timeout</i>	Timeout pair for send an receive phase (see l4_timeout_t).

Returns

result tag

A message is sent to the object and the invoker waits for a reply from the object. Messages from other sources are not accepted.

Note

The send-to-receive transition needs no time, the object can reply with a send timeout of zero.

If a finite receive timeout is specified, the IPC receive operation could abort before the partner was able to send the reply message. Under certain circumstances the partner may still have the temporary reply capability to the calling thread and may use this capability to reply to the caller at a later, unexpected time specifying an arbitrary IPC label. This case is relevant for servers which call another, possibly untrusted, server while serving a client request.

See also

[L4 Inter-Process Communication \(IPC\)](#)

Examples

[examples/sys/aliens/main.c](#), [examples/sys/ipc/ipc_example.c](#), and [examples/sys/singlestep/main.c](#).

Definition at line 565 of file [ipc.h](#).

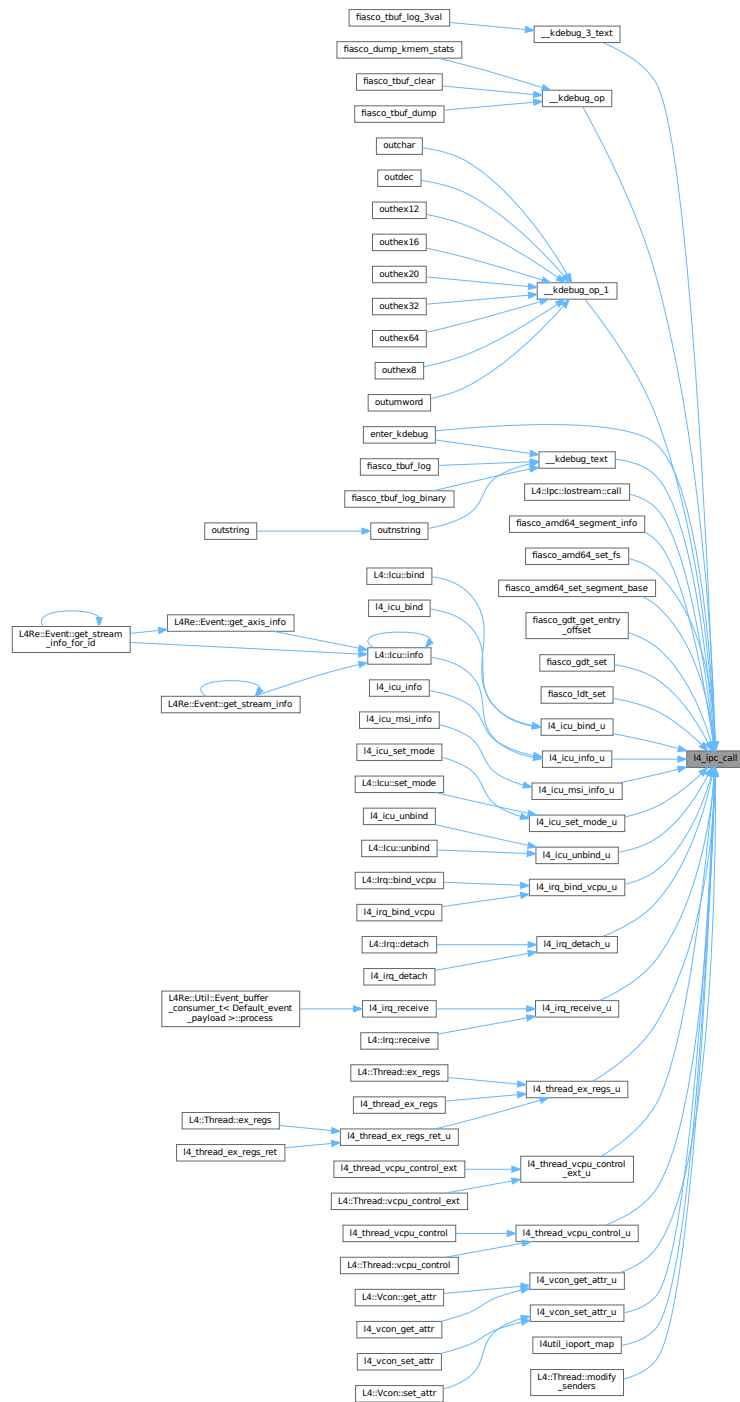
References [l4_ipc\(\)](#), [L4_NOTHROW](#), and [L4_SYSF_CALL](#).

Referenced by [__kdebug_3_text\(\)](#), [__kdebug_op\(\)](#), [__kdebug_op_1\(\)](#), [__kdebug_text\(\)](#), [L4::lpc::loststream::call\(\)](#), [enter_kdebug\(\)](#), [fiasco_amd64_segment_info\(\)](#), [fiasco_amd64_set_fs\(\)](#), [fiasco_amd64_set_segment_base\(\)](#), [fiasco_gdt_get_entry_offset\(\)](#), [fiasco_gdt_set\(\)](#), [fiasco_ldt_set\(\)](#), [l4_icu_bind_u\(\)](#), [l4_icu_info_u\(\)](#), [l4_icu_msi_info_u\(\)](#), [l4_icu_set_mode_u\(\)](#), [l4_icu_unbind_u\(\)](#), [l4_irq_bind_vcpu_u\(\)](#), [l4_irq_detach_u\(\)](#), [l4_irq_receive_u\(\)](#), [l4_thread_ex_regs_u\(\)](#), [l4_thread_vcpu_control_ext_u\(\)](#), [l4_thread_vcpu_control_u\(\)](#), [l4_vcon_get_attr_u\(\)](#), [l4_vcon_set_attr_u\(\)](#), [l4util_ioport_map\(\)](#), and [L4::Thread::modify_senders\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.9.4.3 l4_ipc_receive()

```

l4_msgtag_t l4_ipc_receive (
    l4_cap_idx_t object,
    l4_utcb_t * utcb,
    l4_timeout_t timeout) [inline]

```

Wait for a message from a specific source.

Parameters

<i>object</i>	Object to receive a message from. A value of L4_INVALID_CAP denotes the current thread. It could be used for sleeping without busy waiting for the time specified in the <code>rcv</code> part of the <code>timeout</code> parameter.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .
<i>timeout</i>	Timeout pair (see l4_timeout_t , only the receive part matters).

Returns

result tag.

This operation waits for a message from the specified object. Messages from other sources are not accepted by this operation. The operation does not include a send phase, this means no message is sent to the object.

Note

This operation is usually used to receive messages from a specific IRQ or thread. However, it is not common to use this operation for normal applications.

See also

[L4 Inter-Process Communication \(IPC\)](#)

Examples

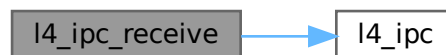
[examples/sys/aliens/main.c](#), [examples/sys/singlestep/main.c](#), [examples/sys/start-with-exc/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

Definition at line 602 of file `ipc.h`.

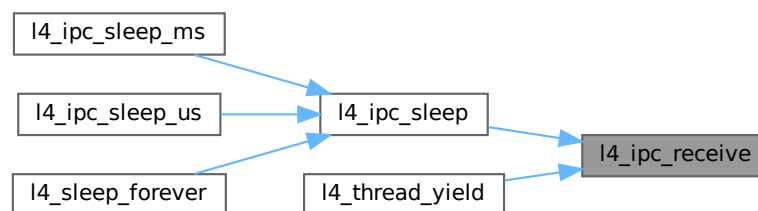
References [l4_ipc\(\)](#), [L4_NOTHROW](#), [L4_SYSF_RECV](#), and [l4_msgtag_t::raw](#).

Referenced by [l4_ipc_sleep\(\)](#), and [l4_thread_yield\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.9.4.4 l4_ipc_reply_and_wait()

```
l4_msgtag_t l4_ipc_reply_and_wait (
    l4_utcb_t * utcb,
    l4_msgtag_t tag,
    l4_umword_t * label,
    l4_timeout_t timeout) [inline]
```

Reply and wait operation (uses the *reply* capability).

Parameters

	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .
	<i>tag</i>	Describes the message to be sent as reply.
out	<i>label</i>	Label assigned to the source object of the received message.
	<i>timeout</i>	Timeout pair (see l4_timeout_t).

Returns

result tag

A message is sent to the previous caller using the implicit reply capability. Afterwards the invoking thread waits for a message from any source.

Note

This is the standard server operation: it sends a reply to the actual client and waits for the next incoming request, which may come from any other client.

In case of multiple senders trying to send to the thread performing this system call, the thread receives from a sender with the highest priority. In this respect, IRQ sources have the highest priority 255.

See also

[L4 Inter-Process Communication \(IPC\)](#)

Examples

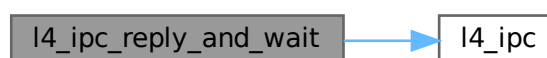
[examples/sys/ipc/ipc_example.c](#).

Definition at line 572 of file [ipc.h](#).

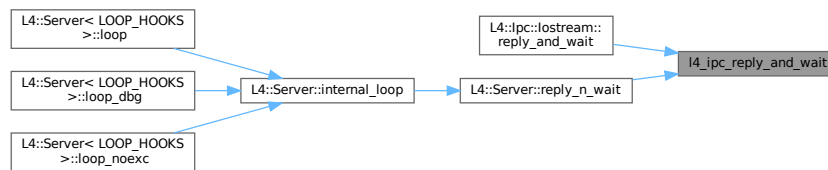
References [L4_INVALID_CAP](#), [l4_ipc\(\)](#), [L4_NOTHROW](#), and [L4_SYSF_REPLY_AND_WAIT](#).

Referenced by [L4::ipc::loststream::reply_and_wait\(\)](#), and [L4::Server< LOOP_HOOKS >::reply_n_wait\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.9.4.5 l4_ipc_send()

```

l4_msgtag_t l4_ipc_send (
    l4_cap_idx_t dest,
    l4_utcb_t * utcb,
    l4_msgtag_t tag,
    l4_timeout_t timeout) [inline]

```

Send a message to an object (do **not** wait for a reply).

Parameters

<i>dest</i>	Capability selector for the destination object. A value of L4_INVALID_CAP denotes the current thread and could be used for sleeping without busy waiting for the time specified in the <code>snd</code> part of the <code>timeout</code> parameter.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .
<i>tag</i>	Descriptor for the message to be sent.
<i>timeout</i>	Timeout pair (see l4_timeout_t) only send part is relevant.

Returns

Syscall return tag for the send-only operation, this means there is no return value except [L4_MSGTAG_ERROR](#) indicating success or failure of the send operation. Use [l4_ipc_error\(\)](#) to check for errors and **do not** use [l4_error\(\)](#).

A message is sent to the destination object. There is no receive phase included. The invoker continues working after sending the message.

Note

This is a special-purpose message transfer. Objects usually support only invocation via [l4_ipc_call\(\)](#) consisting of a send phase and a receive phase for returning the result of the object invocation. For example, [l4_icu_unmask\(\)](#), [l4_icu_mask\(\)](#) and [l4_irq_trigger\(\)](#) use send-only IPC operations for object invocation.

	<i>dest</i>	Object to send a message to. A value of L4_INVALID_CAP denotes the current thread and will abort the IPC after the time specified in the <code>snd</code> part of the <code>timeout</code> parameter has expired.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .
	<i>tag</i>	Describes the message that shall be sent.
out	<i>label</i>	Label assigned to the source object of the receive phase.
	<i>timeout</i>	Timeout pair (see l4_timeout_t).

Returns

result tag

A message is sent to the destination object and the invoking thread waits for a reply from any source.

Note

This is a special-purpose operation and shall not be used in general applications.

In case of multiple senders trying to send to the thread performing this system call, the thread receives from a sender with the highest priority. In this respect, IRQ sources have the highest priority 255.

See also

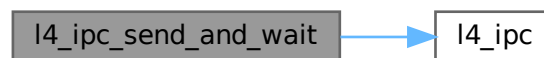
[L4 Inter-Process Communication \(IPC\)](#)

Definition at line 579 of file [ipc.h](#).

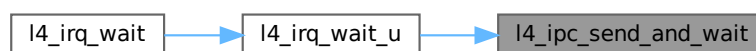
References [l4_ipc\(\)](#), [L4_NOTHROW](#), and [L4_SYSF_SEND_AND_WAIT](#).

Referenced by [l4_irq_wait_u\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.9.4.7 l4_ipc_sleep()

```
l4_msgtag_t l4_ipc_sleep (  
    l4_timeout_t timeout)  [inline]
```

Sleep for an amount of time.

Parameters

<i>timeout</i>	Timeout pair (see l4_timeout_t , the receive part matters).
----------------	---

Returns

error code:

- [L4_IPC_RETIMEOUT](#): success
- [L4_IPC_RECANCELED](#) woken up by a different thread ([l4_thread_ex_regs\(\)](#)).

The invoking thread waits until the timeout is expired or the wait was aborted by another thread by [l4_thread_ex_regs\(\)](#).

See also

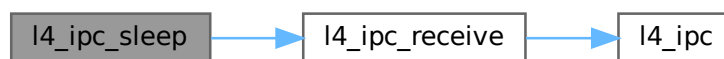
[L4 Inter-Process Communication \(IPC\)](#)

Definition at line 611 of file [ipc.h](#).

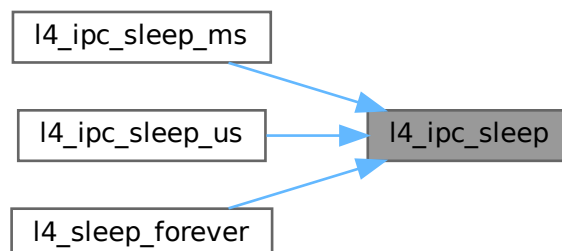
References [L4_INVALID_CAP](#), [l4_ipc_receive\(\)](#), and [L4_NOTHROW](#).

Referenced by [l4_ipc_sleep_ms\(\)](#), [l4_ipc_sleep_us\(\)](#), and [l4_sleep_forever\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.9.4.8 l4_ipc_sleep_ms()

```
l4_msgtag_t l4_ipc_sleep_ms (  
    l4_uint32_t ms) [inline]
```

Sleep for a certain amount of milliseconds.

Parameters

<i>ms</i>	Number of milliseconds to wait.
-----------	---------------------------------

Returns

error code:

- [L4_IPC_RETIMEOUT](#): success
- [L4_IPC_RECANCELED](#) woken up by a different thread ([l4_thread_ex_regs\(\)](#)).

The invoking thread waits until the timeout is expired or the wait was aborted by another thread by [l4_thread_ex_regs\(\)](#).

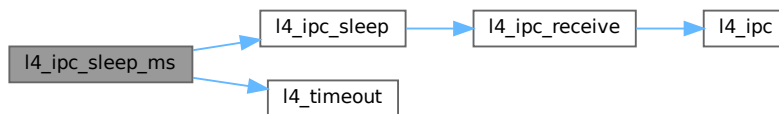
See also

[L4 Inter-Process Communication \(IPC\)](#)

Definition at line 615 of file [ipc.h](#).

References [l4_ipc_sleep\(\)](#), [L4_IPC_TIMEOUT_NEVER](#), [L4_NOTHROW](#), and [l4_timeout\(\)](#).

Here is the call graph for this function:



14.1.9.4.9 l4_ipc_sleep_us()

```
l4_msgtag_t l4_ipc_sleep_us (
    l4_uint64_t us) [inline]
```

Sleep for a certain amount of microseconds.

Parameters

<i>us</i>	Number of microseconds to wait.
-----------	---------------------------------

Returns

error code:

- [L4_IPC_RETIMEOUT](#): success
- [L4_IPC_RECANCELED](#) woken up by a different thread ([l4_thread_ex_regs\(\)](#)).

The invoking thread waits until the timeout is expired or the wait was aborted by another thread by [l4_thread_ex_regs\(\)](#).

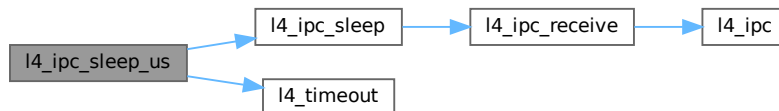
See also

[L4 Inter-Process Communication \(IPC\)](#)

Definition at line 622 of file [ipc.h](#).

References [l4_ipc_sleep\(\)](#), [L4_IPC_TIMEOUT_NEVER](#), [L4_NOTHROW](#), and [l4_timeout\(\)](#).

Here is the call graph for this function:



14.1.9.4.10 l4_ipc_wait()

```

l4_msgtag_t l4_ipc_wait (
    l4_utcb_t * utcb,
    l4_umword_t * label,
    l4_timeout_t timeout) [inline]
  
```

Wait for an incoming message from any possible sender.

Parameters

	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .
out	<i>label</i>	Label assigned to the source object (IPC gate or IRQ).
	<i>timeout</i>	Timeout pair (see l4_timeout_t , only the receive part is used).

Returns

return tag

This operation does an open wait, and therefore needs no capability to denote the possible source of a message. This means the calling thread waits for an incoming message from any possible source. There is no send phase included in this operation.

The usual usage of this function is to call that function when entering a server loop in a user-level server that implements user-level objects, see also [l4_ipc_reply_and_wait\(\)](#).

Note

In case of multiple senders trying to send to the thread performing this system call, the thread receives from a sender with the highest priority. In this respect, IRQ sources have the highest priority 255.

See also

[L4 Inter-Process Communication \(IPC\)](#)

Examples

[examples/sys/ipc/ipc_example.c](#).

Definition at line 593 of file [ipc.h](#).

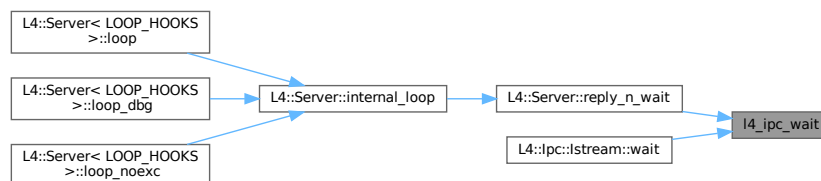
References [L4_INVALID_CAP](#), [l4_ipc\(\)](#), [L4_NOTHROW](#), [L4_SYSF_WAIT](#), and [l4_msgtag_t::raw](#).

Referenced by [L4::Server< LOOP_HOOKS >::reply_n_wait\(\)](#), and [L4::ipc::lstream::wait\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.9.4.11 l4_sndfpage_add()

```

int l4_sndfpage_add (
    l4_fpage_t const snd_fpage,
    unsigned long snd_base,
    l4_msgtag_t * tag) [inline]
  
```

Add a flexpage to be sent to the UTCB.

Parameters

<i>snd_fpage</i>	Flexpage.
<i>snd_base</i>	Send base.

<code>in, out</code>	<code>tag</code>	Tag to be updated. Only the number of items is incremented in the updated tag, all other members remain unmodified.
----------------------	------------------	---

Returns

0 on success, negative error code otherwise

See also

[L4 Inter-Process Communication \(IPC\)](#)

Definition at line 685 of file [ipc.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

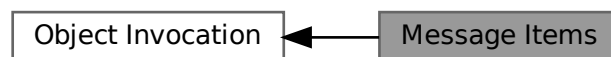
Here is the call graph for this function:



14.1.9.5 Message Items

Message-item-related functionality.

Collaboration diagram for Message Items:



Data Structures

- struct [l4_snd_fpage_t](#)
Send-flexpage types.

Enumerations

- enum `L4_obj_fpage_ctl` {
`L4_FPAGE_C_REF_CNT` = 0x00 , `L4_FPAGE_C_NO_REF_CNT` = 0x10 , `L4_FPAGE_C_OBJ_RIGHT1` = 0x20 , `L4_FPAGE_C_OBJ_RIGHT2` = 0x40 ,
`L4_FPAGE_C_OBJ_RIGHT3` = 0x80 , `L4_FPAGE_C_OBJ_RIGHTS` = 0xe0 , `L4_FPAGE_C_IPCGATE_SVR` = `L4_FPAGE_C_OBJ_RIGHT1` }

Attributes and additional permissions for object send items.

- enum `l4_fpage_cacheability_opt_t` { `L4_FPAGE_CACHE_OPT` = 0x1 , `L4_FPAGE_CACHEABLE` = 0x3 ,
`L4_FPAGE_BUFFERABLE` = 0x5 , `L4_FPAGE_UNCACHEABLE` = 0x1 }

Cacheability options for memory send items.

- enum `l4_msg_item_consts_t` {
`L4_ITEM_MAP` = 8 , `L4_ITEM_CONT` = 1 , `L4_MAP_ITEM_GRANT` = 2 , `L4_MAP_ITEM_MAP` = 0 ,
`L4_RCV_ITEM_FORWARD_MAPPINGS` = 1 , `L4_RCV_ITEM_SINGLE_CAP` = `L4_ITEM_MAP` | 2 ,
`L4_RCV_ITEM_LOCAL_ID` = 4 }

Constants for message items.

Functions

- `l4_umword_t l4_map_control` (`l4_umword_t` spot, unsigned char cache, unsigned grant) `L4_NOTHROW`

Create the first word for a map item that is a send item for the memory space.

- `l4_umword_t l4_map_obj_control` (`l4_umword_t` spot, unsigned grant) `L4_NOTHROW`

Create the first word for a map item that is a send item for the object space.

14.1.9.5.1 Detailed Description

Message-item-related functionality.

Message items are typed items that are used for transferring capabilities during IPC. There are three sub-types of typed message items with variations in the layout:

1. Typed message items set by the sender in its message registers (MRs) of the UTCB for specifying what shall be sent.
2. Typed message items set by the receiver in its buffer registers (BRs) of the UTCB for specifying which types of capabilities may be received at which addresses.
3. Typed message items set by the kernel in the receiver's message registers (MRs) of the UTCB for providing information about the transfer to the receiver.

They are abbreviated by *send item*, *receive item*, and *return item*, respectively.

A typed message item in the message registers (case 1 and case 3) always consists of two words (even if it is a void item). The size of a typed message item in the buffer registers (case 2) is determined by its first word. The size is up to three words (see `L4_RCV_ITEM_SINGLE_CAP` and `L4_RCV_ITEM_FORWARD_MAPPINGS`). A void item in the buffer registers consists of a single word.

Include File

```
#include <l4/sys/types.h>
```

14.1.9.5.2 Enumeration Type Documentation

14.1.9.5.2.1 l4_fpage_cacheability_opt_t

enum `l4_fpage_cacheability_opt_t`

Cacheability options for memory send items.

Only the IPC sender and the thread performing the map operation can specify the caching mode of the target mapping. By default, the caching mode of the sender is used as caching mode for the target mapping. If `L4_FPAGE_CACHE_OPT` is set in the send item, the caching mode is overridden by the respective mode from below.

Enumerator

L4_FPAGE_CACHE_OPT	Enable the cacheability option in a memory send item. Without this flag, the options are copied from the sender.
L4_FPAGE_CACHEABLE	Cacheability option to enable caches for the mapping. Implies L4_FPAGE_CACHE_OPT .
L4_FPAGE_BUFFERABLE	Cacheability option to enable buffered writes for the mapping. Implies L4_FPAGE_CACHE_OPT .
L4_FPAGE_UNCACHEABLE	Cacheability option to disable caching for the mapping. Implies L4_FPAGE_CACHE_OPT .

Definition at line 291 of file [__l4_fpage.h](#).

14.1.9.5.2.2 l4_msg_item_consts_t

enum [l4_msg_item_consts_t](#)

Constants for message items.

Enumerator

L4_ITEM_MAP	Identify a message item as <i>map item</i> .
L4_ITEM_CONT	Denote that the following item shall be put into the same receive item as this one.
L4_MAP_ITEM_GRANT	<p>Flag as <i>grant</i> instead of <i>map</i> operation. This means, the sender delegates access to the receiver and the kernel removes the rights from the sender (basically a move operation). The mapping in the receiver gets the new parent of any child mappings of the mapping of the sender. Rights revocation via send item/flexpage is <i>not</i> guaranteed to be applied to descendant mappings in case of grant. See Spaces and Mappings for more details on map/grant.</p> <p>Note</p> <p>The grant operation is not performed if the resulting rights of the receiver mapping would not contain the L4_CAP_FPAGE_R bit (for object capabilities) or none of the L4_FPAGE_RWX bits (memory and IO ports). In that case, the mapping is not created in the receiver space and not removed from the sender space.</p> <p>If the removal of the whole mapping from the sender is not possible because the size of the mapped frame at the sender exceeds the size defined by the send or receive flexpage, the grant operation is turned into a regular map operation and the mapping is <i>not</i> removed from the sender. This would happen if, for example, a smaller part of an L4 superpage mapping shall be granted.</p>
L4_MAP_ITEM_MAP	Flag as usual <i>map</i> operation.

L4_RCV_ITEM_FORWARD_MAPPINGS	<p>This flag specifies if received capabilities shall be mapped to a particular task instead of the invoking task. This flag may be used only if L4_RCV_ITEM_LOCAL_ID is unset.</p> <p>Setting this flag increases the size of the buffer item by one word. This word is used to specify a capability index for the task that shall receive the mappings.</p>
L4_RCV_ITEM_SINGLE_CAP	<p>Mark the receive buffer to be a small receive item that describes a buffer for a single object capability. A receive item needs to specify a <i>receive window</i>. The receive window determines which kind of capabilities (object, memory, I/O ports) may be received where in the respective space. If this flag is unset, the receive window is specified in the second word of the receive item via a flexpage. If this flag is set, the receive window consists of a single capability index in the object space and the capability index is specified in the most significant bits of the first word of the receive item (see L4_CAP_SHIFT).</p>
L4_RCV_ITEM_LOCAL_ID	<p>The receiver requests to receive a local ID instead of a mapping whenever possible. This flag may be used only if L4_RCV_ITEM_SINGLE_CAP is set and L4_RCV_ITEM_FORWARD_MAPPINGS is unset.</p> <p>When this flag is set, then,</p> <ul style="list-style-type: none"> • when sender and receiver are bound to the same task, then no mapping is done for this item and just the raw flexpage (l4_fpage_t) is transferred, • otherwise, when the sender specified an IPC gate for transfer that is bound to a thread that is bound to the same task as the receiving thread, then no mapping is done for this item and just the bitwise OR () of the label and the L4_CAP_FPAGE_W and L4_CAP_FPAGE_S permissions that would have been mapped is transferred, • otherwise a regular mapping is done for this item.

Definition at line 212 of file [consts.h](#).

14.1.9.5.2.3 L4_obj_fpage_ctl

```
enum L4_obj_fpage_ctl
```

Attributes and additional permissions for object send items.

These rights need to be added to the `snd_base` when mapping and control internal behavior. The exact meaning depends on the type of capability (currently used only with IPC gates).

Enumerator

L4_FPAGE_C_REF_CNT	Mapping is reference-counted (default).
L4_FPAGE_C_NO_REF_CNT	Don't increase the reference counter.
L4_FPAGE_C_OBJ_RIGHT1	Object-type specific right.

L4_FPAGE_C_OBJ_RIGHT2	Object-type specific right.
L4_FPAGE_C_OBJ_RIGHT3	Object-type specific right.
L4_FPAGE_C_OBJ_RIGHTS	All Object-type specific right bits.
L4_FPAGE_C_IPCGATE_SVR	The receiver may invoke IPC-gate-specific functions on the capability, e.g. bind a thread to the gate and modify the label. Needed if the receiver implements the server side of an IPC gate.

Definition at line 262 of file [__l4_fpage.h](#).

14.1.9.5.3 Function Documentation

14.1.9.5.3.1 l4_map_control()

```
l4_umword_t l4_map_control (
    l4_umword_t spot,
    unsigned char cache,
    unsigned grant) [inline]
```

Create the first word for a map item that is a send item for the memory space.

Parameters

<i>spot</i>	Hot spot address, used to determine what is actually mapped when send and receive flexpage have differing sizes.
<i>cache</i>	Cacheability hints for memory flexpages. See Cacheability options .
<i>grant</i>	Indicates if it is a map or a grant item. Allowed values: L4_MAP_ITEM_MAP , L4_MAP_ITEM_GRANT .

Returns

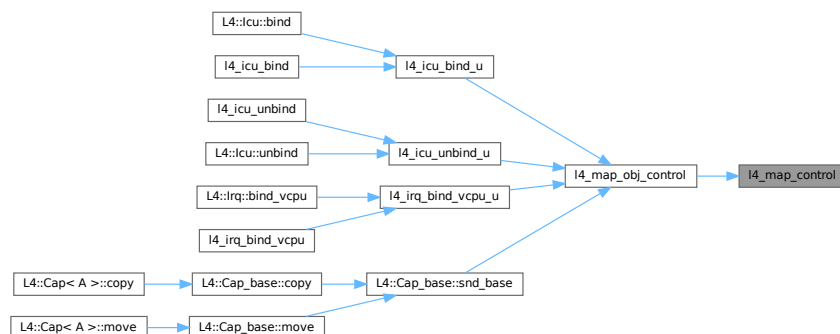
The value to be used as first word in a send item for memory.

Definition at line 742 of file [__l4_fpage.h](#).

References [L4_FPAGE_CONTROL_MASK](#), [L4_ITEM_MAP](#), and [L4_NOTHROW](#).

Referenced by [l4_map_obj_control\(\)](#).

Here is the caller graph for this function:



14.1.9.5.3.2 l4_map_obj_control()

```
l4_umword_t l4_map_obj_control (
    l4_umword_t spot,
    unsigned grant) [inline]
```

Create the first word for a map item that is a send item for the object space.

Parameters

<i>spot</i>	Hot spot address, used to determine what is actually mapped when send and receive flexpages have different size.
<i>grant</i>	Indicates if it is a map item or a grant item. Allowed values: L4_MAP_ITEM_MAP , L4_MAP_ITEM_GRANT .

Returns

The value to be used as first word in a send item for kernel objects or IO-ports.

Definition at line 749 of file [__l4_fpage.h](#).

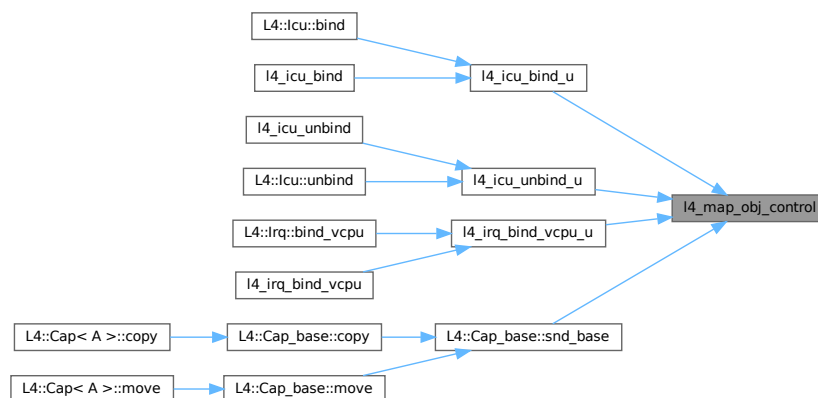
References [l4_map_control\(\)](#), and [L4_NOTHROW](#).

Referenced by [l4_icu_bind_u\(\)](#), [l4_icu_unbind_u\(\)](#), [l4_irq_bind_vcpu_u\(\)](#), and [L4::Cap_base::snd_base\(\)](#).

Here is the call graph for this function:



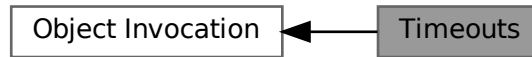
Here is the caller graph for this function:



14.1.9.6 Timeouts

All kinds of timeouts and time related functions.

Collaboration diagram for Timeouts:



Data Structures

- struct `l4_timeout_s`
Basic timeout specification.
- union `l4_timeout_t`
Timeout pair.

Macros

- `#define L4_IPC_TIMEOUT_0 ((l4_timeout_s){0x0400})`
Timeout constants.
- `#define L4_IPC_TIMEOUT_NEVER ((l4_timeout_s){0})`
never timeout
- `#define L4_IPC_NEVER_INITIALIZER {0}`
never timeout, initializer
- `#define L4_IPC_NEVER ((l4_timeout_t){0})`
never timeout
- `#define L4_IPC_RECV_TIMEOUT_0 ((l4_timeout_t){0x00000400})`
0 receive timeout
- `#define L4_IPC_SEND_TIMEOUT_0 ((l4_timeout_t){0x04000000})`
0 send timeout
- `#define L4_IPC_BOTH_TIMEOUT_0 ((l4_timeout_t){0x04000400})`
0 receive and send timeout
- `#define L4_TIMEOUT_US_NEVER (~0ULL)`
The waiting period in microseconds which is interpreted as "never" by l4_timeout_from_us().
- `#define L4_TIMEOUT_US_MAX ((1ULL << 41) - 1)`
The longest waiting period in microseconds accepted by l4_timeout_from_us().

Typedefs

- `typedef struct l4_timeout_s l4_timeout_s`
Basic timeout specification.
- `typedef union l4_timeout_t l4_timeout_t`
Timeout pair.

Functions

- [L4_CONSTEXPR l4_timeout_s l4_timeout_rel](#) (unsigned man, unsigned exp) [L4_NOTHROW](#)
Get relative timeout consisting of mantissa and exponent.
- [L4_CONSTEXPR l4_timeout_t l4_ipc_timeout](#) (unsigned snd_man, unsigned snd_exp, unsigned rcv_man, unsigned rcv_exp) [L4_NOTHROW](#)
Convert explicit timeout values to [l4_timeout_t](#) type.
- [L4_CONSTEXPR l4_timeout_t l4_timeout](#) (l4_timeout_s snd, l4_timeout_s rcv) [L4_NOTHROW](#)
Combine send and receive timeout in a timeout.
- [L4_CONSTEXPR void l4_snd_timeout](#) (l4_timeout_s snd, l4_timeout_t *to) [L4_NOTHROW](#)
Set send timeout in given to timeout.
- [L4_CONSTEXPR void l4_rcv_timeout](#) (l4_timeout_s rcv, l4_timeout_t *to) [L4_NOTHROW](#)
Set receive timeout in given to timeout.
- [L4_CONSTEXPR l4_kernel_clock_t l4_timeout_rel_get](#) (l4_timeout_s to) [L4_NOTHROW](#)
Get clock value of out timeout.
- [L4_CONSTEXPR unsigned l4_timeout_is_absolute](#) (l4_timeout_s to) [L4_NOTHROW](#)
Return whether the given timeout is absolute or not.
- [L4_CONSTEXPR l4_kernel_clock_t l4_timeout_get](#) (l4_kernel_clock_t cur, l4_timeout_s to) [L4_NOTHROW](#)
Get clock value for a clock + a timeout.
- [l4_timeout_s l4_timeout_abs](#) (l4_kernel_clock_t pint, int br) [L4_NOTHROW](#)
Set an absolute timeout.
- unsigned [l4_utcb_mr64_idx](#) (unsigned idx) [L4_NOTHROW](#)
Get index into 64bit message registers alias from native-sized index.

14.1.9.6.1 Detailed Description

All kinds of timeouts and time related functions.

14.1.9.6.2 Macro Definition Documentation

14.1.9.6.2.1 L4_IPC_TIMEOUT_0

```
#define L4_IPC_TIMEOUT_0 ((l4_timeout_s){0x0400})
```

Timeout constants.

0 timeout

Definition at line 73 of file [__timeout.h](#).

Referenced by [L4::lpc_svr::Timeout_queue_hooks<HOOKS, BR_MAN>::timeout\(\)](#).

14.1.9.6.2.2 L4_TIMEOUT_US_MAX

```
#define L4_TIMEOUT_US_MAX ((1ULL << 41) - 1)
```

The longest waiting period in microseconds accepted by [l4_timeout_from_us\(\)](#).

See [l4_timeout_from_us\(\)](#) for an explanation.

Definition at line 91 of file [__timeout.h](#).

14.1.9.6.3 Typedef Documentation

14.1.9.6.3.1 l4_timeout_s

```
typedef struct l4_timeout_s l4_timeout_s
```

Basic timeout specification.

If bit 15 == 0, basically a floating point number with 10 bits mantissa and 5 bits exponent ($t = m \cdot 2^e$).

If the mantissa is zero, the exponent encodes special values, see [L4_IPC_TIMEOUT_0](#) and [L4_IPC_TIMEOUT_NEVER](#).

If bit 15 == 1 the timeout is absolute and the lower 6 bits encode the index of the UTCB buffer register(s) holding the absolute 64-bit timeout value. On 32-bit systems, two consecutive UTCB buffer registers are used.

14.1.9.6.3.2 l4_timeout_t

```
typedef union l4_timeout_t l4_timeout_t
```

Timeout pair.

For IPC there are usually a send and a receive timeout. So this structure contains a pair of timeouts.

14.1.9.6.4 Function Documentation

14.1.9.6.4.1 l4_ipc_timeout()

```
L4_CONSTEXPR l4_timeout_t l4_ipc_timeout (
    unsigned snd_man,
    unsigned snd_exp,
    unsigned rcv_man,
    unsigned rcv_exp) [inline]
```

Convert explicit timeout values to [l4_timeout_t](#) type.

Parameters

<i>snd_man</i>	Mantissa of send timeout.
<i>snd_exp</i>	Exponent of send timeout.
<i>rcv_man</i>	Mantissa of receive timeout.
<i>rcv_exp</i>	Exponent of receive timeout.

Definition at line 203 of file [__timeout.h](#).

References [L4_NOTHROW](#), and [l4_timeout\(\)](#).

Here is the call graph for this function:



14.1.9.6.4.2 l4_rcv_timeout()

```
L4_CONSTEXPR void l4_rcv_timeout (
    l4_timeout_s rcv,
    l4_timeout_t * to) [inline]
```

Set receive timeout in given to timeout.

Parameters

	<i>rcv</i>	Receive timeout
out	<i>to</i>	L4 timeout

Definition at line 227 of file [__timeout.h](#).

References [L4_NOTHROW](#).

14.1.9.6.4.3 l4_snd_timeout()

```
L4_CONSTEXPR void l4_snd_timeout (
    l4_timeout_s snd,
    l4_timeout_t * to) [inline]
```

Set send timeout in given to timeout.

Parameters

	<i>snd</i>	Send timeout
out	<i>to</i>	L4 timeout

Definition at line 220 of file [__timeout.h](#).

References [L4_NOTHROW](#).

14.1.9.6.4.4 l4_timeout()

```
L4_CONSTEXPR l4_timeout_t l4_timeout (
    l4_timeout_s snd,
    l4_timeout_s rcv) [inline]
```

Combine send and receive timeout in a timeout.

Parameters

<i>snd</i>	Send timeout
<i>rcv</i>	Receive timeout

Returns

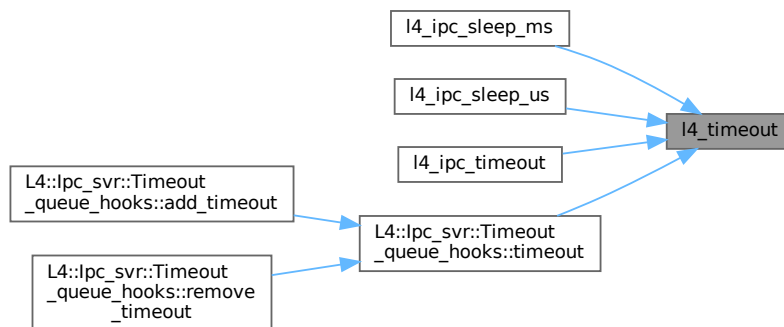
L4 timeout

Definition at line 213 of file `__timeout.h`.

References `L4_NOTHROW`.

Referenced by `l4_ipc_sleep_ms()`, `l4_ipc_sleep_us()`, `l4_ipc_timeout()`, and `L4::lpc_svr::Timeout_queue_hooks<HOOKS, BR_MAN`

Here is the caller graph for this function:



14.1.9.6.4.5 l4_timeout_abs()

```
l4_timeout_s l4_timeout_abs (
    l4_kernel_clock_t pint,
    int br) [inline]
```

Set an absolute timeout.

Parameters

<i>pint</i>	Point in time in clocks
<i>br</i>	The buffer register the timeout shall be placed in. (

Note

On 32bit architectures the timeout needs two consecutive buffers.)

The absolute timeout value will be placed into the buffer register *br* of the current thread.

Returns

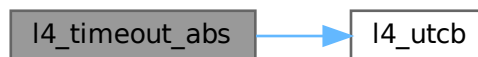
timeout value

Definition at line 389 of file [utcb.h](#).

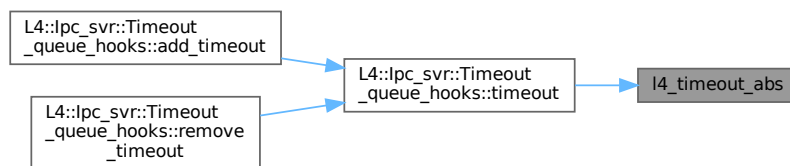
References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Referenced by [L4::lpc_svr::Timeout_queue_hooks<HOOKS, BR_MAN>::timeout\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**14.1.9.6.4.6 l4_timeout_get()**

```

L4_CONSTEXPR l4_kernel_clock_t l4_timeout_get (
    l4_kernel_clock_t cur,
    l4_timeout_s to) [inline]
  
```

Get clock value for a clock + a timeout.

Parameters

<i>cur</i>	Clock value
<i>to</i>	L4 timeout

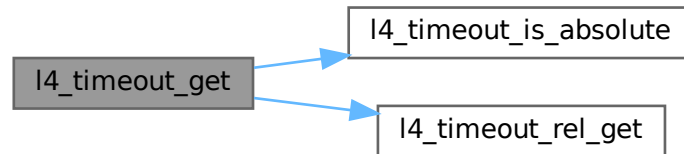
Returns

Clock sum

Definition at line 257 of file [__timeout.h](#).

References [L4_NOTHROW](#), [l4_timeout_is_absolute\(\)](#), and [l4_timeout_rel_get\(\)](#).

Here is the call graph for this function:

**14.1.9.6.4.7 l4_timeout_is_absolute()**

```
L4_CONSTEXPR unsigned l4_timeout_is_absolute (  
    l4_timeout_s to) [inline]
```

Return whether the given timeout is absolute or not.

Parameters

<i>to</i>	L4 timeout
-----------	----------------------------

Returns

!= 0 if absolute, 0 if relative

Definition at line 250 of file [__timeout.h](#).

References [L4_NOTHROW](#).

Referenced by [l4_timeout_get\(\)](#).

Here is the caller graph for this function:



14.1.9.6.4.8 l4_timeout_rel()

```
L4_CONSTEXPR l4_timeout_s l4_timeout_rel (  
    unsigned man,  
    unsigned exp) [inline]
```

Get relative timeout consisting of mantissa and exponent.

Parameters

<i>man</i>	Mantissa of timeout
<i>exp</i>	Exponent of timeout

Returns

timeout value

Definition at line 234 of file [__timeout.h](#).

References [L4_NOTHROW](#).

14.1.9.6.4.9 l4_timeout_rel_get()

```
L4_CONSTEXPR l4_kernel_clock_t l4_timeout_rel_get (  
    l4_timeout_s to) [inline]
```

Get clock value of out timeout.

Parameters

<i>to</i>	L4 timeout
-----------	----------------------------

Returns

Clock value

Definition at line 241 of file [__timeout.h](#).

References [L4_NOTHROW](#).

Referenced by [l4_timeout_get\(\)](#).

Here is the caller graph for this function:



14.1.9.6.4.10 l4_utcb_mr64_idx()

```
unsigned l4_utcb_mr64_idx (
    unsigned idx) [inline]
```

Get index into 64bit message registers alias from native-sized index.

Parameters

<i>idx</i>	Index to native-sized message register
------------	--

Returns

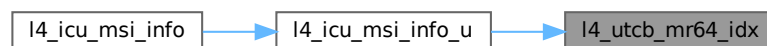
Index to 64bit message register alias

Definition at line 392 of file [utcb.h](#).

References [L4_INLINE](#), and [L4_NOTHROW](#).

Referenced by [l4_icu_msi_info_u\(\)](#).

Here is the caller graph for this function:



14.1.9.7 Error Handling

Error handling for [L4](#) object invocation.

Collaboration diagram for Error Handling:



Enumerations

- enum [l4_ipc_tcr_error_t](#) {
[L4_IPC_ERROR_MASK](#) = 0x1F , [L4_IPC_SND_ERR_MASK](#) = 0x01 , [L4_IPC_ENOT_EXISTENT](#) = 0x04 ,
[L4_IPC_RETIMEOUT](#) = 0x03 ,
[L4_IPC_SETIMEOUT](#) = 0x02 , [L4_IPC_RECANCELED](#) = 0x07 , [L4_IPC_SECANCELED](#) = 0x06 ,
[L4_IPC_REMAPFAILED](#) = 0x11 ,
[L4_IPC_SEMAPFAILED](#) = 0x10 , [L4_IPC_RESNDPFTO](#) = 0x0b , [L4_IPC_SESNDPFTO](#) = 0x0a ,
[L4_IPC_RERCVPFTO](#) = 0x0d ,
[L4_IPC_SERCVPFTO](#) = 0x0c , [L4_IPC_REABORTED](#) = 0x0f , [L4_IPC_SEABORTED](#) = 0x0e ,
[L4_IPC_REMSGCUT](#) = 0x09 ,
[L4_IPC_SEMSGCUT](#) = 0x08 }

Error codes in the error TCR.

Functions

- [l4_umword_t](#) [l4_ipc_error](#) ([l4_msgtag_t](#) tag, [l4_utcb_t](#) *utcb) [L4_NOTHROW](#)
Get the IPC error code for an IPC operation.
- long [l4_error](#) ([l4_msgtag_t](#) tag) [L4_NOTHROW](#)
Get IPC error code if any or message tag label otherwise for an IPC call.
- int [l4_ipc_is_snd_error](#) ([l4_utcb_t](#) *utcb) [L4_NOTHROW](#)
Returns whether an error occurred in send phase of an invocation.
- int [l4_ipc_is_rcv_error](#) ([l4_utcb_t](#) *utcb) [L4_NOTHROW](#)
Returns whether an error occurred in receive phase of an invocation.
- int [l4_ipc_error_code](#) ([l4_utcb_t](#) *utcb) [L4_NOTHROW](#)
Get the error condition of the last invocation from the TCR.

14.1.9.7.1 Detailed Description

Error handling for [L4](#) object invocation.

Include File

```
#include <l4/sys/ipc.h>
```

14.1.9.7.2 Enumeration Type Documentation

14.1.9.7.2.1 [l4_ipc_tcr_error_t](#)

```
enum l4\_ipc\_tcr\_error\_t
```

Error codes in the *error* TCR.

The error codes are accessible via the *error* TCR, see [l4_thread_regs_t.error](#).

Enumerator

L4_IPC_ERROR_MASK	Mask for error bits.
-----------------------------------	----------------------

L4_IPC_SND_ERR_MASK	Send error mask.
L4_IPC_ENOT_EXISTENT	Non-existing destination or source.
L4_IPC_RETIMEOUT	Timeout during receive operation.
L4_IPC_SETIMEOUT	Timeout during send operation.
L4_IPC_RECANCELED	Receive operation canceled.
L4_IPC_SEANCELED	Send operation canceled.
L4_IPC_REMAPFAILED	Map flexpage failed in receive operation.
L4_IPC_SEMAPFAILED	Map flexpage failed in send operation.
L4_IPC_RESNDPFTO	Send-pagefault timeout in receive operation.
L4_IPC_SESNDPFTO	Send-pagefault timeout in send operation.
L4_IPC_RERCVPFTO	Receive-pagefault timeout in receive operation.
L4_IPC_SERCVPFTO	Receive-pagefault timeout in send operation.
L4_IPC_REABORTED	Receive operation aborted.
L4_IPC_SEABORTED	Send operation aborted.
L4_IPC_REMSGCUT	Received message truncated. Usually returned when the typed items to be sent by the IPC partner exceed the buffer registers of the respective types.
L4_IPC_SEMSGCUT	Sent message truncated. Usually returned when the typed items to be sent exceed the IPC partner's buffer registers of the respective types.

Definition at line 81 of file [ipc.h](#).

14.1.9.7.3 Function Documentation

14.1.9.7.3.1 l4_error()

```
long l4_error (
    l4_msgtag_t tag) [inline]
```

Get IPC error code if any or message tag label otherwise for an IPC call.

This function shall only be used if the IPC operation includes a receive phase (usually a call operation), otherwise no tag label is received and the return value of this function is undefined.

Parameters

<i>tag</i>	Message tag returned by the IPC call.
------------	---------------------------------------

Returns

In case of an IPC error, a negative error code in the range of [L4_EIPC_LO](#) to [L4_EIPC_HI](#) (see [l4_ipc_to_errno\(\)](#) and [l4_ipc_tcr_error_t](#)), otherwise the tag label. By convention, the callee can signal errors via a negative tag label (negated value from [l4_error_code_t](#)) and success via a non-negative value.

Examples

[examples/libs/l4re/streammap/client.cc](#), [examples/sys/aliens/main.c](#), [examples/sys/isr/main.c](#), [examples/sys/migrate/thread_migration.c](#), [examples/sys/singlestep/main.c](#), [examples/sys/start-with-exc/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

Definition at line 646 of file [ipc.h](#).

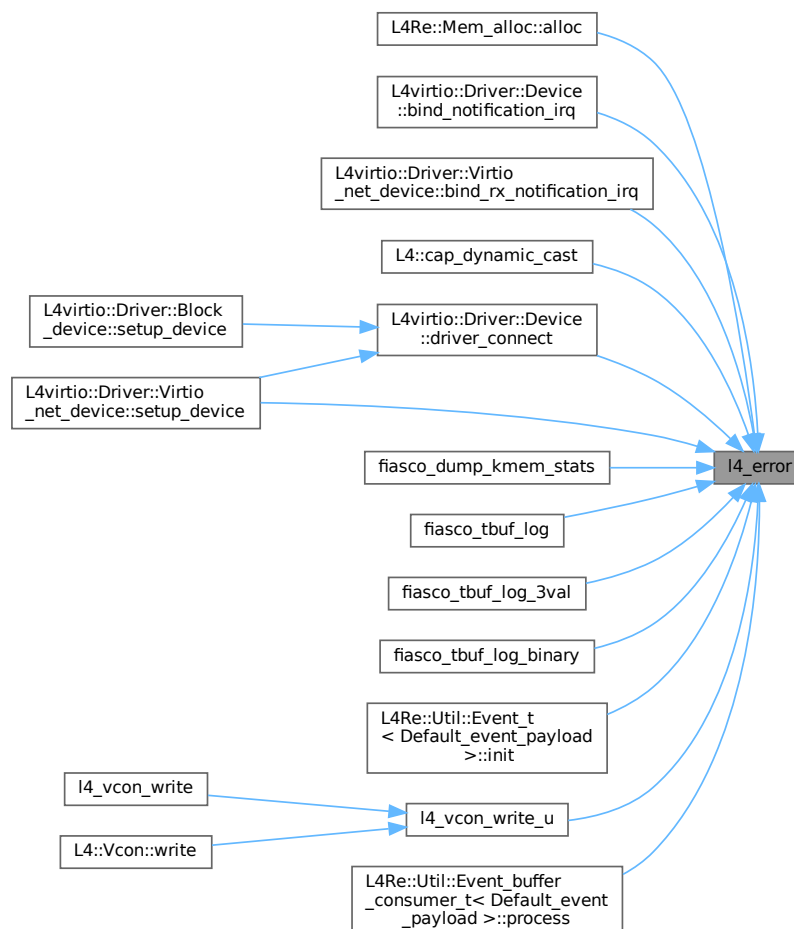
References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Referenced by [L4Re::Mem_alloc::alloc\(\)](#), [L4virtio::Driver::Device::bind_notification_irq\(\)](#), [L4virtio::Driver::Virtio_net_device::bind_rx_notification_irq\(\)](#), [L4::cap_dynamic_cast\(\)](#), [L4virtio::Driver::Device::driver_connect\(\)](#), [fiasco_dump_kmem_stats\(\)](#), [fiasco_tbuf_log\(\)](#), [fiasco_tbuf_log_3val\(\)](#), [fiasco_tbuf_log_binary\(\)](#), [L4Re::Util::Event_t< Default_event_payload >::init\(\)](#), [l4_vcon_write_u\(\)](#), [L4Re::Util::Event_buffer_consumer_t< Default_event_payload >::process\(\)](#), and [L4virtio::Driver::Virtio_net_device::setup_device\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.9.7.3.2 l4_ipc_error()

```

14_umword_t 14_ipc_error (
    14_msgtag_t tag,
    14_utcb_t * utcb) [inline]

```

Get the IPC error code for an IPC operation.

Parameters

<i>tag</i>	Message tag returned by the IPC operation.
<i>utcb</i>	UTCB that was used for the IPC operation.

Returns

0 if no error condition is set, error code otherwise (see [l4_ipc_tcr_error_t](#)).

Examples

examples/sys/ipc/ipc_example.c, and examples/sys/start-with-exc/main.c.

Definition at line 629 of file ipc.h.

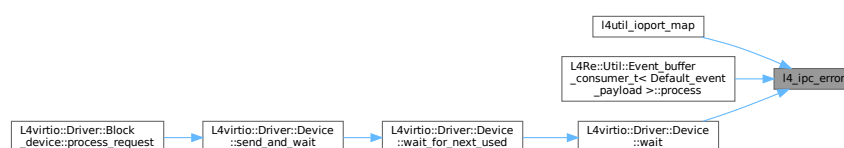
References [L4_IPC_ERROR_MASK](#), [L4_LIKELY](#), [l4_msgtag_has_error\(\)](#), and [L4_NOTHROW](#).

Referenced by [l4util_ioport_map\(\)](#), [L4Re::Util::Event_buffer_consumer_t< Default_event_payload >::process\(\)](#), and [L4virtio::Driver::Device::wait\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.9.7.3.3 `l4_ipc_error_code()`

```
int l4_ipc_error_code (
    l4_utcb_t * utcb) [inline]
```

Get the error condition of the last invocation from the TCR.

Precondition

`l4_msgtag_has_error(tag) == true`

Parameters

<i>utcb</i>	UTCB to check.
-------------	----------------

Returns

Error condition of type [l4_ipc_tcr_error_t](#).

Definition at line 658 of file [ipc.h](#).

References [L4_INLINE](#), [L4_IPC_ERROR_MASK](#), and [L4_NOTHROW](#).

14.1.9.7.3.4 `l4_ipc_is_rcv_error()`

```
int l4_ipc_is_rcv_error (
    l4_utcb_t * utcb) [inline]
```

Returns whether an error occurred in receive phase of an invocation.

Precondition

`l4_msgtag_has_error(tag) == true`

Parameters

<i>utcb</i>	UTCB to check.
-------------	----------------

Returns

Boolean value.

Definition at line 655 of file [ipc.h](#).

References [L4_INLINE](#), and [L4_NOTHROW](#).

14.1.9.7.3.5 `l4_ipc_is_snd_error()`

```
int l4_ipc_is_snd_error (
    l4_utcb_t * utcb) [inline]
```

Returns whether an error occurred in send phase of an invocation.

Precondition

`l4_msgtag_has_error(tag) == true`

Parameters

<i>utcb</i>	UTCB to check.
-------------	----------------

Returns

Boolean value.

Definition at line 652 of file [ipc.h](#).

References [L4_INLINE](#), and [L4_NOTHROW](#).

14.1.9.8 Realtime API

Collaboration diagram for Realtime API:



14.1.9.9 Message Tag

API related to the message tag data type.

Collaboration diagram for Message Tag:



Data Structures

- struct [l4_msgtag_t](#)
Message tag data structure.

Typedefs

- typedef struct l4_msgtag_t [l4_msgtag_t](#)
Message tag data structure.

Enumerations

- enum `L4_platform_ctl_proto` { `L4_PROTO_PLATFORM_CTL` = 0 }
Predefined protocol type for messages to platform-control objects.
- enum `L4_msgtag_protocol` {
`L4_PROTO_NONE` = 0 , `L4_PROTO_ALLOW_SYSCALL` = 1 , `L4_PROTO_PF_EXCEPTION` = 1 ,
`L4_PROTO_IRQ` = -1L ,
`L4_PROTO_PAGE_FAULT` = -2L , `L4_PROTO_EXCEPTION` = -5L , `L4_PROTO_SIGMA0` = -6L ,
`L4_PROTO_IO_PAGE_FAULT` = -8L ,
`L4_PROTO_THREAD_GROUP` = -9L , `L4_PROTO_KOBJECT` = -10L , `L4_PROTO_TASK` = -11L ,
`L4_PROTO_THREAD` = -12L ,
`L4_PROTO_LOG` = -13L , `L4_PROTO_SCHEDULER` = -14L , `L4_PROTO_FACTORY` = -15L ,
`L4_PROTO_VM` = -16L ,
`L4_PROTO_DMA_SPACE` = -17L , `L4_PROTO_IRQ_SENDER` = -18L , `L4_PROTO_SEMAPHORE` = -20L ,
`L4_PROTO_META` = -21L ,
`L4_PROTO_IOMMU` = -22L , `L4_PROTO_DEBUGGER` = -23L , `L4_PROTO_SMCCC` = -24L ,
`L4_PROTO_VCPU_CONTEXT` = -25L }
Message tag for IPC operations.
- enum `L4_msgtag_flags` { `L4_MSGTAG_ERROR` , `L4_MSGTAG_TRANSFER_FPU` , `L4_MSGTAG_SCHEDULE` ,
`L4_MSGTAG_PROPAGATE` = 0x4000 , `L4_MSGTAG_FLAGS` }
Flags for message tags.

Functions

- `l4_msgtag_t l4_msgtag` (long label, unsigned words, unsigned items, unsigned flags) `L4_NOTHROW`
Create a message tag from the specified values.
- long `l4_msgtag_label` (`l4_msgtag_t` t) `L4_NOTHROW`
Get the protocol of tag.
- unsigned `l4_msgtag_words` (`l4_msgtag_t` t) `L4_NOTHROW`
Get the number of untyped words.
- unsigned `l4_msgtag_items` (`l4_msgtag_t` t) `L4_NOTHROW`
Get the number of typed items.
- unsigned `l4_msgtag_flags` (`l4_msgtag_t` t) `L4_NOTHROW`
Get the flags.
- unsigned `l4_msgtag_has_error` (`l4_msgtag_t` t) `L4_NOTHROW`
Test for error indicator flag.
- unsigned `l4_msgtag_is_page_fault` (`l4_msgtag_t` t) `L4_NOTHROW`
Test for page-fault protocol.
- unsigned `l4_msgtag_is_exception` (`l4_msgtag_t` t) `L4_NOTHROW`
Test for exception protocol.
- unsigned `l4_msgtag_is_sigma0` (`l4_msgtag_t` t) `L4_NOTHROW`
Test for sigma0 protocol.
- unsigned `l4_msgtag_is_io_page_fault` (`l4_msgtag_t` t) `L4_NOTHROW`
Test for IO-page-fault protocol.

14.1.9.9.1 Detailed Description

API related to the message tag data type.

Include File

```
#include <l4/sys/types.h>
```

14.1.9.9.2 Typedef Documentation

14.1.9.9.2.1 l4_msgtag_t

```
typedef struct l4_msgtag_t l4_msgtag_t
```

Message tag data structure.

Include File

```
#include <l4/sys/types.h>
```

Describes the details of an IPC operation, in particular which parts of the UTCB have to be transmitted, and also flags to enable real-time and FPU extensions.

The message tag also contains a user-defined label that could be used to specify a protocol ID. Some negative values are reserved for kernel protocols such as page faults and exceptions.

The type must be treated completely opaque.

14.1.9.9.3 Enumeration Type Documentation

14.1.9.9.3.1 L4_msgtag_flags

```
enum L4_msgtag_flags
```

Flags for message tags.

Enumerator

L4_MSGTAG_ERROR	Error indicator flag.
L4_MSGTAG_TRANSFER_FPU	Enable FPU transfer flag for IPC. By enabling this flag when sending IPC, the sender indicates that the contents of the FPU shall be transferred to the receiving thread. However, the receiver has to indicate its willingness to receive FPU context in its buffer descriptor register (BDR).
L4_MSGTAG_SCHEDULE	Enable schedule in IPC flag. Usually IPC operations donate the remaining time slice of a thread to the called thread. Enabling this flag when sending IPC does a real scheduling decision. However, this flag decreases IPC performance.
L4_MSGTAG_FLAGS	Mask for all flags.

Definition at line 86 of file [types.h](#).

14.1.9.9.3.2 L4_msgtag_protocol

```
enum L4_msgtag_protocol
```

Message tag for IPC operations.

All predefined protocols used by the kernel.

Enumerator

L4_PROTO_NONE	Default protocol tag to reply to kernel.
L4_PROTO_ALLOW_SYSCALL	Allow an alien the system call.
L4_PROTO_PF_EXCEPTION	Make an exception out of a page fault.
L4_PROTO_IRQ	IRQ message.
L4_PROTO_PAGE_FAULT	Page fault message.
L4_PROTO_EXCEPTION	Exception.
L4_PROTO_SIGMA0	Sigma0 protocol.
L4_PROTO_IO_PAGE_FAULT	I/O page fault message.
L4_PROTO_THREAD_GROUP	Protocol for messages to a thread group obj.
L4_PROTO_KOBJECT	Protocol for messages to a generic kobject.
L4_PROTO_TASK	Protocol for messages to a task object.
L4_PROTO_THREAD	Protocol for messages to a thread object.
L4_PROTO_LOG	Protocol for messages to a log object.
L4_PROTO_SCHEDULER	Protocol for messages to a scheduler object.
L4_PROTO_FACTORY	Protocol for messages to a factory object.
L4_PROTO_VM	Protocol for messages to a virtual machine object.
L4_PROTO_DMA_SPACE	Protocol for (creating) kernel DMA space objects.
L4_PROTO_IRQ_SENDER	Protocol for IRQ senders (IRQ -> IPC).
L4_PROTO_SEMAPHORE	Protocol for semaphore objects.
L4_PROTO_META	Meta information protocol.
L4_PROTO_IOMMU	Protocol ID for IO-MMUs.
L4_PROTO_DEBUGGER	Protocol ID for the debugger.
L4_PROTO_SMCCC	Protocol ID for ARM SMCCC calls.
L4_PROTO_VCPU_CONTEXT	Protocol for hardware vCPU contexts.

Definition at line 38 of file [types.h](#).

14.1.9.9.3.3 L4_platform_ctl_proto

```
enum L4_platform_ctl_proto
```

Predefined protocol type for messages to platform-control objects.

Enumerator

L4_PROTO_PLATFORM_CTL	Protocol messages to a platform control object. See L4_platform_ctl_ops for allowed operations.
-----------------------	---

Definition at line 174 of file [platform_control.h](#).

14.1.9.9.4 Function Documentation

14.1.9.9.4.1 l4_msgtag()

```
l4_msgtag_t l4_msgtag (
    long label,
    unsigned words,
    unsigned items,
    unsigned flags) [inline]
```

Create a message tag from the specified values.

Message tag functions.

Parameters

<i>label</i>	The user-defined label
<i>words</i>	The number of untyped words within the UTCB
<i>items</i>	The number of typed items (e.g., flexpages) within the UTCB
<i>flags</i>	The IPC flags for realtime and FPU extensions

Returns

Message tag

Examples

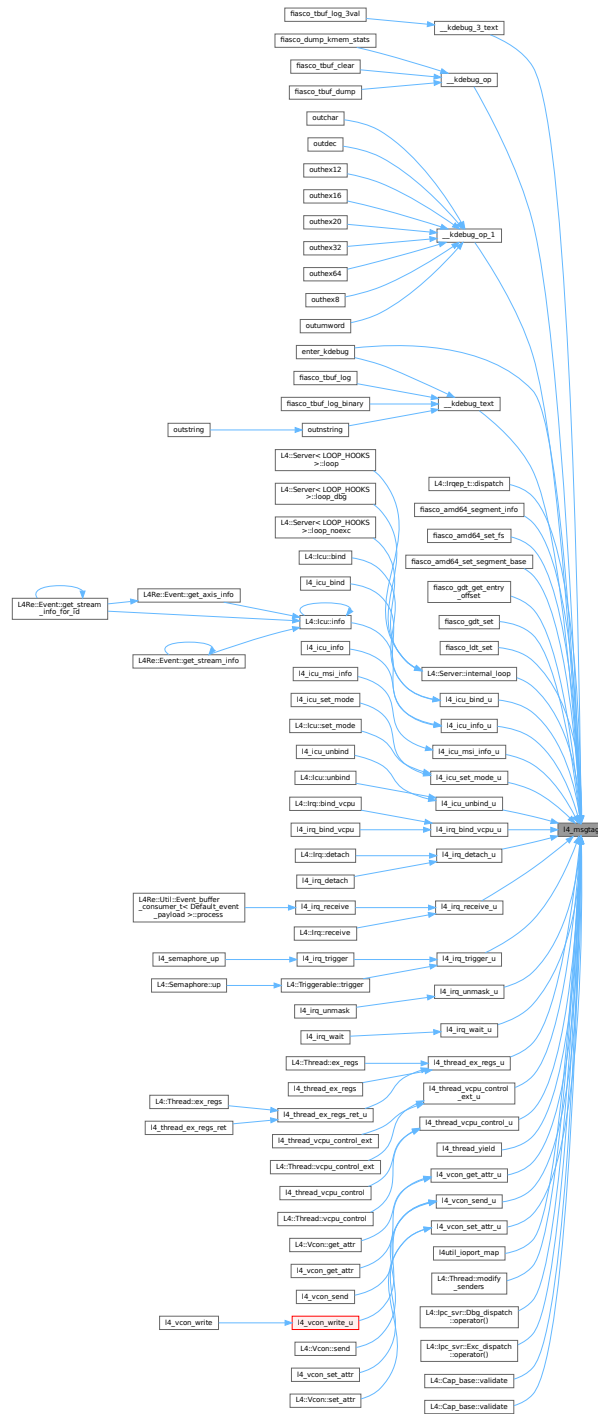
[examples/sys/aliens/main.c](#), [examples/sys/ipc/ipc_example.c](#), [examples/sys/singlestep/main.c](#), [examples/sys/start-with-exc/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

Definition at line 405 of file [types.h](#).

References [L4_NOTHROW](#).

Referenced by [__kdebug_3_text\(\)](#), [__kdebug_op\(\)](#), [__kdebug_op_1\(\)](#), [__kdebug_text\(\)](#), [L4::Irqep_t< Derived, BASE, bool >::dispatch\(\)](#), [enter_kdebug\(\)](#), [fiasco_amd64_segment_info\(\)](#), [fiasco_amd64_set_fs\(\)](#), [fiasco_amd64_set_segment_base\(\)](#), [fiasco_gdt_get_entry_offset\(\)](#), [fiasco_gdt_set\(\)](#), [fiasco_ldt_set\(\)](#), [L4::Server< LOOP_HOOKS >::internal_loop\(\)](#), [l4_icu_bind_u\(\)](#), [l4_icu_info_u\(\)](#), [l4_icu_msi_info_u\(\)](#), [l4_icu_set_mode_u\(\)](#), [l4_icu_unbind_u\(\)](#), [l4_irq_bind_vcpu_u\(\)](#), [l4_irq_detach_u\(\)](#), [l4_irq_receive_u\(\)](#), [l4_irq_trigger_u\(\)](#), [l4_irq_unmask_u\(\)](#), [l4_irq_wait_u\(\)](#), [l4_thread_ex_regs_u\(\)](#), [l4_thread_vcpu_control_ext_u\(\)](#), [l4_thread_vcpu_control_u\(\)](#), [l4_thread_yield\(\)](#), [l4_vcon_get_attr_u\(\)](#), [l4_vcon_send_u\(\)](#), [l4_vcon_set_attr_u\(\)](#), [l4util_ioport_map\(\)](#), [L4::Thread::modify_senders\(\)](#), [L4::lpc_svr::Dbg_dispatch< R, Exc, Printer >::operator\(\)\(\)](#), [L4::lpc_svr::Exc_dispatch< R, Exc >::operator\(\)\(\)](#), [L4::Cap_base::validate\(\)](#), and [L4::Cap_base::validate\(\)](#).

Here is the caller graph for this function:



14.1.9.9.4.2 l4_msgtag_flags()

```
unsigned l4_msgtag_flags (
    l4_msgtag_t t) [inline]
```

Get the flags.

The flag are defined by [L4_msgtag_flags](#).

Parameters

<i>t</i>	The tag
----------	---------

Returns

Boolean value

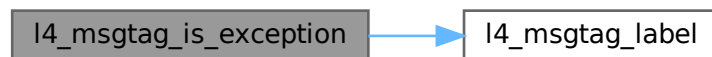
Examples

[examples/sys/aliens/main.c](#), [examples/sys/singlestep/main.c](#), and [examples/sys/start-with-exc/main.c](#).

Definition at line 448 of file [types.h](#).

References [L4_INLINE](#), [l4_msgtag_label\(\)](#), [L4_NOTHROW](#), and [L4_PROTO_EXCEPTION](#).

Here is the call graph for this function:

**14.1.9.9.4.5 l4_msgtag_is_io_page_fault()**

```
unsigned l4_msgtag_is_io_page_fault (  
    l4_msgtag_t t) [inline]
```

Test for IO-page-fault protocol.

Parameters

<i>t</i>	The tag
----------	---------

Returns

Boolean value

Definition at line 454 of file [types.h](#).

References [L4_INLINE](#), [l4_msgtag_label\(\)](#), [L4_NOTHROW](#), and [L4_PROTO_IO_PAGE_FAULT](#).

Here is the call graph for this function:



14.1.9.9.4.6 l4_msgtag_is_page_fault()

```
unsigned l4_msgtag_is_page_fault (  
    l4_msgtag_t t) [inline]
```

Test for page-fault protocol.

Parameters

<i>t</i>	The tag
----------	---------

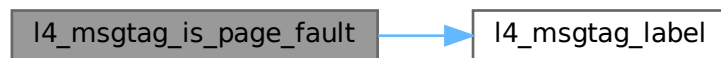
Returns

Boolean value

Definition at line 445 of file [types.h](#).

References [L4_INLINE](#), [l4_msgtag_label\(\)](#), [L4_NOTHROW](#), and [L4_PROTO_PAGE_FAULT](#).

Here is the call graph for this function:



14.1.9.9.4.7 l4_msgtag_is_sigma0()

```
unsigned l4_msgtag_is_sigma0 (  
    l4_msgtag_t t) [inline]
```

Test for sigma0 protocol.

Parameters

<i>t</i>	The tag
----------	---------

Returns

Boolean value

Definition at line 451 of file [types.h](#).

References [L4_INLINE](#), [l4_msgtag_label\(\)](#), [L4_NOTHROW](#), and [L4_PROTO_SIGMA0](#).

Here is the call graph for this function:

**14.1.9.9.4.8 l4_msgtag_items()**

```
unsigned l4_msgtag_items (  
    l4_msgtag_t t) [inline]
```

Get the number of typed items.

Parameters

<i>t</i>	The tag
----------	---------

Returns

Number of items.

Definition at line 431 of file [types.h](#).

References [L4_NOTHROW](#).

Referenced by [l4util_ioport_map\(\)](#).

Here is the caller graph for this function:



14.1.9.9.4.9 l4_msgtag_label()

```
long l4_msgtag_label (  
    l4_msgtag_t t) [inline]
```

Get the protocol of tag.

Parameters

<i>t</i>	The tag
----------	---------

Returns

Label

Examples

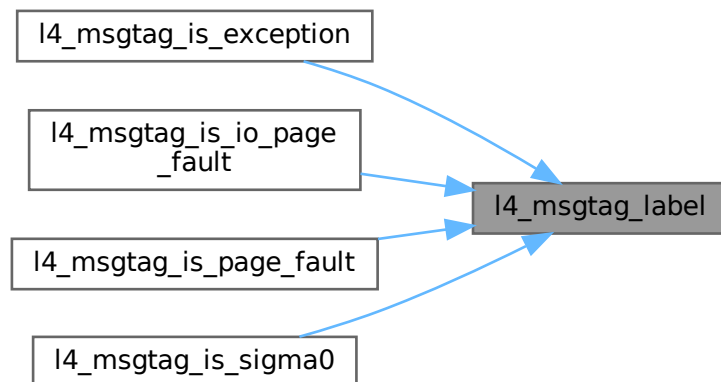
[examples/sys/singlestep/main.c](#), and [examples/sys/start-with-exc/main.c](#).

Definition at line 417 of file [types.h](#).

References [L4_NOTHROW](#).

Referenced by [l4_msgtag_is_exception\(\)](#), [l4_msgtag_is_io_page_fault\(\)](#), [l4_msgtag_is_page_fault\(\)](#), and [l4_msgtag_is_sigma0\(\)](#).

Here is the caller graph for this function:



14.1.9.9.4.10 `l4_msgtag_words()`

```
unsigned l4_msgtag_words (  
    l4_msgtag_t t) [inline]
```

Get the number of untyped words.

Parameters

<i>t</i>	The tag
----------	---------

Returns

Number of words

Examples

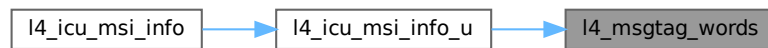
[examples/sys/utcb-ipc/main.c](#).

Definition at line 427 of file [types.h](#).

References [L4_NOTHROW](#).

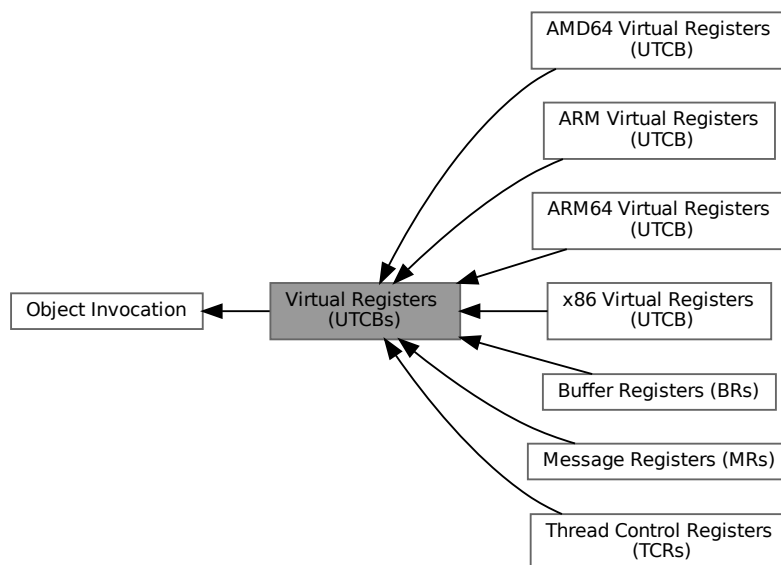
Referenced by [l4_icu_msi_info_u\(\)](#).

Here is the caller graph for this function:

**14.1.9.10 Virtual Registers (UTCBS)**

[L4 Virtual Registers \(UTCB\)](#).

Collaboration diagram for Virtual Registers (UTCBS):



Topics

- Message Registers (MRs) ??
- [Buffer](#) Registers (BRs) ??
- Thread Control Registers (TCRs) ??
- ARM Virtual Registers (UTCB) ??
- ARM64 Virtual Registers (UTCB) ??
- AMD64 Virtual Registers (UTCB) ??
- x86 Virtual Registers (UTCB) ??

Files

- file [utcb.h](#)
UTCB definitions for ARM.
- file [utcb.h](#)
UTCB definitions for ARM64.
- file [utcb.h](#)
UTCB definitions for AMD64.
- file [utcb.h](#)
UTCB definitions for x86.

Typedefs

- typedef struct [l4_utcb_t](#) [l4_utcb_t](#)
Opaque type for the UTCB.

Functions

- [l4_utcb_t](#) * [l4_utcb](#) (void) [L4_NOTHROW](#) [L4_PURE](#)
Get the UTCB address.
- [l4_msg_regs_t](#) * [l4_utcb_mr](#) (void) [L4_NOTHROW](#) [L4_PURE](#)
Get the message-register block of a UTCB.
- [l4_buf_regs_t](#) * [l4_utcb_br](#) (void) [L4_NOTHROW](#) [L4_PURE](#)
Get the buffer-register block of a UTCB.
- [l4_thread_regs_t](#) * [l4_utcb_tcr](#) (void) [L4_NOTHROW](#) [L4_PURE](#)
Get the thread-control-register block of a UTCB.

14.1.9.10.1 Detailed Description

[L4](#) Virtual Registers (UTCB).

Include File

```
#include <l4/sys/utcb.h>
```

The virtual registers are part of the micro-kernel API and are located in the user-level thread control block (UTCB). The UTCB is a data structure defined by the micro kernel and located on kernel-provided memory. Each [L4](#) thread gets a unique UTCB assigned when it is bound to a task (see [Thread Control](#) , [l4_thread_control_bind\(\)](#) for more information).

The UTCB is arranged in three blocks of virtual registers.

- [Thread Control Registers \(TCRs\)](#)
- [Message Registers \(MRs\)](#)
- [Buffer Registers \(BRs\)](#)

To access the contents of the virtual registers the [l4_utcb_mr\(\)](#), [l4_utcb_tcr\(\)](#), and [l4_utcb_br\(\)](#) functions must be used.

14.1.9.10.2 Typedef Documentation

14.1.9.10.2.1 [l4_utcb_t](#)

```
typedef struct l4\_utcb\_t l4\_utcb\_t
```

Opaque type for the UTCB.

To access the contents of the virtual registers the [l4_utcb_mr\(\)](#), [l4_utcb_tcr\(\)](#), and [l4_utcb_br\(\)](#) functions must be used.

Examples

[examples/sys/aliens/main.c](#), [examples/sys/singlestep/main.c](#), [examples/sys/start-with-exc/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

Definition at line [56](#) of file [utcb.h](#).

14.1.9.10.3 Function Documentation

14.1.9.10.3.1 l4_utcb_br()

```
l4_buf_regs_t * l4_utcb_br (  
    void )    [inline]
```

Get the buffer-register block of a UTCB.

Returns

A pointer to the buffer-register block of `u`.

Definition at line 361 of file `utcb.h`.

References [L4_INLINE](#), [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Referenced by [l4util_ioport_map\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.9.10.3.2 l4_utcb_mr()

```
l4_msg_regs_t * l4_utcb_mr (  
    void )    [inline]
```

Get the message-register block of a UTCB.

Returns

A pointer to the message-register block of `u`.

Examples

[examples/sys/aliens/main.c](#), [examples/sys/ipc/ipc_example.c](#), [examples/sys/singlestep/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

Definition at line 358 of file [utcb.h](#).

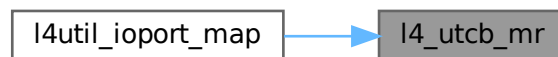
References [L4_INLINE](#), [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Referenced by [l4util_ioport_map\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.9.10.3.3 l4_utcb_tcr()

```
l4_thread_regs_t * l4_utcb_tcr (  
    void ) [inline]
```

Get the thread-control-register block of a UTCB.

Returns

A pointer to the thread-control-register block of `u`.

Definition at line 364 of file `utcb.h`.

References [L4_INLINE](#), [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:

**14.1.9.10.4 Message Registers (MRs)**

Collaboration diagram for Message Registers (MRs):

**Topics**

- [Exception registers](#) ??
Overly definition of the MRs for exception messages.

Data Structures

- union [l4_msg_regs_t](#)
Encapsulation of the message-register block in the UTCB.

Typedefs

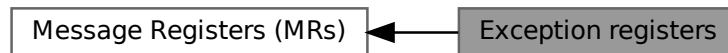
- typedef union [l4_msg_regs_t](#) **[l4_msg_regs_t](#)**
Encapsulation of the message-register block in the UTCB.

14.1.9.10.4.1 Detailed Description

14.1.9.10.4.2 Exception registers

Overly definition of the MRs for exception messages.

Collaboration diagram for Exception registers:



Functions

- `l4_exc_regs_t * l4_utcb_exc (void) L4_NOTHROW L4_PURE`
Get the message-register block of a UTCTB (for an exception IPC).
- `l4_umword_t l4_utcb_exc_pc (l4_exc_regs_t const *u) L4_NOTHROW L4_PURE`
Access function to get the program counter of the exception state.
- `void l4_utcb_exc_pc_set (l4_exc_regs_t *u, l4_addr_t pc) L4_NOTHROW`
Set the program counter register in the exception state.
- `unsigned long l4_utcb_exc_typeval (l4_exc_regs_t const *u) L4_NOTHROW L4_PURE`
Get the value out of an exception UTCTB that describes the type of exception.
- `int l4_utcb_exc_is_pf (l4_exc_regs_t const *u) L4_NOTHROW L4_PURE`
Check whether an exception IPC is a page fault.
- `l4_addr_t l4_utcb_exc_pfa (l4_exc_regs_t const *u) L4_NOTHROW L4_PURE`
Function to get the L4 style page fault address out of an exception.
- `int l4_utcb_exc_is_ex_regs_exception (l4_exc_regs_t const *u) L4_NOTHROW L4_PURE`
Check whether an exception IPC was triggered via `l4_thread_ex_regs()`.

Detailed Description

Overly definition of the MRs for exception messages.

Function Documentation

`l4_utcb_exc()`

```
l4_exc_regs_t * l4_utcb_exc (
    void ) [inline]
```

Get the message-register block of a UTCTB (for an exception IPC).

Returns

A pointer to the exception message in `u`.

Examples

[examples/sys/aliens/main.c](#), and [examples/sys/singlestep/main.c](#).

Definition at line 367 of file [utcb.h](#).

References [L4_INLINE](#), [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:

**`l4_utcb_exc_is_ex_regs_exception()`**

```
int l4_utcb_exc_is_ex_regs_exception (
    l4_exc_regs_t const * u) [inline]
```

Check whether an exception IPC was triggered via [l4_thread_ex_regs\(\)](#).

Return values

<code>0</code>	Exception was not triggered through <code>ex_regs</code> .
<code>!=0</code>	Exception was triggered through <code>ex_regs</code> .

This function checks if the exception was emitted by using the `L4_THREAD_EX_REGS_TRIGGER_EXCEPTION` flag in an [l4_thread_ex_regs\(\)](#) call.

Definition at line 110 of file [utcb.h](#).

References [L4_INLINE](#), [L4_NOTHROW](#), and [l4_utcb_exc_typeval\(\)](#).

Here is the call graph for this function:



l4_utcb_exc_is_pf()

```
int l4_utcb_exc_is_pf (
    l4_exc_regs_t const * u) [inline]
```

Check whether an exception IPC is a page fault.

Returns

0 if not, != 0 if yes

Function to check whether an exception IPC is a page fault, also applies to I/O pagefaults.

Definition at line 100 of file [utcb.h](#).

References [L4_INLINE](#), and [L4_NOTHROW](#).

l4_utcb_exc_pc()

```
l4_umword_t l4_utcb_exc_pc (
    l4_exc_regs_t const * u) [inline]
```

Access function to get the program counter of the exception state.

Parameters

<i>u</i>	UTCB
----------	------

Returns

The program counter register out of the exception state.

Examples

[examples/sys/aliens/main.c](#), and [examples/sys/singlestep/main.c](#).

Definition at line 85 of file [utcb.h](#).

References [L4_INLINE](#), and [L4_NOTHROW](#).

l4_utcb_exc_pc_set()

```
void l4_utcb_exc_pc_set (
    l4_exc_regs_t * u,
    l4_addr_t pc) [inline]
```

Set the program counter register in the exception state.

Parameters

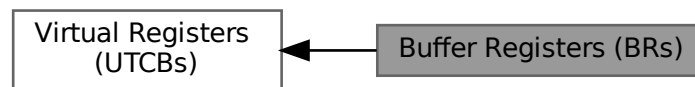
<i>u</i>	UTCB
<i>pc</i>	The program counter to set.

Definition at line 90 of file [utcb.h](#).

References [L4_INLINE](#), and [L4_NOTHROW](#).

14.1.9.10.5 Buffer Registers (BRs)

Collaboration diagram for Buffer Registers (BRs):



Data Structures

- struct [l4_buf_regs_t](#)
Encapsulation of the buffer-registers block in the UTCB.

Typedefs

- typedef struct l4_buf_regs_t [l4_buf_regs_t](#)
Encapsulation of the buffer-registers block in the UTCB.

Enumerations

- enum [l4_buffer_desc_consts_t](#) { [L4_BDR_MEM_SHIFT](#) = 0 , [L4_BDR_IO_SHIFT](#) = 5 , [L4_BDR_OBJ_SHIFT](#) = 10 , [L4_BDR_OFFSET_MASK](#) = (1UL << 20) - 1 }
Constants for buffer descriptors.

Functions

- void [l4_utcb_inherit_fpu](#) (int switch_on) [L4_NOTHROW](#)
Enable or disable inheritance of FPU state to receiver.

14.1.9.10.5.1 Detailed Description

14.1.9.10.5.2 Enumeration Type Documentation

[l4_buffer_desc_consts_t](#)

enum [l4_buffer_desc_consts_t](#)

Constants for buffer descriptors.

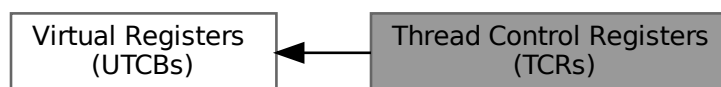
Enumerator

L4_BDR_MEM_SHIFT	Bit offset for the memory-buffer index.
L4_BDR_IO_SHIFT	Bit offset for the IO-buffer index.
L4_BDR_OBJ_SHIFT	Bit offset for the capability-buffer index.

Definition at line 303 of file [consts.h](#).

14.1.9.10.6 Thread Control Registers (TCRs)

Collaboration diagram for Thread Control Registers (TCRs):



Data Structures

- struct [l4_thread_regs_t](#)
Encapsulation of the thread-control-register block of the UTCB.

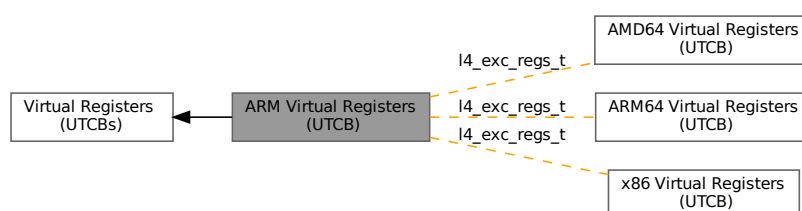
Typedefs

- typedef struct l4_thread_regs_t **l4_thread_regs_t**
Encapsulation of the thread-control-register block of the UTCB.

14.1.9.10.6.1 Detailed Description

14.1.9.10.7 ARM Virtual Registers (UTCB)

Collaboration diagram for ARM Virtual Registers (UTCB):



Data Structures

- struct [l4_exc_regs_t](#)
UTCB structure for exceptions.

Typedefs

- typedef struct l4_exc_regs_t **l4_exc_regs_t**
UTCB structure for exceptions.

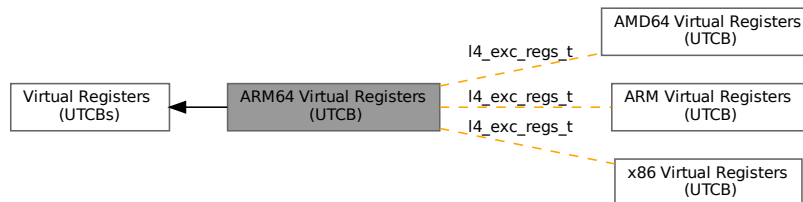
Enumerations

- enum [L4_utcb_consts_arm](#)
UTCB constants for ARM.

14.1.9.10.7.1 Detailed Description

14.1.9.10.8 ARM64 Virtual Registers (UTCB)

Collaboration diagram for ARM64 Virtual Registers (UTCB):



Data Structures

- struct [l4_exc_regs_t](#)
UTCB structure for exceptions.

Typedefs

- typedef struct l4_exc_regs_t **l4_exc_regs_t**
UTCB structure for exceptions.

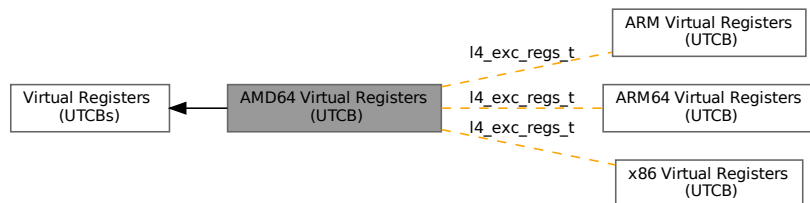
Enumerations

- enum [L4_utcb_consts_arm64](#)
UTCB constants for ARM64.

14.1.9.10.8.1 Detailed Description

14.1.9.10.9 AMD64 Virtual Registers (UTCB)

Collaboration diagram for AMD64 Virtual Registers (UTCB):



Data Structures

- struct [l4_exc_regs_t](#)
UTCB structure for exceptions.

Typedefs

- typedef struct [l4_exc_regs_t](#) **l4_exc_regs_t**
UTCB structure for exceptions.

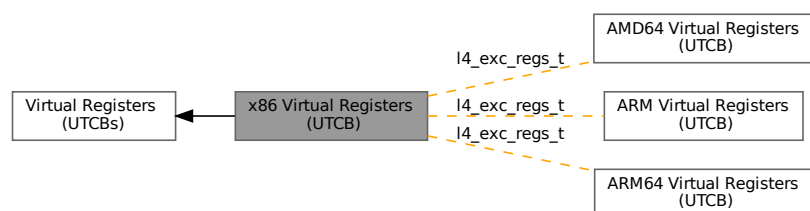
Enumerations

- enum [L4_utcb_consts_amd64](#)
UTCB constants for AMD64.

14.1.9.10.9.1 Detailed Description

14.1.9.10.10 x86 Virtual Registers (UTCB)

Collaboration diagram for x86 Virtual Registers (UTCB):



Data Structures

- struct [l4_exc_regs_t](#)
UTCB structure for exceptions.

Typedefs

- typedef struct l4_exc_regs_t [l4_exc_regs_t](#)
UTCB structure for exceptions.

Enumerations

- enum [L4_utcb_consts_x86](#) {
[L4_UTCB_EXCEPTION_REGS_SIZE](#) = 19 , [L4_UTCB_GENERIC_DATA_SIZE](#) = 63 , [L4_UTCB_GENERIC_BUFFERS_SIZE](#) = 58 , [L4_UTCB_MSG_REGS_OFFSET](#) = 0 ,
[L4_UTCB_BUF_REGS_OFFSET](#) = 64 * sizeof(l4_umword_t) , [L4_UTCB_THREAD_REGS_OFFSET](#) = 123 * sizeof(l4_umword_t) , [L4_UTCB_INHERIT_FPU](#) = 1UL << 24 , [L4_UTCB_OFFSET](#) = 512 }
UTCB constants for x86.

14.1.9.10.10.1 Detailed Description

14.1.9.10.10.2 Enumeration Type Documentation

L4_utcb_consts_x86

enum [L4_utcb_consts_x86](#)

UTCB constants for x86.

Enumerator

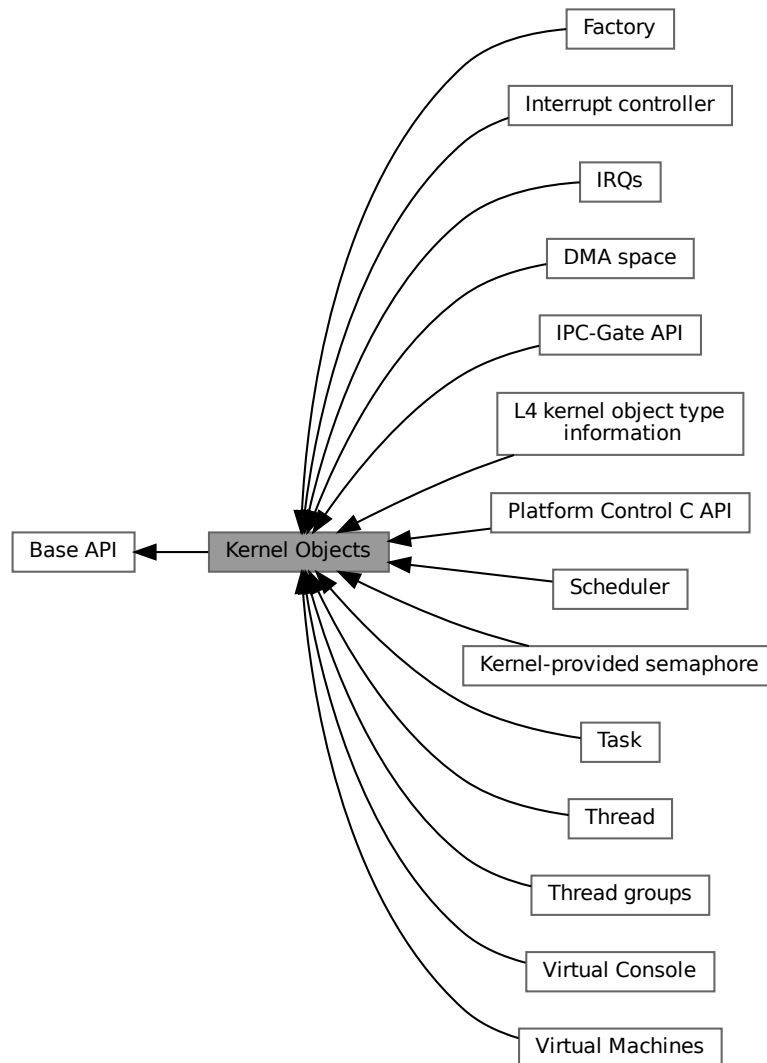
L4_UTCB_EXCEPTION_REGS_SIZE	Number if message registers used for exception IPC.
L4_UTCB_GENERIC_DATA_SIZE	Total number of message register (MRs) available.
L4_UTCB_GENERIC_BUFFERS_SIZE	Total number of buffer registers (BRs) available.
L4_UTCB_MSG_REGS_OFFSET	Offset of MR[0] relative to the UTCB pointer.
L4_UTCB_BUF_REGS_OFFSET	Offset of BR[0] relative to the UTCB pointer.
L4_UTCB_THREAD_REGS_OFFSET	Offset of TCR[0] relative to the UTCB pointer.
L4_UTCB_INHERIT_FPU	BDR flag to accept reception of FPU state.
L4_UTCB_OFFSET	Offset of two consecutive UTCBs.

Definition at line 30 of file [utcb.h](#).

14.1.10 Kernel Objects

API of kernel objects.

Collaboration diagram for Kernel Objects:



Topics

- IPC-Gate API ??
The C IPC gate interface, see [L4::lpc_gate](#) for the C++ interface.
- DMA space ??
A DMA space represents a device memory address space managed by an IOMMU.
-

L4 kernel object type information	??
<i>Type information for L4 server objects that can be called via IPC.</i>	
•	
Factory	??
<i>C factory interface to create objects, see L4::Factory for the C++ interface.</i>	
•	
Virtual Machines	??
<i>Virtual Machine API.</i>	
•	
Interrupt controller	??
<i>The C Icu interface, see L4::Icu for the C++ interface.</i>	
•	
IRQs	??
<i>C IRQ interface, see L4::Irq for the C++ interface.</i>	
•	
Platform Control C API	??
<i>C interface for controlling platform-wide properties, see L4::Platform_control for the C++ interface.</i>	
•	
Scheduler	??
<i>C interface of the Scheduler kernel object, see L4::Scheduler for the C++ interface.</i>	
•	
Kernel-provided semaphore	??
<i>C semaphore interface, see L4::Semaphore for the C++ interface.</i>	
•	
Task	??
<i>C interface of the Task kernel object, see L4::Task for the C++ interface.</i>	
•	
Thread	??
<i>C Thread object interface, see L4::Thread for the C++ interface.</i>	
•	
Thread groups	??
<i>C thread group interface, see L4::Thread_group for the C++ interface.</i>	
•	
Virtual Console	??
<i>C Virtual console interface for simple character based input and output, see L4::Vcon for the C++ interface.</i>	

Data Structures

- class [L4::Kobject](#)
Base class for all kinds of kernel objects and remote objects, referenced by capabilities.
- class [L4::Vm](#)
Virtual machine host address space.

14.1.10.1 Detailed Description

API of kernel objects.

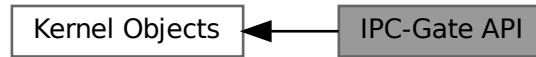
Include File

```
#include <l4/sys/kernel_object.h>
```

14.1.10.2 IPC-Gate API

The C IPC gate interface, see [L4::ipc_gate](#) for the C++ interface.

Collaboration diagram for IPC-Gate API:



Functions

- [l4_msgtag_t l4_ipc_gate_get_infos](#) ([l4_cap_idx_t](#) gate, [l4_umword_t](#) *label)
Get information about the IPC-gate.
- [l4_msgtag_t l4_rcv_ep_bind_thread](#) ([l4_cap_idx_t](#) ep, [l4_cap_idx_t](#) thread, [l4_umword_t](#) label)
Bind the IPC receive endpoint to a thread.
- [l4_msgtag_t l4_rcv_ep_bind_snd_destination](#) ([l4_cap_idx_t](#) ep, [l4_cap_idx_t](#) snd_dst, [l4_umword_t](#) label)
Bind the IPC receive endpoint to a send destination (a thread).

14.1.10.2.1 Detailed Description

The C IPC gate interface, see [L4::ipc_gate](#) for the C++ interface.

IPC gates are used to create secure communication channels between protection domains. An IPC gate can be created using the [Factory](#) interface.

Depending on the permissions of the capability used, an IPC gate forwards IPC to the [Thread](#) the IPC gate is *bound* to (cf. [l4_rcv_ep_bind_thread\(\)](#) and [l4_rcv_ep_bind_snd_destination\(\)](#)). If the capability has the [L4_FPAGE_C_IPCGATE_SVR](#) permission, only IPC using a protocol different from the [L4_PROTO_KOBJECT](#) protocol is forwarded. Without the [L4_FPAGE_C_IPCGATE_SVR](#) permission, all IPC is forwarded. The latter is the usual case for a client in a client/server scenario. When not bound to a thread or thread group yet, the forwarded IPC blocks until the IPC gate is bound to a thread or thread group, or the IPC times out.

Forwarded IPC is always forwarded to the userland of the thread the IPC gate is bound to, either directly or indirectly using a thread group. That means, the [Thread](#) interface of that thread is not accessible via an IPC gate. The [IPC-Gate API](#) of an IPC gate is only accessible if the capability used has the [L4_FPAGE_C_IPCGATE_SVR](#) permission (cf. previous paragraph). Conversely that means, if the capability used lacks the [L4_FPAGE_C_IPCGATE_SVR](#) permission, [IPC-Gate API](#) calls are forwarded to the thread or thread group the IPC gate is bound to instead of being processed by the IPC gate itself. In a client/server scenario, a client should only get IPC gate capabilities without [L4_FPAGE_C_IPCGATE_SVR](#) permission so the client cannot tamper with the IPC gate.

When binding an IPC gate to a thread or thread group, a user-defined, kernel protected, machine-word sized payload called the IPC gate's *label* is assigned to the IPC gate (note that the two least significant bits of the label must be zero; cf. [l4_rcv_ep_bind_thread\(\)](#) and [l4_rcv_ep_bind_snd_destination\(\)](#)). When a send-only IPC or call IPC is forwarded via an IPC gate, the label provided by the sender is ignored and replaced by the IPC gate's label where the two least significant bits are set to the [L4_CAP_FPAGE_S](#) and [L4_CAP_FPAGE_W](#) permissions of the capability used. The replaced label is only visible to the thread the IPC gate is bound to upon receive (or to the

selected thread from the thread group the IPC gate is bound to). However, the configured label of an IPC gate can also be queried via [l4_ipc_gate_get_infos\(\)](#) if the capability used has the [L4_FPAGE_C_IPCGATE_SVR](#) permission.

When deleting an IPC gate or when unbinding it from a thread or thread group, the label of IPC already in flight won't be changed. To ensure that no IPC from this IPC gate is received by a thread with an unexpected label, [l4_thread_modify_sender_start\(\)](#) shall be used to change the labels of every pending IPC to that gate. This is also required if the label of an already bound IPC gate is changed. It is not necessary after binding the IPC gate to a thread or thread group for the first time.

When binding a currently bound IPC gate to a new thread or thread group, the same label should be used that was used with the old thread. Otherwise the old and the new thread need to synchronize to avoid IPC messages with unexpected labels.

Include File

```
#include <l4/sys/ipc_gate.h>
```

For the C++ interface refer to the [L4::ipc_gate](#) documentation.

See also

[Object Invocation](#)

14.1.10.2.2 Function Documentation

14.1.10.2.2.1 l4_ipc_gate_get_infos()

```
l4_msgtag_t l4_ipc_gate_get_infos (
    l4_cap_idx_t gate,
    l4_umword_t * label) [inline]
```

Get information about the IPC-gate.

Parameters

	<i>gate</i>	The IPC gate object to get information about.
out	<i>label</i>	The label of the IPC gate is returned here.

Returns

System call return tag.

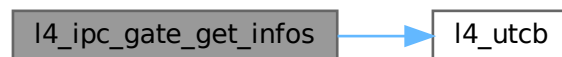
Precondition

If `gate` does not possess the [L4_FPAGE_C_IPCGATE_SVR](#) right, the kernel will not perform this operation. Instead, the underlying IPC message will be forwarded to the thread or thread group the IPC gate is bound to, blocking the caller if the IPC gate is not bound yet.

Definition at line 166 of file [ipc_gate.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:

**14.1.10.2.2.2 l4_rcv_ep_bind_snd_destination()**

```

l4_msgtag_t l4_rcv_ep_bind_snd_destination (
    l4_cap_idx_t ep,
    l4_cap_idx_t snd_dst,
    l4_umword_t label) [inline]
  
```

Bind the IPC receive endpoint to a send destination (a thread).

Parameters

<i>ep</i>	The IPC receive endpoint object.
<i>snd_dst</i>	The send destination (thread) object <i>ep</i> shall be bound to.
<i>label</i>	Label to assign to <i>ep</i> . For IPC gates, the two least significant bits must be set to zero.

Returns

Syscall return tag containing one of the following return codes.

Return values

<i>L4_EOK</i>	Operation successful.
<i>-L4_EINVAL</i>	<i>snd_dst</i> is not a thread object or other arguments were malformed.
<i>-L4_EPERM</i>	No L4_CAP_FPAGE_S right on <i>ep</i> or <i>snd_dst</i> .

Precondition

If `ep` is an IPC gate capability without the `L4_FPAGE_C_IPCGATE_SVR` right, the kernel will not perform this operation. Instead, the underlying IPC message will be forwarded to the send destination the IPC gate is bound to, blocking the caller if the IPC gate was not bound yet.

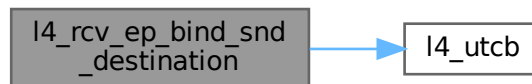
The specified `label` is passed to the receiver of the incoming IPC. It is possible to re-bind a receive endpoint to the same or a different send destination. In this case, IPC already in flight will be delivered with the old label to the previously bound thread unless `l4_thread_modify_sender_start()` is used to change these labels.

Definition at line 138 of file `rcv_endpoint.h`.

References `l4_utcb()`.

Referenced by `L4::Rcv_endpoint::bind_snd_destination()`.

Here is the call graph for this function:



Here is the caller graph for this function:

**14.1.10.2.2.3 l4_rcv_ep_bind_thread()**

```

l4_msgtag_t l4_rcv_ep_bind_thread (
    l4_cap_idx_t ep,
    l4_cap_idx_t thread,
    l4_umword_t label) [inline]
  
```

Bind the IPC receive endpoint to a thread.

Parameters

<i>ep</i>	The IPC receive endpoint object.
<i>thread</i>	The thread object <i>ep</i> shall be bound to.
<i>label</i>	Label to assign to <i>ep</i> . For IPC gates, the two least significant bits must be set to zero.

Returns

Syscall return tag containing one of the following return codes.

Return values

<i>L4_EOK</i>	Operation successful.
<i>-L4_EINVAL</i>	<i>thread</i> is not a thread object or other arguments were malformed.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.

Precondition

The capabilities *ep* and *thread* both must have the permission [L4_CAP_FPAGE_S](#).

If *ep* is an IPC gate capability without the [L4_FPAGE_C_IPCGATE_SVR](#) right, the kernel will not perform this operation. Instead, the underlying IPC message will be forwarded to the thread the IPC gate is bound to, blocking the caller if the IPC gate was not bound yet.

The specified *label* is passed to the receiver of the incoming IPC. It is possible to re-bind a receive endpoint to the same or a different thread. In this case, IPC already in flight will be delivered with the old label to the previously bound thread unless [l4_thread_modify_sender_start\(\)](#) is used to change these labels.

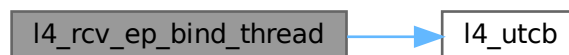
Examples

[examples/sys/isr/main.c](#).

Definition at line [131](#) of file [rcv_endpoint.h](#).

References [l4_utcb\(\)](#).

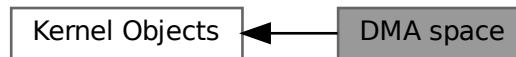
Here is the call graph for this function:



14.1.10.3 DMA space

A DMA space represents a device memory address space managed by an IOMMU.

Collaboration diagram for DMA space:



A DMA space represents a device memory address space managed by an IOMMU.

That is, it manages the translation of virtual addresses used by devices to physical addresses. It is accessed via the [L4::Task](#) interface, but with the following caveats:

- No threads can be bound to it.
- No objects (and IO ports on IA32) can be mapped to it.
- No kernel-user memory can be added to it.
- It must be constructed by passing the `L4_PROTO_DMA_SPACE` protocol constant to the kernel factory's [L4::Factory.create\(\)](#) call.

A DMA space must be bound to an [L4::iommu](#) to enable the address translation for specific devices.

The kernel factory allows to create DMA spaces only if the kernel has been configured with IOMMU support and if an IOMMU was detected.

14.1.10.4 L4 kernel object type information

Type information for [L4](#) server objects that can be called via IPC.

Collaboration diagram for L4 kernel object type information:



Data Structures

- struct [L4::Type_info](#)
Dynamic Type Information for [L4Re](#) Interfaces.
- struct [L4::Kobject_typeid< T >](#)
Meta object for handling access to type information of Kobjects.
- class [L4::Kobject_t< Derived, Base, PROTO, S_DEMAND >](#)
Helper class to create an [L4Re](#) interface class that is derived from a single base class.
- class [L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND >](#)
Helper class to create an [L4Re](#) interface class that is derived from two base classes (see [L4::Kobject_t](#)).
- struct [L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND >](#)
Helper class to create an [L4Re](#) interface class that is derived from three base classes (see [L4::Kobject_t](#)).
- struct [L4::Kobject_x< Derived, ARGS >](#)
Generic [Kobject](#) inheritance template.

Functions

- `template<typename T>`
`Type_info const * L4::kobject_typeid () noexcept`
Get the [L4::Type_info](#) for the [L4Re](#) interface given in T.

14.1.10.4.1 Detailed Description

Type information for [L4](#) server objects that can be called via IPC.

This type information consists of inheritance information, the protocol number assigned to an interface as well as the demand on server-side resources.

14.1.10.4.2 Function Documentation

14.1.10.4.2.1 kobject_typeid()

```
template<typename T>
Type\_info const * L4::kobject\_typeid () [inline], [noexcept]
```

Get the [L4::Type_info](#) for the [L4Re](#) interface given in T.

Template Parameters

<i>T</i>	The type (L4Re interface) for which the information shall be returned.
----------	---

Returns

A pointer to the [L4::Type_info](#) structure for T.

Definition at line 682 of file [__typeinfo.h](#).

References [L4::Kobject_typeid< T >::id\(\)](#).

Referenced by [cap_dynamic_cast\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**14.1.10.5 Factory**

C factory interface to create objects, see [L4::Factory](#) for the C++ interface.

Collaboration diagram for Factory:



Functions

- [l4_msgtag_t l4_factory_create_task](#) ([l4_cap_idx_t](#) factory, [l4_cap_idx_t](#) target_cap, [l4_fpage_t](#) *utcb_area) [L4_NOTHROW](#)
Create a new task.
- [l4_msgtag_t l4_factory_create_thread](#) ([l4_cap_idx_t](#) factory, [l4_cap_idx_t](#) target_cap) [L4_NOTHROW](#)
Create a new thread.
- [l4_msgtag_t l4_factory_create_factory](#) ([l4_cap_idx_t](#) factory, [l4_cap_idx_t](#) target_cap, unsigned long limit) [L4_NOTHROW](#)
Create a new factory.
- [l4_msgtag_t l4_factory_create_gate](#) ([l4_cap_idx_t](#) factory, [l4_cap_idx_t](#) target_cap, [l4_cap_idx_t](#) snd_dst_↔ cap, [l4_umword_t](#) label) [L4_NOTHROW](#)
Create a new IPC gate, optionally bound to a send destination (a thread or thread group).
- [l4_msgtag_t l4_factory_create_irq](#) ([l4_cap_idx_t](#) factory, [l4_cap_idx_t](#) target_cap) [L4_NOTHROW](#)
Create a new IRQ sender.
- [l4_msgtag_t l4_factory_create_vm](#) ([l4_cap_idx_t](#) factory, [l4_cap_idx_t](#) target_cap) [L4_NOTHROW](#)
Create a new virtual machine.
- [l4_msgtag_t l4_factory_create_vcpu_context](#) ([l4_cap_idx_t](#) factory, [l4_cap_idx_t](#) target_cap) [L4_NOTHROW](#)
Create a new vCPU context.
- [l4_msgtag_t l4_factory_create_thread_group](#) ([l4_cap_idx_t](#) factory, [l4_cap_idx_t](#) target_cap, unsigned policy) [L4_NOTHROW](#)
Create a new thread group.
- [l4_msgtag_t l4_factory_create](#) ([l4_cap_idx_t](#) factory, long obj, [l4_cap_idx_t](#) target) [L4_NOTHROW](#)
Create a new object.

14.1.10.5.1 Detailed Description

C factory interface to create objects, see [L4::Factory](#) for the C++ interface.

A factory is used to create all kinds of kernel objects:

- [Task](#)
- [Thread](#)
- [Factory](#)
- [IPC-Gate API](#)
- [IRQs](#)
- [Virtual Machines](#)

To create a new kernel object the caller has to specify the factory to use for creation. The caller has to allocate a capability slot where the kernel stores the new object's capability.

The factory is equipped with a limit that limits the amount of kernel memory available for that factory.

Note

The limit does not give any guarantee for the amount of available kernel memory.

Include File

```
#include <l4/sys/factory.h>
```

For the C++ interface refer to [L4::Factory](#).

14.1.10.5.2 Function Documentation

14.1.10.5.2.1 l4_factory_create()

```
l4_msgtag_t l4_factory_create (
    l4_cap_idx_t factory,
    long obj,
    l4_cap_idx_t target) [inline]
```

Create a new object.

Parameters

	<i>factory</i>	Factory to use for creation.
	<i>obj</i>	Protocol ID to describe the type of the object to create.
out	<i>target</i>	The kernel stores the new objects's capability into this slot.

Return values

<i>L4_EOK</i>	No error occurred.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i><0</i>	Error code.

Precondition

The capability *factory* must have the permission [L4_CAP_FPAGE_S](#).

Definition at line 707 of file [factory.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.5.2.2 l4_factory_create_factory()

```
l4_msgtag_t l4_factory_create_factory (
    l4_cap_idx_t factory,
    l4_cap_idx_t target_cap,
    unsigned long limit) [inline]
```

Create a new factory.

Parameters

	<i>factory</i>	Capability selector for factory to use for creation.
out	<i>target_cap</i>	The kernel stores the new factory's capability into this slot.
	<i>limit</i>	Limit for the new factory in bytes.

Returns

Syscall return tag

Return values

<i>L4_EOK</i>	No error occurred.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i><0</i>	Error code.

Precondition

The capability `factory` must have the permission [L4_CAP_FPAGE_S](#).

Note

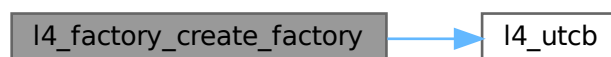
The limit of the new factory is subtracted from the available amount of the factory used for creation.

This method is only guaranteed to work with the [Kernel Factory](#). For other services, use the generic [L4::Factory::create\(\)](#) method and consult the service documentation for information on the arguments that need to be passed to the create stream.

Definition at line 545 of file [factory.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.5.2.3 l4_factory_create_gate()

```

l4_msgtag_t l4_factory_create_gate (
    l4_cap_idx_t factory,
    l4_cap_idx_t target_cap,
    l4_cap_idx_t snd_dst_cap,
    l4_umword_t label) [inline]
  
```

Create a new IPC gate, optionally bound to a send destination (a thread or thread group).

Parameters

	<i>factory</i>	Capability selector for factory to use for creation.
out	<i>target_cap</i>	The kernel stores the new IPC gate's capability into this slot.
	<i>snd_dst_cap</i>	Optional capability selector of a thread or thread group to bind the gate to. Use L4_INVALID_CAP to create an unbound IPC gate.
	<i>label</i>	Optional label of the gate (precisely used if <i>snd_dst_cap</i> is valid). If <i>snd_dst_cap</i> is valid, <i>label</i> must be present.

Returns

Syscall return tag containing one of the following return codes.

Return values

<i>L4_EOK</i>	No error occurred.
<i>-L4_ENOMEM</i>	Out-of-memory during allocation of the <i>lpc_gate</i> object.
<i>-L4_EINVAL</i>	<i>snd_dst_cap</i> is void or points to something that is not a thread or thread group.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.

Precondition

The capability *factory* must have the permission [L4_CAP_FPAGE_S](#). Also *snd_dst_cap* (if not [L4_INVALID_CAP](#)) must have the permission [L4_CAP_FPAGE_S](#).

An unbound IPC gate can be bound to a thread using [l4_rcv_ep_bind_thread\(\)](#).

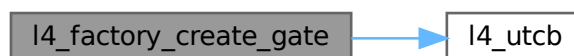
See also

[IPC-Gate API](#)

Definition at line 553 of file [factory.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.5.2.4 l4_factory_create_irq()

```

l4_msgtag_t l4_factory_create_irq (
    l4_cap_idx_t factory,
    l4_cap_idx_t target_cap) [inline]
  
```

Create a new IRQ sender.

Parameters

	<i>factory</i>	Factory to use for creation.
out	<i>target_cap</i>	The kernel stores the new IRQ's capability into this slot.

Return values

<i>L4_EOK</i>	No error occurred.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i><0</i>	Error code.

Precondition

The capability *factory* must have the permission [L4_CAP_FPAGE_S](#).

See also

[IRQs](#)

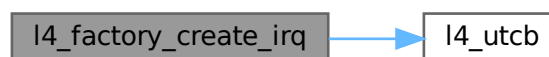
Examples

[examples/sys/isr/main.c](#).

Definition at line 561 of file [factory.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.5.2.5 l4_factory_create_task()

```

l4_msgtag_t l4_factory_create_task (
    l4_cap_idx_t factory,
    l4_cap_idx_t target_cap,
    l4_fpage_t * utcb_area) [inline]
  
```

Create a new task.

Parameters

	<i>factory</i>	Capability selector for factory to use for creation.
out	<i>target_cap</i>	The kernel stores the new task's capability into this slot.
in, out	<i>utcb_area</i>	Pointer to flexpage that describes an area of kernel-user memory that can be used for UTCBs and vCPU state-save-areas of the new task.

On systems without MMU, the flexpage is adjusted to reflect the actually allocated physical address.

Returns

Syscall return tag.

Return values

<i>L4_EOK</i>	No error occurred.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i><0</i>	Error code.

Precondition

The capability *factory* must have the permission [L4_CAP_FPAGE_S](#).

Note

The size of the UTCB area specifies indirectly the number of UTCBs available for this task. Refer to [l4_task_add_ku_mem\(\)](#) for adding more of this type of memory.

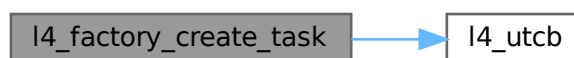
See also

[Task](#)

Definition at line 531 of file [factory.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.5.2.6 l4_factory_create_thread()

```

l4_msgtag_t l4_factory_create_thread (
    l4_cap_idx_t factory,
    l4_cap_idx_t target_cap) [inline]
  
```

Create a new thread.

Parameters

	<i>factory</i>	Capability selector for factory to use for creation.
out	<i>target_cap</i>	The kernel stores the new thread's capability into this slot.

Returns

Syscall return tag

Return values

<i>L4_EOK</i>	No error occurred.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i><0</i>	Error code.

Precondition

The capability *factory* must have the permission [L4_CAP_FPAGE_S](#).

See also

[Thread](#)

Examples

[examples/sys/aliens/main.c](#), [examples/sys/singlestep/main.c](#), [examples/sys/start-with-exc/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

Definition at line 538 of file [factory.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.5.2.7 l4_factory_create_thread_group()

```

l4_msgtag_t l4_factory_create_thread_group (
    l4_cap_idx_t factory,
    l4_cap_idx_t target_cap,
    unsigned policy) [inline]
  
```

Create a new thread group.

An IPC endpoint can be bound to a thread group. When a message arrives at the IPC endpoint, a specific thread of the thread group is selected to actually receive the message. A thread group is a send destination for an IPC endpoint.

Parameters

	<i>factory</i>	Capability selector for factory to use for creation.
out	<i>target_cap</i>	The kernel stores the new thread group's capability into this slot.
	<i>policy</i>	Policy parameter for the thread group. See #L4_thread_group_policy for a list of supported values.

Returns

Syscall return tag

Return values

<i>L4_EOK</i>	No error occurred.
<i>-L4_ENOMEM</i>	Out-of-memory during allocation of the thread group object.
<i>-L4_EINVAL</i>	Invalid policy parameter.
<i>-L4_EPERM</i>	The factory instance requires L4_CAP_FPAGE_S rights on <code>factory</code> and L4_CAP_FPAGE_S is not present.
<i><0</i>	Error code.

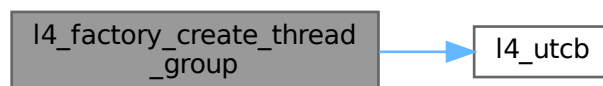
See also

[Thread groups](#)

Definition at line 582 of file [factory.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:

**14.1.10.5.2.8 l4_factory_create_vcpu_context()**

```

l4_msgtag_t l4_factory_create_vcpu_context (
    l4_cap_idx_t factory,
    l4_cap_idx_t target_cap) [inline]
  
```

Create a new vCPU context.

A vCPU context typically represents a hardware structure that captures the state of a vCPU on a CPU (e.g. VMX VMCS).

Parameters

	<i>factory</i>	Capability selector for factory to use for creation.
out	<i>target_cap</i>	The kernel stores the new vCPU context's capability into this slot.

Returns

Syscall return tag

Return values

<i>L4_EOK</i>	No error occurred.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i><0</i>	Error code.

Precondition

The capability *factory* must have the permission [L4_CAP_FPAGE_S](#).

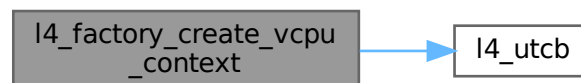
See also

[Virtual Machines](#)

Definition at line [575](#) of file [factory.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.5.2.9 l4_factory_create_vm()

```

l4_msgtag_t l4_factory_create_vm (
    l4_cap_idx_t factory,
    l4_cap_idx_t target_cap) [inline]
  
```

Create a new virtual machine.

Parameters

	<i>factory</i>	Capability selector for factory to use for creation.
out	<i>target_cap</i>	The kernel stores the new VM's capability into this slot.

Returns

Syscall return tag

Return values

<i>L4_EOK</i>	No error occurred.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i><0</i>	Error code.

Precondition

The capability *factory* must have the permission [L4_CAP_FPAGE_S](#).

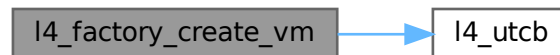
See also

[Virtual Machines](#)

Definition at line 568 of file [factory.h](#).

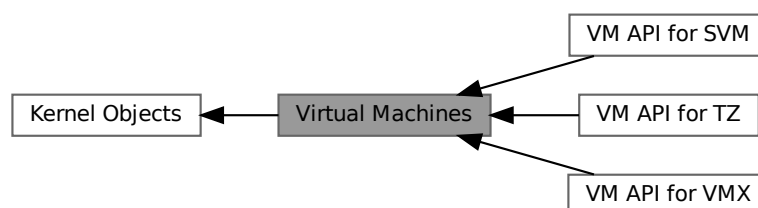
References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:

**14.1.10.6 Virtual Machines**

Virtual Machine API.

Collaboration diagram for Virtual Machines:



Topics

- VM API for SVM ??
Virtual machine API for SVM.
- VM API for VMX ??
Virtual machine API for VMX.
- VM API for TZ ??
Virtual Machine API for ARM TrustZone.

14.1.10.6.1 Detailed Description

Virtual Machine API.

14.1.10.6.2 VM API for SVM

Virtual machine API for SVM.

Collaboration diagram for VM API for SVM:



Data Structures

- struct [l4_vm_svm_vmcb_control_area](#)
VMCB structure for SVM VMs.
- struct [l4_vm_svm_vmcb_state_save_area_seg](#)
State save area segment selector struct.
- struct [l4_vm_svm_vmcb_state_save_area](#)
State save area structure for SVM VMs.
- struct [l4_vm_svm_vmcb_t](#)
Control structure for SVM VMs.

Typedefs

- typedef struct [l4_vm_svm_vmcb_control_area](#) [l4_vm_svm_vmcb_control_area_t](#)
VMCB structure for SVM VMs.
- typedef struct [l4_vm_svm_vmcb_state_save_area_seg](#) [l4_vm_svm_vmcb_state_save_area_seg_t](#)
State save area segment selector struct.
- typedef struct [l4_vm_svm_vmcb_state_save_area](#) [l4_vm_svm_vmcb_state_save_area_t](#)
State save area structure for SVM VMs.
- typedef struct [l4_vm_svm_vmcb_t](#) [l4_vm_svm_vmcb_t](#)
Control structure for SVM VMs.

14.1.10.6.2.1 Detailed Description

Virtual machine API for SVM.

14.1.10.6.3 VM API for VMX

Virtual machine API for VMX.

Collaboration diagram for VM API for VMX:



Data Structures

- struct [l4_vmx_offset_table_t](#)
Software VMCS field offset table.
- struct [l4_vm_vmx_vcpu_vmcs_t](#)
VMX software VMCS.
- struct [l4_vm_vmx_vcpu_infos_t](#)
VMX information members.
- struct [l4_vm_vmx_vcpu_state_t](#)
VMX vCPU state.

Typedefs

- typedef struct [l4_vmx_offset_table_t](#) [l4_vmx_offset_table_t](#)
Software VMCS field offset table.
- typedef struct [l4_vm_vmx_vcpu_vmcs_t](#) [l4_vm_vmx_vcpu_vmcs_t](#)
VMX software VMCS.
- typedef struct [l4_vm_vmx_vcpu_infos_t](#) [l4_vm_vmx_vcpu_infos_t](#)
VMX information members.
- typedef struct [l4_vm_vmx_vcpu_state_t](#) [l4_vm_vmx_vcpu_state_t](#)
VMX vCPU state.

Enumerations

- enum `L4_vm_vmx_caps_regs` {
`L4_VM_VMX_BASIC_REG` = 0 , `L4_VM_VMX_TRUE_PINBASED_CTLS_REG` = 1 , `L4_VM_VMX_TRUE_PROCBASED_CTL`
= 2 , `L4_VM_VMX_TRUE_EXIT_CTLS_REG` = 3 ,
`L4_VM_VMX_TRUE_ENTRY_CTLS_REG` = 4 , `L4_VM_VMX_MISC_REG` = 5 , `L4_VM_VMX_CR0_FIXED0_REG`
= 6 , `L4_VM_VMX_CR0_FIXED1_REG` = 7 ,
`L4_VM_VMX_CR4_FIXED0_REG` = 8 , `L4_VM_VMX_CR4_FIXED1_REG` = 9 , `L4_VM_VMX_VMCS_ENUM_REG`
= 10 , `L4_VM_VMX_PROCBASED_CTLS2_REG` = 11 ,
`L4_VM_VMX_EPT_VPID_CAP_REG` = 12 , `L4_VM_VMX_NESTED_REVISION` = 13 , `L4_VM_VMX_NUM_CAPS_REGS`
}

Exported VMX capability registers.

- enum `L4_vm_vmx_dfl1_regs` {
`L4_VM_VMX_PINBASED_CTLS_DFL1_REG` = 0 , `L4_VM_VMX_PROCBASED_CTLS_DFL1_REG` = 1 ,
`L4_VM_VMX_EXIT_CTLS_DFL1_REG` = 2 , `L4_VM_VMX_ENTRY_CTLS_DFL1_REG` = 3 ,
`L4_VM_VMX_NUM_DFL1_REGS` }

Exported VMX capability registers (default to 1 bits).

- enum `L4_vm_vmx_sw_fields` {
`L4_VM_VMX_VMCS_CR2` = 0x6880 , `L4_VM_VMX_VMCS_NAT_ARG0` = 0x6882 , `L4_VM_VMX_VMCS_NAT_ARG1`
= 0x6884 , `L4_VM_VMX_VMCS_NAT_ARG2` = 0x6886 ,
`L4_VM_VMX_VMCS_NAT_ARG3` = 0x6888 , `L4_VM_VMX_VMCS_XCR0` = 0x2880 , `L4_VM_VMX_VMCS_MSR_SYSCALL_M`
= 0x2882 , `L4_VM_VMX_VMCS_MSR_LSTAR` = 0x2884 ,
`L4_VM_VMX_VMCS_MSR_CSTAR` = 0x2886 , `L4_VM_VMX_VMCS_MSR_TSC_AUX` = 0x2888 ,
`L4_VM_VMX_VMCS_MSR_STAR` = 0x288a , `L4_VM_VMX_VMCS_MSR_KERNEL_GS_BASE` = 0x288c }

Additional (software-defined) VMCS fields.

- enum `L4_vm_vmx_vmcs_sizes` { `L4_VM_VMX_VMCS_SIZE_VALUES` = 2560 , `L4_VM_VMX_VMCS_SIZE_DIRTY_BITMAP`
= 320 }

Sizes of software VMCS members.

Functions

- `l4_uint64_t l4_vm_vmx_get_caps` (`l4_vm_vmx_vcpu_state_t` const *vcpu_state, enum `L4_vm_vmx_caps_regs`
caps_reg) `L4_NOTHROW`
Get a capability register for VMX.
- `l4_uint32_t l4_vm_vmx_get_caps_default1` (`l4_vm_vmx_vcpu_state_t` const *vcpu_state, enum `L4_vm_vmx_dfl1_regs`
dfl1_reg) `L4_NOTHROW`
Get a default to one capability register for VMX.
- unsigned `l4_vm_vmx_field_len` (unsigned field) `L4_NOTHROW`
Return length in bytes of a VMCS field.
- unsigned `l4_vm_vmx_field_order` (unsigned field) `L4_NOTHROW`
Return length in power of two (bytes) of a VMCS field.
- void `l4_vm_vmx_clear` (`l4_vm_vmx_vcpu_vmcs_t` *vmcs, `l4_vm_vmx_vcpu_vmcs_t` *dest_vmcs)
`L4_NOTHROW`
Save the content from the software VMCS to a different software VMCS.
- void `l4_vm_vmx_ptr_load` (`l4_vm_vmx_vcpu_vmcs_t` *vmcs, `l4_vm_vmx_vcpu_vmcs_t` *src_vmcs)
`L4_NOTHROW`
Load the content from a different software VMCS to the software VMCS.
- `l4_uint32_t l4_vm_vmx_get_cr2_index` (`l4_vm_vmx_vcpu_vmcs_t` const *vmcs) `L4_NOTHROW`
Get the software VMCS field index of the virtual CR2 register.
- `l4_umword_t l4_vm_vmx_read_nat` (`l4_vm_vmx_vcpu_vmcs_t` *vmcs, unsigned field) `L4_NOTHROW`
Read a natural-width software VMCS field.
- `l4_uint16_t l4_vm_vmx_read_16` (`l4_vm_vmx_vcpu_vmcs_t` *vmcs, unsigned field) `L4_NOTHROW`
Read a 16-bit software VMCS field.

- [l4_uint32_t l4_vm_vmx_read_32](#) ([l4_vm_vmx_vcpu_vmcs_t](#) *vmcs, unsigned field) [L4_NOTHROW](#)
Read a 32-bit software VMCS field.
- [l4_uint64_t l4_vm_vmx_read_64](#) ([l4_vm_vmx_vcpu_vmcs_t](#) *vmcs, unsigned field) [L4_NOTHROW](#)
Read a 64-bit software VMCS field.
- [l4_uint64_t l4_vm_vmx_read](#) ([l4_vm_vmx_vcpu_vmcs_t](#) *vmcs, unsigned field) [L4_NOTHROW](#)
Read any software VMCS field.
- void [l4_vm_vmx_write_nat](#) ([l4_vm_vmx_vcpu_vmcs_t](#) *vmcs, unsigned field, [l4_umword_t](#) val) [L4_NOTHROW](#)
Write to a natural-width software VMCS field.
- void [l4_vm_vmx_write_16](#) ([l4_vm_vmx_vcpu_vmcs_t](#) *vmcs, unsigned field, [l4_uint16_t](#) val) [L4_NOTHROW](#)
Write to a 16-bit software VMCS field.
- void [l4_vm_vmx_write_32](#) ([l4_vm_vmx_vcpu_vmcs_t](#) *vmcs, unsigned field, [l4_uint32_t](#) val) [L4_NOTHROW](#)
Write to a 32-bit software VMCS field.
- void [l4_vm_vmx_write_64](#) ([l4_vm_vmx_vcpu_vmcs_t](#) *vmcs, unsigned field, [l4_uint64_t](#) val) [L4_NOTHROW](#)
Write to a 64-bit software VMCS field.
- void [l4_vm_vmx_write](#) ([l4_vm_vmx_vcpu_vmcs_t](#) *vmcs, unsigned field, [l4_uint64_t](#) val) [L4_NOTHROW](#)
Write to an arbitrary software VMCS field.
- void [l4_vm_vmx_set_hw_vmcs](#) ([l4_vm_vmx_vcpu_vmcs_t](#) *vmcs, [l4_cap_idx_t](#) vmcs_cap) [L4_NOTHROW](#)
Associate the software VMCS with a vCPU context, i.e.
- [l4_cap_idx_t l4_vm_vmx_get_hw_vmcs](#) ([l4_vm_vmx_vcpu_vmcs_t](#) *vmcs) [L4_NOTHROW](#)
Get the vCPU context (i.e.

14.1.10.6.3.1 Detailed Description

Virtual machine API for VMX.

14.1.10.6.3.2 Typedef Documentation

[l4_vm_vmx_vcpu_state_t](#)

```
typedef struct l4_vm_vmx_vcpu_state_t l4_vm_vmx_vcpu_state_t
```

VMX vCPU state.

This is a specialization of the generic vCPU state for VMX. This data structure represents the following memory layout:

- 0x000 - 0x1ff: Standard vCPU state (with padding). See [l4_vcpu_state_t](#).
- 0x200 - 0x3ff: VMX information members (with padding). See [l4_vm_vmx_vcpu_infos_t](#).
- 0x400 - 0xffff: VMX software VMCS. See [l4_vm_vmx_vcpu_vmcs_t](#).

Note

The memory layout is documented here for reference purposes. However, the users are strongly discouraged from accessing the data structure directly. The API functions defined in this file are the preferred way of achieving the functionality.

l4_vm_vmx_vcpu_vmcs_t

```
typedef struct l4_vm_vmx_vcpu_vmcs_t l4_vm_vmx_vcpu_vmcs_t
```

VMX software VMCS.

This data structure represents the following memory layout:

- 0x000 - 0x007: Reserved (ignored by the kernel). In the hardware VMCS, the revision identifier and the abort indicator are stored in this area. Hereby we simply ignore these two entries.
- 0x008 - 0x00f: User space data (ignored by the kernel). This currently stores the pointer to a different software VMCS whose content has been loaded to this software VMCS.
- 0x010 - 0x013: VMCS field index of the software-defined CR2 field in the software VMCS.
- 0x014 - 0x017: Reserved.
- 0x018 - 0x01f: Capability of the vCPU context, i.e. the hardware VMCS object (with padding).
- 0x020 - 0x047: Software VMCS field offset table. See [l4_vmx_offset_table_t](#).
- 0x048 - 0x0bf: Reserved.
- 0x0c0 - 0xabf: Software VMCS fields (with padding).
- 0xac0 - 0xbff: Software VMCS fields dirty bitmap (with padding).

Note

The memory layout is documented here for reference purposes. However, the users are strongly discouraged from accessing the data structure directly. The API functions defined in this file are the preferred way of achieving the functionality.

l4_vmx_offset_table_t

```
typedef struct l4_vmx_offset_table_t l4_vmx_offset_table_t
```

Software VMCS field offset table.

This data structure represents the following memory layout:

- 0x00 - 0x02: 3 offsets for 16-bit fields.
- 0x03: Reserved.
- 0x04 - 0x06: 3 offsets for 64-bit fields.
- 0x07: Reserved.
- 0x08 - 0x0a: 3 offsets for 32-bit fields.
- 0x0b: Reserved.
- 0x0c - 0x0e: 3 offsets for natural-width fields.
- 0x0f: Reserved.
- 0x10 - 0x12: 3 limits for 16-bit fields.

- 0x13: Reserved.
- 0x14 - 0x16: 3 limits for 64-bit fields.
- 0x17: Reserved.
- 0x18 - 0x1a: 3 limits for 32-bit fields.
- 0x1b: Reserved.
- 0x1c - 0x1e: 3 limits for natural-width fields.
- 0x1f: Reserved.
- 0x20 - 0x23: 4 index shifts.
- 0x24: Offset of the first software VMCS field.
- 0x25: Size of the software VMCS fields.
- 0x26 - 0x27: Reserved.

The offsets/limits in each size category are in the following order:

- Control fields.
- Read-only fields.
- Guest fields.

The index shifts are in the following order:

- 16-bit.
- 64-bit.
- 32-bit.
- Natural-width.

All offsets/limits/sizes are represented in a 64-byte granule.

The offsets (after being multiplied by 64) are indexes in the values array in [l4_vm_vmx_vcpu_vmcs_t](#) and bit indexes in the dirty_bitmap array in [l4_vm_vmx_vcpu_vmcs_t](#).

The limits (after being multiplied by 64) represent the range of the available indexes.

Note

The memory layout is documented here for reference purposes. However, the users are strongly discouraged from accessing the data structure directly. The API functions defined in this file are the preferred way of achieving the functionality.

14.1.10.6.3.3 Enumeration Type Documentation

L4_vm_vmx_caps_regs

```
enum L4_vm_vmx_caps_regs
```

Exported VMX capability registers.

Enumerator

L4_VM_VMX_BASIC_REG	Basic VMX capabilities.
L4_VM_VMX_TRUE_PINBASED_CTLIS_REG	True pin-based control caps.
L4_VM_VMX_TRUE_PROCBASED_CTLIS_REG	True processor based control caps.
L4_VM_VMX_TRUE_EXIT_CTLIS_REG	True exit control caps.
L4_VM_VMX_TRUE_ENTRY_CTLIS_REG	True entry control caps.
L4_VM_VMX_MISC_REG	Misc caps.
L4_VM_VMX_CR0_FIXED0_REG	Fixed to 0 bits of CR0.
L4_VM_VMX_CR0_FIXED1_REG	Fixed to 1 bits of CR0.
L4_VM_VMX_CR4_FIXED0_REG	Fixed to 0 bits of CR4.
L4_VM_VMX_CR4_FIXED1_REG	Fixed to 1 bits of CR4.
L4_VM_VMX_VMCS_ENUM_REG	VMCS enumeration info.
L4_VM_VMX_PROCBASED_CTLIS2_REG	Processor based control 2 caps.
L4_VM_VMX_EPT_VPID_CAP_REG	EPT and VPID caps.
L4_VM_VMX_NESTED_REVISION	Nested VMCS revision.
L4_VM_VMX_NUM_CAPS_REGS	Total number of VMX capability registers.

Definition at line 28 of file [__vm-vmx.h](#).

L4_vm_vmx_dfl1_regs

```
enum L4_vm_vmx_dfl1_regs
```

Exported VMX capability registers (default to 1 bits).

Enumerator

L4_VM_VMX_PINBASED_CTLIS_DFL1_REG	Default 1 bits in pin-based controls.
L4_VM_VMX_PROCBASED_CTLIS_DFL1_REG	Default 1 bits in processor-based controls.
L4_VM_VMX_EXIT_CTLIS_DFL1_REG	Default 1 bits in exit controls.
L4_VM_VMX_ENTRY_CTLIS_DFL1_REG	Default 1 bits in entry controls.
L4_VM_VMX_NUM_DFL1_REGS	Total number of default on registers.

Definition at line 51 of file [__vm-vmx.h](#).

L4_vm_vmx_sw_fields

```
enum L4_vm_vmx_sw_fields
```

Additional (software-defined) VMCS fields.

The VMCS offsets defined here are actually not in the hardware VMCS. However our VMMs run in user mode and need to have access to certain registers available in kernel mode only. So we put them into our software VMCS.

Enumerator

L4_VM_VMX_VMCS_CR2	Software VMCS offset for CR2. Note You usually need to check this value against the value you get from <code>l4_vm_vmx_get_cr2_index()</code> to make sure you are running on a compatible kernel.
L4_VM_VMX_VMCS_NAT_ARG0	Custom argument passed from kernel to user space.
L4_VM_VMX_VMCS_NAT_ARG1	Custom argument passed from kernel to user space.
L4_VM_VMX_VMCS_NAT_ARG2	Custom argument passed from kernel to user space.
L4_VM_VMX_VMCS_NAT_ARG3	Custom argument passed from kernel to user space.
L4_VM_VMX_VMCS_XCR0	VMCS offset of extended control register XCR0.
L4_VM_VMX_VMCS_MSR_SYSCALL_MASK	VMCS offset of system call flag mask MSR.
L4_VM_VMX_VMCS_MSR_LSTAR	VMCS offset of IA32e mode system call target address MSR.
L4_VM_VMX_VMCS_MSR_CSTAR	VMCS offset of IA32 mode system call target address MSR.
L4_VM_VMX_VMCS_MSR_TSC_AUX	VMCS offset of auxiliary TSC signature MSR.
L4_VM_VMX_VMCS_MSR_STAR	VMCS offset of system call target address MSR.
L4_VM_VMX_VMCS_MSR_KERNEL_GS_BASE	VMCS offset of GS base address swap target MSR.

Definition at line 69 of file [__vm-vmx.h](#).

L4_vm_vmx_vmcs_sizes

```
enum L4_vm_vmx_vmcs_sizes
```

Sizes of software VMCS members.

Enumerator

L4_VM_VMX_VMCS_SIZE_VALUES	Size of the software VMCS values member.
L4_VM_VMX_VMCS_SIZE_DIRTY_BITMAP	Size of the software VMCS dirty bitmap member.

Definition at line 170 of file [__vm-vmx.h](#).

14.1.10.6.3.4 Function Documentation

l4_vm_vmx_clear()

```
void l4_vm_vmx_clear (
    l4_vm_vmx_vcpu_vmcs_t * vmcs,
    l4_vm_vmx_vcpu_vmcs_t * dest_vmcs) [inline]
```

Save the content from the software VMCS to a different software VMCS.

Parameters

<i>vmcs</i>	Pointer to the source software VMCS.
<i>dest_vmcs</i>	Pointer to the destination software VMCS.

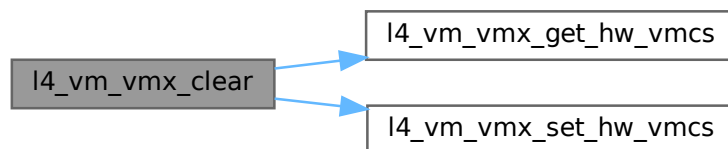
This function is comparable to the VMX VMCLEAR instruction.

Definition at line 698 of file [__vm-vmx.h](#).

References [L4_NOTHROW](#), [l4_vm_vmx_get_hw_vmcs\(\)](#), [l4_vm_vmx_set_hw_vmcs\(\)](#), and [L4_VM_VMX_VMCS_SIZE_DIRTY_BITM](#)

Referenced by [l4_vm_vmx_ptr_load\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



`l4_vm_vmx_field_len()`

```

unsigned l4_vm_vmx_field_len (
    unsigned field) [inline]
  
```

Return length in bytes of a VMCS field.

Parameters

<i>field</i>	Field number.
--------------	---------------

Returns

Width of field in bytes.

Definition at line 593 of file [__vm-vmx.h](#).

References [L4_NOTHROW](#), and [l4_vm_vmx_field_order\(\)](#).

Here is the call graph for this function:

**`l4_vm_vmx_field_order()`**

```
unsigned l4_vm_vmx_field_order (  
    unsigned field) [inline]
```

Return length in power of two (bytes) of a VMCS field.

Parameters

<i>field</i>	Field number.
--------------	---------------

Returns

Width of field in power of two (bytes).

Definition at line 600 of file [__vm-vmx.h](#).

References [L4_NOTHROW](#).

Referenced by [l4_vm_vmx_field_len\(\)](#).

Here is the caller graph for this function:



l4_vm_vmx_get_caps()

```
l4_uint64_t l4_vm_vmx_get_caps (
    l4_vm_vmx_vcpu_state_t const * vcpu_state,
    enum L4_vm_vmx_caps_regs caps_reg) [inline]
```

Get a capability register for VMX.

Parameters

<i>vcpu_state</i>	Pointer to the vCPU state.
<i>caps_reg</i>	Capability register index (see L4_vm_vmx_caps_regs).

Returns

The value of the capability register.

Definition at line 884 of file [__vm-vmx.h](#).

References [L4_NOTHROW](#).

l4_vm_vmx_get_caps_default1()

```
l4_uint32_t l4_vm_vmx_get_caps_default1 (
    l4_vm_vmx_vcpu_state_t const * vcpu_state,
    enum L4_vm_vmx_dfl1_regs dfl1_reg) [inline]
```

Get a default to one capability register for VMX.

Parameters

<i>vcpu_state</i>	Pointer to the vCPU state.
<i>dfl1_reg</i>	Default to 1 capability register index (see L4_vm_vmx_dfl1_regs).

Returns

The value of the capability register.

Definition at line 892 of file [__vm-vmx.h](#).

References [L4_NOTHROW](#).

l4_vm_vmx_get_cr2_index()

```
l4_uint32_t l4_vm_vmx_get_cr2_index (
    l4_vm_vmx_vcpu_vmcs_t const * vmcs) [inline]
```

Get the software VMCS field index of the virtual CR2 register.

Parameters

<i>vmcs</i>	Pointer to the software VMCS.
-------------	-------------------------------

Returns

The field index used for the virtual CR2 register as used by the current Fiasco.OC interface.

The CR2 register is actually not in the hardware VMCS, however our VMMs run in user mode and need to have access to this register so we put it into our software VMCS.

See also

[L4_VM_VMX_VMCS_CR2](#)

Definition at line 900 of file [__vm-vmx.h](#).

References [L4_NOTHROW](#).

l4_vm_vmx_get_hw_vmcs()

```
l4_cap_idx_t l4_vm_vmx_get_hw_vmcs (
    l4_vm_vmx_vcpu_vmcs_t * vmcs) [inline]
```

Get the vCPU context (i.e.

the hardware VMCS object) associated with the software VMCS.

Parameters

<i>vmcs</i>	Pointer to the software VMCS.
-------------	-------------------------------

Returns

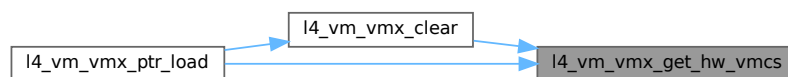
vCPU context (hardware VMCS object) capability.

Definition at line 915 of file [__vm-vmx.h](#).

References [L4_CAP_MASK](#), and [L4_NOTHROW](#).

Referenced by [l4_vm_vmx_clear\(\)](#), and [l4_vm_vmx_ptr_load\(\)](#).

Here is the caller graph for this function:



l4_vm_vmx_ptr_load()

```
void l4_vm_vmx_ptr_load (
    l4_vm_vmx_vcpu_vmcs_t * vmcs,
    l4_vm_vmx_vcpu_vmcs_t * src_vmcs) [inline]
```

Load the content from a different software VMCS to the software VMCS.

Parameters

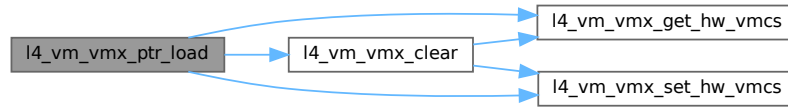
<i>vmcs</i>	Pointer to the destination software VMCS.
<i>src_vmcs</i>	Pointer to the source software VMCS.

This function is comparable to the VMX VMPTRLD instruction.

Definition at line 719 of file [__vm-vmx.h](#).

References [L4_NOTHROW](#), [l4_vm_vmx_clear\(\)](#), [l4_vm_vmx_get_hw_vmcs\(\)](#), [l4_vm_vmx_set_hw_vmcs\(\)](#), and [L4_VM_VMX_VMCS_SIZE_DIRTY_BITMAP](#).

Here is the call graph for this function:



`l4_vm_vmx_read()`

```

l4_uint64_t l4_vm_vmx_read (
    l4_vm_vmx_vcpu_vmcs_t * vmcs,
    unsigned field) [inline]
  
```

Read any software VMCS field.

Parameters

<i>vmcs</i>	Pointer to the software VMCS.
<i>field</i>	The VMCS field index as used on VMX hardware.

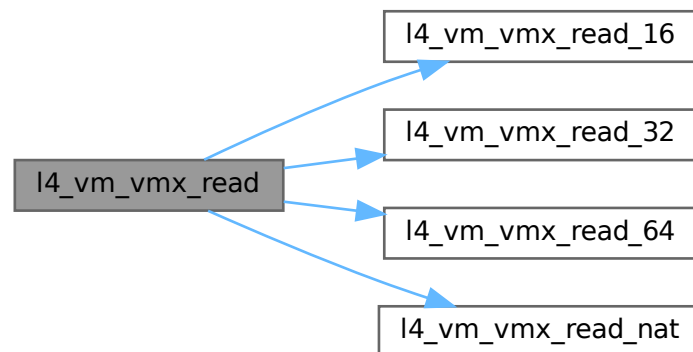
Returns

The value of the software VMCS field with the given index.

Definition at line 787 of file `__vm-vmx.h`.

References [L4_NOTHROW](#), [l4_vm_vmx_read_16\(\)](#), [l4_vm_vmx_read_32\(\)](#), [l4_vm_vmx_read_64\(\)](#), and [l4_vm_vmx_read_nat\(\)](#).

Here is the call graph for this function:



l4_vm_vmx_read_16()

```
l4_uint16_t l4_vm_vmx_read_16 (  
    l4_vm_vmx_vcpu_vmcs_t * vmcs,  
    unsigned field) [inline]
```

Read a 16-bit software VMCS field.

Parameters

<i>vmcs</i>	Pointer to the software VMCS.
<i>field</i>	The VMCS field index as used on VMX hardware.

Returns

The value of the software VMCS field with the given index.

Definition at line 754 of file [__vm-vmx.h](#).

References [L4_NOTHROW](#).

Referenced by [l4_vm_vmx_read\(\)](#).

Here is the caller graph for this function:



l4_vm_vmx_read_32()

```
l4_uint32_t l4_vm_vmx_read_32 (  
    l4_vm_vmx_vcpu_vmcs_t * vmcs,  
    unsigned field) [inline]
```

Read a 32-bit software VMCS field.

Parameters

<i>vmcs</i>	Pointer to the software VMCS.
<i>field</i>	The VMCS field index as used on VMX hardware.

Returns

The value of the software VMCS field with the given index.

Definition at line 765 of file [__vm-vmx.h](#).

References [L4_NOTHROW](#).

Referenced by [l4_vm_vmx_read\(\)](#).

Here is the caller graph for this function:

**l4_vm_vmx_read_64()**

```
l4_uint64_t l4_vm_vmx_read_64 (  
    l4_vm_vmx_vcpu_vmcs_t * vmcs,  
    unsigned field) [inline]
```

Read a 64-bit software VMCS field.

Parameters

<i>vmcs</i>	Pointer to the software VMCS.
<i>field</i>	The VMCS field index as used on VMX hardware.

Returns

The value of the software VMCS field with the given index.

Definition at line 776 of file [__vm-vmx.h](#).

References [L4_NOTHROW](#).

Referenced by [l4_vm_vmx_read\(\)](#).

Here is the caller graph for this function:



l4_vm_vmx_read_nat()

```
l4_umword_t l4_vm_vmx_read_nat (
    l4_vm_vmx_vcpu_vmcs_t * vmcs,
    unsigned field) [inline]
```

Read a natural-width software VMCS field.

Parameters

<i>vmcs</i>	Pointer to the software VMCS.
<i>field</i>	The VMCS field index as used on VMX hardware.

Returns

The value of the software VMCS field with the given index.

Definition at line 743 of file [__vm-vmx.h](#).

References [L4_NOTHROW](#).

Referenced by [l4_vm_vmx_read\(\)](#).

Here is the caller graph for this function:

**l4_vm_vmx_set_hw_vmcs()**

```
void l4_vm_vmx_set_hw_vmcs (
    l4_vm_vmx_vcpu_vmcs_t * vmcs,
    l4_cap_idx_t vmcs_cap) [inline]
```

Associate the software VMCS with a vCPU context, i.e.

a hardware VMCS object.

The VMX extended vCPU state is unable to be resumed unless it is associated with a vCPU context, i.e. a hardware VMCS object: An `L4::Vcpu_context` from the user space point of view with its kernel counterpart `Vmx_vmcs`.

Note

When replacing the vCPU context, the dirty bitmap of the software VMCS is not touched, neither by the kernel nor by the API functions. This is on purpose, to enable efficient switching between separate VMs in the common case. If there is a logical discrepancy between the content of the software VMCS and the replaced vCPU context, the user is responsible for explicitly setting the relevant software VMCS fields and/or the relevant software VMCS dirty bitmap bits to ensure that the discrepancy is rectified on the next vCPU resume. This needs to be done regardless of using the API functions (the preferred way) or accessing the data structures directly (the discouraged way).

Replacing the vCPU context while the vCPU is currently running has no immediate effect until the next vCPU resume. In addition to that, the kernel might cache the vCPU context internally (in other words, the capability is not looked up on every vCPU resume). To remove the association of the current vCPU context, simply replace it by another vCPU context. The reference count of the previous vCPU context will be decremented accordingly on the next vCPU resume.

To remove the association of the current vCPU context without replacing it by another vCPU context, pass an invalid capability with the bit 3 set and trigger a vCPU resume. The vCPU resume will fail in this case (due to the missing vCPU context), but the reference count of the previous vCPU context will be decremented accordingly.

There is no need to explicitly remove the association of the current vCPU context before deleting the software VMCS. Deleting the software VMCS automatically disassociates it from the vCPU context and a vCPU context with a reference count of 0 will be eventually deleted as well.

If the hardware limitations on the usage of the vCPU context are not observed (i.e. no hardware VMCS being active on more than one physical CPU), the vCPU will fail to resume.

Parameters

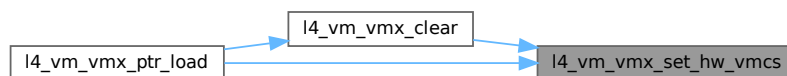
<i>vmcs</i>	Pointer to the software VMCS.
<i>vmcs_cap</i>	vCPU context (hardware VMCS object) capability.

Definition at line 907 of file [__vm-vmx.h](#).

References [L4_NOTHROW](#).

Referenced by [l4_vm_vmx_clear\(\)](#), and [l4_vm_vmx_ptr_load\(\)](#).

Here is the caller graph for this function:

**`l4_vm_vmx_write()`**

```
void l4_vm_vmx_write (
    l4_vm_vmx_vcpu_vmcs_t * vmcs,
    unsigned field,
    l4_uint64_t val) [inline]
```

Write to an arbitrary software VMCS field.

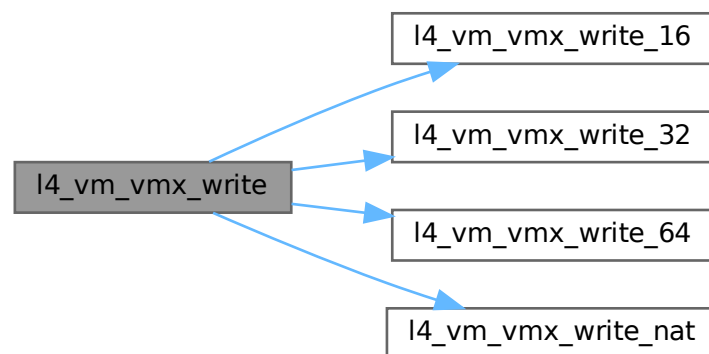
Parameters

<i>vmcs</i>	Pointer to the software VMCS.
<i>field</i>	The VMCS field index as used on VMX hardware.
<i>val</i>	The value that shall be written to the given field.

Definition at line 868 of file [__vm-vmx.h](#).

References [L4_NOTHROW](#), [l4_vm_vmx_write_16\(\)](#), [l4_vm_vmx_write_32\(\)](#), [l4_vm_vmx_write_64\(\)](#), and [l4_vm_vmx_write_nat\(\)](#).

Here is the call graph for this function:



`l4_vm_vmx_write_16()`

```
void l4_vm_vmx_write_16 (
    l4_vm_vmx_vcpu_vmcs_t * vmcs,
    unsigned field,
    l4_uint16_t val) [inline]
```

Write to a 16-bit software VMCS field.

Parameters

<i>vmcs</i>	Pointer to the software VMCS.
<i>field</i>	The VMCS field index as used on VMX hardware.
<i>val</i>	The value that shall be written to the given field.

Definition at line 820 of file [__vm-vmx.h](#).

References [L4_NOTHROW](#).

Referenced by [l4_vm_vmx_write\(\)](#).

Here is the caller graph for this function:



l4_vm_vmx_write_32()

```

void l4_vm_vmx_write_32 (
    l4_vm_vmx_vcpu_vmcs_t * vmcs,
    unsigned field,
    l4_uint32_t val) [inline]
  
```

Write to a 32-bit software VMCS field.

Parameters

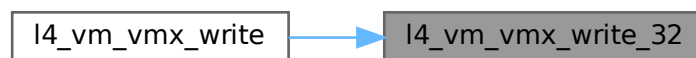
<i>vmcs</i>	Pointer to the software VMCS.
<i>field</i>	The VMCS field index as used on VMX hardware.
<i>val</i>	The value that shall be written to the given field.

Definition at line 836 of file [__vm-vmx.h](#).

References [L4_NOTHROW](#).

Referenced by [l4_vm_vmx_write\(\)](#).

Here is the caller graph for this function:



l4_vm_vmx_write_64()

```

void l4_vm_vmx_write_64 (
    l4_vm_vmx_vcpu_vmcs_t * vmcs,
    unsigned field,
    l4_uint64_t val) [inline]
  
```

Write to a 64-bit software VMCS field.

Parameters

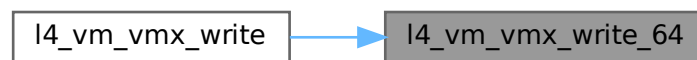
<i>vmcs</i>	Pointer to the software VMCS.
<i>field</i>	The VMCS field index as used on VMX hardware.
<i>val</i>	The value that shall be written to the given field.

Definition at line 852 of file [__vm-vmx.h](#).

References [L4_NOTHROW](#).

Referenced by [l4_vm_vmx_write\(\)](#).

Here is the caller graph for this function:



`l4_vm_vmx_write_nat()`

```

void l4_vm_vmx_write_nat (
    l4_vm_vmx_vcpu_vmcs_t * vmcs,
    unsigned field,
    l4_umword_t val) [inline]
  
```

Write to a natural-width software VMCS field.

Parameters

<i>vmcs</i>	Pointer to the software VMCS.
<i>field</i>	The VMCS field index as used on VMX hardware.
<i>val</i>	The value that shall be written to the given field.

Definition at line 804 of file [__vm-vmx.h](#).

References [L4_NOTHROW](#).

Referenced by [l4_vm_vmx_write\(\)](#).

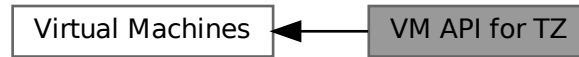
Here is the caller graph for this function:



14.1.10.6.4 VM API for TZ

Virtual Machine API for ARM TrustZone.

Collaboration diagram for VM API for TZ:



Data Structures

- struct [l4_vm_tz_state](#)
state structure for TrustZone VMs

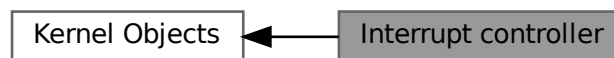
14.1.10.6.4.1 Detailed Description

Virtual Machine API for ARM TrustZone.

14.1.10.7 Interrupt controller

The C Icu interface, see [L4::Icu](#) for the C++ interface.

Collaboration diagram for Interrupt controller:



Data Structures

- struct [l4_icu_info_t](#)
Info structure for an ICU.

Typedefs

- typedef struct l4_icu_info_t [l4_icu_info_t](#)
Info structure for an ICU.

Enumerations

- enum `L4_icu_flags` { `L4_ICU_FLAG_MSI` }
Flags for IRQ numbers used for the ICU.

Functions

- `l4_msgtag_t l4_icu_bind (l4_cap_idx_t icu, unsigned irqnum, l4_cap_idx_t irq) L4_NOTHROW`
Bind an interrupt line of an interrupt controller to an interrupt object.
- `l4_msgtag_t l4_icu_bind_u (l4_cap_idx_t icu, unsigned irqnum, l4_cap_idx_t irq, l4_utcb_t *utcb) L4_NOTHROW`
Bind an interrupt line of an interrupt controller to an interrupt object.
- `l4_msgtag_t l4_icu_unbind (l4_cap_idx_t icu, unsigned irqnum, l4_cap_idx_t irq) L4_NOTHROW`
Remove binding of an interrupt line from the interrupt controller object.
- `l4_msgtag_t l4_icu_unbind_u (l4_cap_idx_t icu, unsigned irqnum, l4_cap_idx_t irq, l4_utcb_t *utcb) L4_NOTHROW`
Remove binding of an interrupt line from the interrupt controller object.
- `l4_msgtag_t l4_icu_set_mode (l4_cap_idx_t icu, unsigned irqnum, l4_umword_t mode) L4_NOTHROW`
Set interrupt mode.
- `l4_msgtag_t l4_icu_set_mode_u (l4_cap_idx_t icu, unsigned irqnum, l4_umword_t mode, l4_utcb_t *utcb) L4_NOTHROW`
Set interrupt mode.
- `l4_msgtag_t l4_icu_info (l4_cap_idx_t icu, l4_icu_info_t *info) L4_NOTHROW`
Get information about the ICU features.
- `l4_msgtag_t l4_icu_info_u (l4_cap_idx_t icu, l4_icu_info_t *info, l4_utcb_t *utcb) L4_NOTHROW`
Get information about the ICU features.
- `l4_msgtag_t l4_icu_msi_info (l4_cap_idx_t icu, unsigned irqnum, l4_uint64_t source, l4_icu_msi_info_t *msi_info) L4_NOTHROW`
Get MSI info about IRQ.
- `l4_msgtag_t l4_icu_msi_info_u (l4_cap_idx_t icu, unsigned irqnum, l4_uint64_t source, l4_icu_msi_info_t *msi_info, l4_utcb_t *utcb) L4_NOTHROW`
Get MSI info about IRQ.
- `l4_msgtag_t l4_icu_unmask (l4_cap_idx_t icu, unsigned irqnum, l4_umword_t *label, l4_timeout_t to) L4_NOTHROW`
Unmask an IRQ line.
- `l4_msgtag_t l4_icu_unmask_u (l4_cap_idx_t icu, unsigned irqnum, l4_umword_t *label, l4_timeout_t to, l4_utcb_t *utcb) L4_NOTHROW`
Unmask the given interrupt line.
- `l4_msgtag_t l4_icu_mask (l4_cap_idx_t icu, unsigned irqnum, l4_umword_t *label, l4_timeout_t to) L4_NOTHROW`
Mask an IRQ line.
- `l4_msgtag_t l4_icu_mask_u (l4_cap_idx_t icu, unsigned irqnum, l4_umword_t *label, l4_timeout_t to, l4_utcb_t *utcb) L4_NOTHROW`
Mask an IRQ line.

14.1.10.7.1 Detailed Description

The C Icu interface, see [L4::Icu](#) for the C++ interface.

Note

"ICU" is short for "interrupt control unit".

These functions define the interface for interrupt controllers, for binding IRQ objects to interrupt lines and other interrupt sources, as well as functions for masking and unmasking of interrupts.

To setup an IRQ line the following steps are required:

1. [l4_icu_set_mode\(\)](#) (optional if IRQ has a default mode)
2. [l4_rcv_ep_bind_thread\(\)](#) or [l4_rcv_ep_bind_snd_destination\(\)](#) to attach the IRQ object to a thread object.
3. [l4_icu_bind\(\)](#)
4. [l4_icu_unmask\(\)](#) to receive the first IRQ

For certain interrupt sources only some of these steps are necessary and supported, see [Scheduler](#) and [Virtual Console](#).

At most one [IRQs](#) object can be bound to a certain interrupt source and a certain [IRQs](#) object can be bound to at most one interrupt source.

Include File

```
#include <l4/sys/icu.h>
```

14.1.10.7.2 Typedef Documentation

14.1.10.7.2.1 l4_icu_info_t

```
typedef struct l4_icu_info_t l4_icu_info_t
```

Info structure for an ICU.

This structure contains information about the features of an ICU.

See also

[l4_icu_info\(\)](#).

14.1.10.7.3 Enumeration Type Documentation

14.1.10.7.3.1 L4_icu_flags

```
enum L4_icu_flags
```

Flags for IRQ numbers used for the ICU.

Enumerator

<code>L4_ICU_FLAG_MSI</code>	Flag to denote that the IRQ is actually an MSI. This flag may be used for l4_icu_bind() and l4_icu_unbind() functions to denote that the IRQ number is meant to be an MSI.
------------------------------	--

Definition at line 53 of file [icu.h](#).

14.1.10.7.4 Function Documentation

14.1.10.7.4.1 l4_icu_bind()

```
l4_msgtag_t l4_icu_bind (
    l4_cap_idx_t icu,
    unsigned irqnum,
    l4_cap_idx_t irq) [inline]
```

Bind an interrupt line of an interrupt controller to an interrupt object.

Parameters

<i>icu</i>	ICU object to bind <i>irq</i> to.
<i>irqnum</i>	IRQ line at the ICU.
<i>irq</i>	IRQ object to bind to this ICU.

Returns

Syscall return tag. The caller should check the return value using [l4_error\(\)](#) to check for errors and to identify the correct method for unmasking the interrupt. Return values < 0 indicate an error. A return value of 0 means a direct unmask via the IRQ object using [l4_irq_unmask\(\)](#). A return value of 1 means that the interrupt has to be unmasked via the ICU using [l4_icu_unmask\(\)](#).

Return values

<code>-L4_EINVAL</code>	<i>irq</i> is bound to an interrupt source.
<code>-L4_EPERM</code>	Insufficient permissions; see precondition.

Precondition

The capability *irq* must have the permission [L4_CAP_FPAGE_W](#).

In case the *irq* is already bound to an interrupt source, it is unbound first. In case the *irq* is bound and the interrupt source is bound to a different IRQ object, only the unbinding happens. An IRQ object that is bound to an interrupt source will get unbound if the IRQ object is deleted.

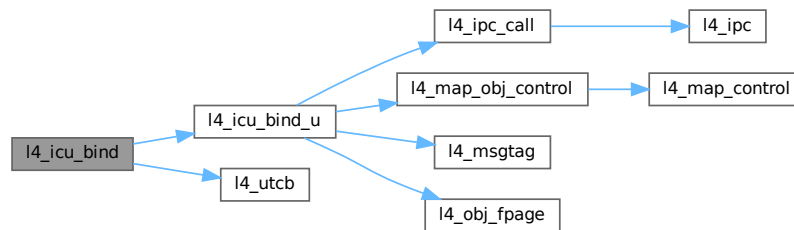
Examples

[examples/sys/isr/main.c](#).

Definition at line 496 of file [icu.h](#).

References [l4_icu_bind_u\(\)](#), [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.7.4.2 l4_icu_bind_u()

```

l4_msgtag_t l4_icu_bind_u (
    l4_cap_idx_t icu,
    unsigned irqnum,
    l4_cap_idx_t irq,
    l4_utcb_t * utcb) [inline]
  
```

Bind an interrupt line of an interrupt controller to an interrupt object.

Parameters

<i>icu</i>	The ICU object to bind <i>irq</i> to.
<i>irqnum</i>	IRQ line at the ICU.
<i>irq</i>	IRQ object for the given IRQ line to bind to this ICU.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Returns

Syscall return tag. The caller should check the return value using [l4_error\(\)](#) to check for errors and to identify the correct method for unmasking the interrupt. Return values < 0 indicate an error. A return value of 0 means a direct unmask via the IRQ object using [L4::irq::unmask](#). A return value of 1 means that the interrupt has to be unmasked via the ICU using [L4::icu::unmask](#).

Return values

<i>-L4_EINVAL</i>	<i>irq</i> is bound to an interrupt source.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.

Precondition

The capability `irq` must have the permission `L4_CAP_FPAGE_W`.

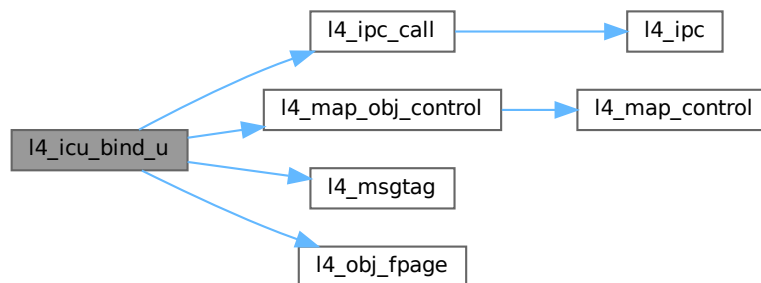
In case the `irq` is already bound to an interrupt source, it is unbound first. In case the `irq` is bound and the interrupt source is bound to a different `L4::lrq` object, only the unbinding happens. An `L4::lrq` object that is bound to an interrupt source will get unbound if the `L4::lrq` object is deleted.

Definition at line 396 of file `icu.h`.

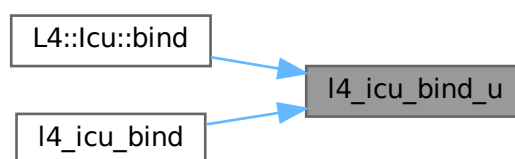
References `L4_CAP_FPAGE_RWS`, `L4_ICU_OP_BIND`, `l4_ipc_call()`, `L4_IPC_NEVER`, `l4_map_obj_control()`, `l4_msgtag()`, `L4_NOTHROW`, `l4_obj_fpage()`, `L4_PROTO_IRQ`, `l4_msg_regs_t::mr`, and `l4_fpage_t::raw`.

Referenced by `L4::lcu::bind()`, and `l4_icu_bind()`.

Here is the call graph for this function:



Here is the caller graph for this function:

**14.1.10.7.4.3 l4_icu_info()**

```

l4_msgtag_t l4_icu_info (
    l4_cap_idx_t icu,
    l4_icu_info_t * info) [inline]
  
```

Get information about the ICU features.

Parameters

	<i>icu</i>	The ICU object from which information shall be retrieved.
out	<i>info</i>	Info structure to be filled with information.

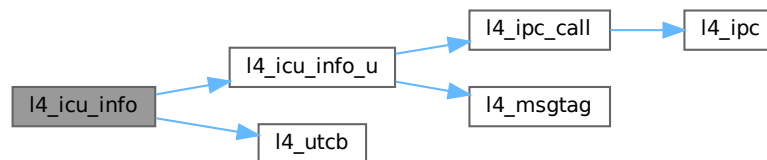
Returns

Syscall return tag

Definition at line 504 of file [icu.h](#).

References [l4_icu_info_u\(\)](#), [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.7.4.4 l4_icu_info_u()

```

l4_msgtag_t l4_icu_info_u (
    l4_cap_idx_t icu,
    l4_icu_info_t * info,
    l4_utcb_t * utcb) [inline]
  
```

Get information about the ICU features.

Parameters

	<i>icu</i>	The ICU object from which MSI information shall be retrieved.
out	<i>info</i>	Info structure to be filled with information.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Returns

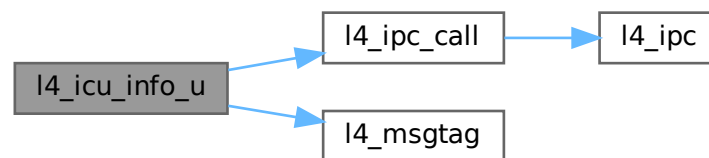
Syscall return tag

Definition at line 420 of file [icu.h](#).

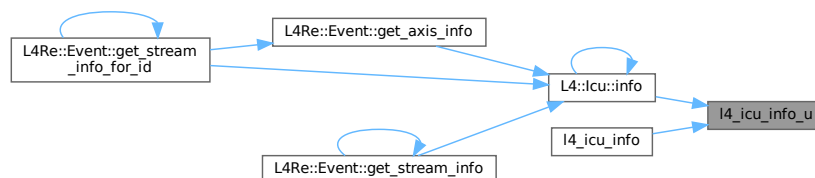
References [L4_ICU_OP_INFO](#), [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), [l4_msgtag\(\)](#), [L4_NOTHROW](#), [L4_PROTO_IRQ](#), and [l4_msg_regs_t::mr](#).

Referenced by [L4::l4u::info\(\)](#), and [l4_icu_info\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**14.1.10.7.4.5 l4_icu_mask()**

```

l4_msgtag_t l4_icu_mask (
    l4_cap_idx_t icu,
    unsigned irqnum,
    l4_umword_t * label,
    l4_timeout_t to) [inline]
  
```

Mask an IRQ line.

Parameters

<i>icu</i>	The ICU object where the IRQ line shall be masked.
------------	--

<i>irqnum</i>	IRQ line at the ICU.
<i>label</i>	If non-NULL, the function also performs an open wait IPC operation waiting for the next message, and the received label is returned here.
<i>to</i>	IPC timeout, if unsure use L4_IPC_NEVER.

Returns

Syscall return tag. If *label* is NULL, this function performs an IPC send-only operation and there is no return value except [L4_MSGTAG_ERROR](#) indicating success or failure of the send operation. In this case use [l4_ipc_error\(\)](#) to check for errors and **do not** use [l4_error\(\)](#).

Definition at line 518 of file [icu.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.7.4.6 l4_icu_mask_u()

```

l4_msgtag_t l4_icu_mask_u (
    l4_cap_idx_t icu,
    unsigned irqnum,
    l4_umword_t * label,
    l4_timeout_t to,
    l4_utcb_t * utcb) [inline]
  
```

Mask an IRQ line.

Parameters

<i>icu</i>	The ICU object where the IRQ line shall be masked.
<i>irqnum</i>	IRQ line at the ICU.
<i>label</i>	If NULL, this function is a send-only message to the ICU. If not NULL, this function will enter an open wait after sending the mask message and the received label is returned here.
<i>to</i>	The timeout-pair (send and receive) that shall be used for this operation. The receive timeout is used with a non-NULL <i>label</i> only.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Returns

Syscall return tag. If `label` is `NULL`, this function performs an IPC send-only operation and there is no return value except `L4_MSGTAG_ERROR` indicating success or failure of the send operation. In this case use `l4_ipc_error()` to check for errors and **do not** use `l4_error()`.

Definition at line 483 of file `icu.h`.

References `L4_NOTHROW`.

Referenced by `L4::l4cu::mask()`.

Here is the caller graph for this function:

**14.1.10.7.4.7 l4_icu_msi_info()**

```

l4_msgtag_t l4_icu_msi_info (
    l4_cap_idx_t icu,
    unsigned irqnum,
    l4_uint64_t source,
    l4_icu_msi_info_t * msi_info) [inline]
  
```

Get MSI info about IRQ.

Parameters

	<i>icu</i>	The ICU object from which MSI information shall be retrieved.
	<i>irqnum</i>	IRQ line at the ICU.
	<i>source</i>	Platform dependent requester ID for MSIs. On IA32 we use a 20bit source filter value as described in the Intel IRQ remapping specification.
out	<i>msi_info</i>	A <code>l4_icu_msi_info_t</code> structure receiving the address and the data value to trigger this MSI.

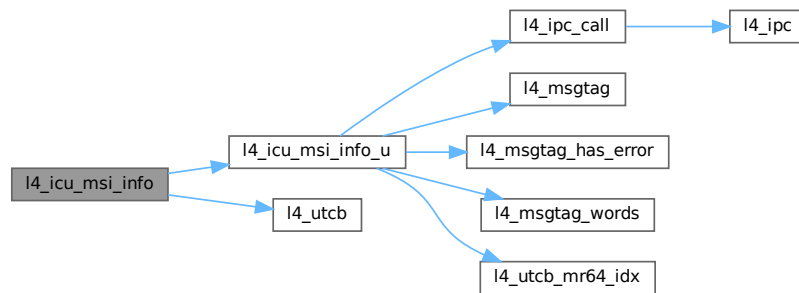
Returns

Syscall return tag

Definition at line 508 of file `icu.h`.

References `l4_icu_msi_info_u()`, `L4_NOTHROW`, and `l4_utcb()`.

Here is the call graph for this function:



14.1.10.7.4.8 l4_icu_msi_info_u()

```

l4_msgtag_t l4_icu_msi_info_u (
    l4_cap_idx_t icu,
    unsigned irqnum,
    l4_uint64_t source,
    l4_icu_msi_info_t * msi_info,
    l4_utcb_t * utcb) [inline]
  
```

Get MSI info about IRQ.

Parameters

	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .
	<i>icu</i>	The ICU object from which MSI information shall be retrieved.
	<i>irqnum</i>	IRQ line at the ICU.
	<i>source</i>	Platform dependent requester ID for MSIs. On IA32 we use a 20bit source filter value as described in the Intel IRQ remapping specification.
out	<i>msi_info</i>	A l4_icu_msi_info_t structure receiving the address and the data value to trigger this MSI.

Returns

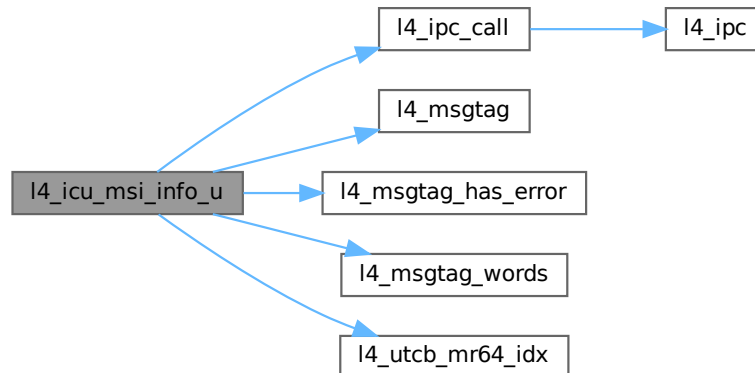
Syscall return tag

Definition at line 434 of file [icu.h](#).

References [L4_ICU_OP_MSI_INFO](#), [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), [l4_msgtag\(\)](#), [l4_msgtag_has_error\(\)](#), [l4_msgtag_words\(\)](#), [L4_NOTHROW](#), [L4_PROTO_IRQ](#), [L4_UNLIKELY](#), [l4_utcb_mr64_idx\(\)](#), [l4_msg_regs_t::mr](#), and [l4_msg_regs_t::mr64](#).

Referenced by [l4_icu_msi_info\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.10.7.4.9 l4_icu_set_mode()

```

l4_msgtag_t l4_icu_set_mode (
    l4_cap_idx_t icu,
    unsigned irqnum,
    l4_umword_t mode) [inline]
  
```

Set interrupt mode.

Parameters

<i>icu</i>	The ICU object.
<i>irqnum</i>	IRQ line at the ICU.
<i>mode</i>	Mode, see L4_irq_mode .

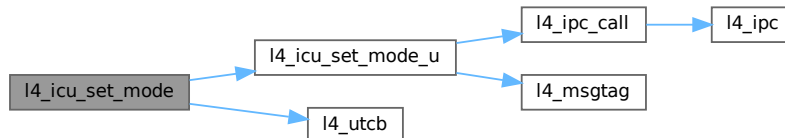
Returns

Syscall return tag

Definition at line 523 of file [icu.h](#).

References [l4_icu_set_mode_u\(\)](#), [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:

**14.1.10.7.4.10 l4_icu_set_mode_u()**

```

l4_msgtag_t l4_icu_set_mode_u (
    l4_cap_idx_t icu,
    unsigned irqnum,
    l4_umword_t mode,
    l4_utcb_t * utcb) [inline]
  
```

Set interrupt mode.

Parameters

<i>icu</i>	The ICU object.
<i>irqnum</i>	IRQ line at the ICU.
<i>mode</i>	Mode, see L4_irq_mode .
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Returns

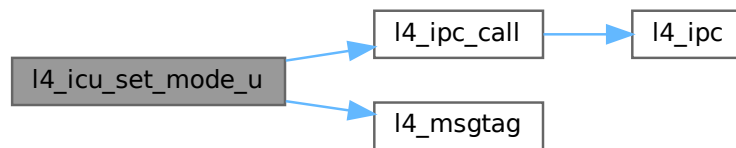
Syscall return tag

Definition at line 457 of file [icu.h](#).

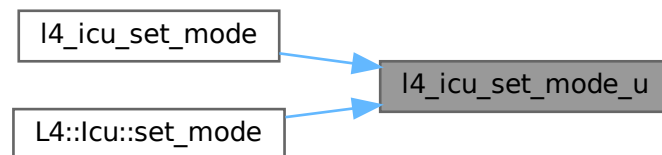
References [L4_ICU_OP_SET_MODE](#), [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), [l4_msgtag\(\)](#), [L4_NOTHROW](#), [L4_PROTO_IRQ](#), and [l4_msg_regs_t::mr](#).

Referenced by [l4_icu_set_mode\(\)](#), and [L4::l4::set_mode\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.10.7.4.11 l4_icu_unbind()

```

l4_msgtag_t l4_icu_unbind (
    l4_cap_idx_t icu,
    unsigned irqnum,
    l4_cap_idx_t irq) [inline]
  
```

Remove binding of an interrupt line from the interrupt controller object.

Parameters

<i>icu</i>	The ICU object from where the binding shall be removed.
<i>irqnum</i>	IRQ line at the ICU.
<i>irq</i>	IRQ object to remove from the ICU.

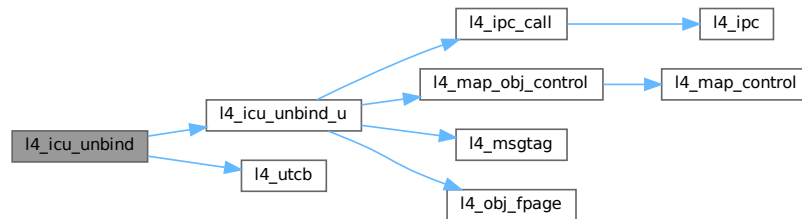
Returns

Syscall return tag

Definition at line 500 of file [icu.h](#).

References [l4_icu_unbind_u\(\)](#), [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:

**14.1.10.7.4.12 l4_icu_unbind_u()**

```

l4_msgtag_t l4_icu_unbind_u (
    l4_cap_idx_t icu,
    unsigned irqnum,
    l4_cap_idx_t irq,
    l4_utcb_t * utcb) [inline]
  
```

Remove binding of an interrupt line from the interrupt controller object.

Parameters

<i>icu</i>	The ICU object from where the binding shall be removed.
<i>irqnum</i>	IRQ line at the ICU.
<i>irq</i>	IRQ object to remove from the ICU.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Returns

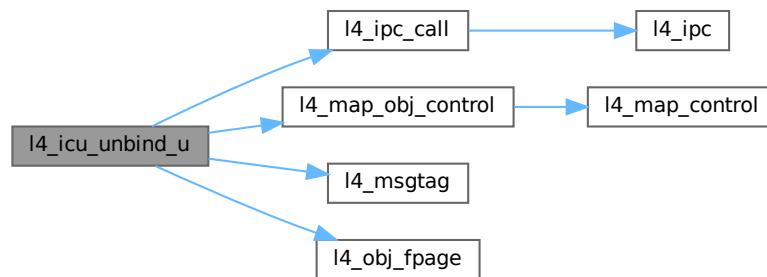
Syscall return tag

Definition at line 408 of file [icu.h](#).

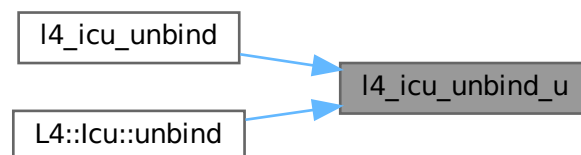
References [L4_CAP_FPAGE_RWS](#), [L4_ICU_OP_UNBIND](#), [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), [l4_map_obj_control\(\)](#), [l4_msgtag\(\)](#), [L4_NOTHROW](#), [l4_obj_fpage\(\)](#), [L4_PROTO_IRQ](#), [l4_msg_regs_t::mr](#), and [l4_fpage_t::raw](#).

Referenced by [l4_icu_unbind\(\)](#), and [L4::Icu::unbind\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.10.7.4.13 l4_icu_unmask()

```

l4_msgtag_t l4_icu_unmask (
    l4_cap_idx_t icu,
    unsigned irqnum,
    l4_umword_t * label,
    l4_timeout_t to) [inline]
  
```

Unmask an IRQ line.

Parameters

<i>icu</i>	The ICU object where the IRQ line shall be unmasked.
<i>irqnum</i>	IRQ line at the ICU.
<i>label</i>	If non-NULL, the function also performs an open wait IPC operation waiting for the next message, and the received label is returned here.
<i>to</i>	IPC timeout, if unsure use <code>L4_IPC_NEVER</code> .

Returns

Syscall return tag. If `label` is `NULL`, this function performs an IPC send-only operation and there is no return value except `L4_MSGTAG_ERROR` indicating success or failure of the send operation. In this case use `l4_ipc_error()` to check for errors and **do not** use `l4_error()`.

Definition at line 513 of file `icu.h`.

References `L4_NOTHROW`, and `l4_utcb()`.

Here is the call graph for this function:

14.1.10.7.4.14 `l4_icu_unmask_u()`

```

l4_msgtag_t l4_icu_unmask_u (
    l4_cap_idx_t icu,
    unsigned irqnum,
    l4_umword_t * label,
    l4_timeout_t to,
    l4_utcb_t * utcb) [inline]
  
```

Unmask the given interrupt line.

Parameters

<i>icu</i>	The ICU object where the IRQ line shall be unmasked. When the object is an IRQ, the given interrupt line is ignored and instead the line which the IRQ is bound to (if any) is unmasked.
------------	--

Its counterpart for explicitly masking an interrupt line is `L4::l4cu::mask()`.

Parameters

	<i>irqnum</i>	The interrupt line that shall be unmasked. Ignored if the object is an IRQ.
out	<i>label</i>	If <code>NULL</code> , this is a send-only unmask. If not <code>NULL</code> , this operation enters an open wait and the <i>protected label</i> shall be received here.
	<i>to</i>	The timeout-pair (send and receive) that shall be used for this operation. The receive timeout is used with a non- <code>NULL label</code> only.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See <code>l4_utcb</code> .

Returns

Syscall return tag. If `label` is `NULL`, this function performs an IPC send-only operation and there is no return value except `L4_MSGTAG_ERROR` indicating success or failure of the send operation. In this case use `l4_ipc_error()` to check for errors and **do not** use `l4_error()`.

Definition at line 488 of file `icu.h`.

References `L4_NOTHROW`.

14.1.10.8 IRQs

C IRQ interface, see `L4::Irq` for the C++ interface.

Collaboration diagram for IRQs:



Enumerations

- enum `L4_irq_mode` {
`L4_IRQ_F_NONE` = 0 , `L4_IRQ_F_SET_MODE` = 0x1 , `L4_IRQ_F_LEVEL` = 0x2 , `L4_IRQ_F_EDGE` = 0x0 ,
`L4_IRQ_F_POS` = 0x0 , `L4_IRQ_F_NEG` = 0x4 , `L4_IRQ_F_BOTH` = 0x8 , `L4_IRQ_F_LEVEL_HIGH` =
`L4_IRQ_F_SET_MODE` | `L4_IRQ_F_LEVEL` | `L4_IRQ_F_POS` ,
`L4_IRQ_F_LEVEL_LOW` = `L4_IRQ_F_SET_MODE` | `L4_IRQ_F_LEVEL` | `L4_IRQ_F_NEG` , `L4_IRQ_F_POS_EDGE`
= `L4_IRQ_F_SET_MODE` | `L4_IRQ_F_EDGE` | `L4_IRQ_F_POS` , `L4_IRQ_F_NEG_EDGE` = `L4_IRQ_F_`
`SET_MODE` | `L4_IRQ_F_EDGE` | `L4_IRQ_F_NEG` , `L4_IRQ_F_BOTH_EDGE` = `L4_IRQ_F_SET_MODE` |
`L4_IRQ_F_EDGE` | `L4_IRQ_F_BOTH` ,
`L4_IRQ_F_MASK` = 0xf , `L4_IRQ_F_SET_WAKEUP` = 0x10 , `L4_IRQ_F_CLEAR_WAKEUP` = 0x20 }

Interrupt attributes.

Functions

- `l4_msgtag_t l4_irq_detach (l4_cap_idx_t irq) L4_NOTHROW`
Detach from an interrupt source.
- `l4_msgtag_t l4_irq_detach_u (l4_cap_idx_t irq, l4_utcb_t *utcb) L4_NOTHROW`
Detach from this interrupt.
- `l4_msgtag_t l4_irq_bind_vcpu (l4_cap_idx_t irq, l4_cap_idx_t thread, l4_umword_t cfg) L4_NOTHROW`
Bind a thread to this Irq for vCPU interrupt forwarding.
- `l4_msgtag_t l4_irq_bind_vcpu_u (l4_cap_idx_t irq, l4_cap_idx_t thread, l4_umword_t cfg, l4_utcb_t *utcb) L4_NOTHROW`
Bind a thread to this Irq for vCPU interrupt forwarding.
- `l4_msgtag_t l4_irq_trigger (l4_cap_idx_t irq) L4_NOTHROW`
Trigger an IRQ.

- [l4_msgtag_t l4_irq_trigger_u](#) ([l4_cap_idx_t](#) irq, [l4_utcb_t](#) *utcb) [L4_NOTHROW](#)
Trigger the object.
- [l4_msgtag_t l4_irq_receive](#) ([l4_cap_idx_t](#) irq, [l4_timeout_t](#) to) [L4_NOTHROW](#)
Unmask and wait for specified IRQ.
- [l4_msgtag_t l4_irq_receive_u](#) ([l4_cap_idx_t](#) irq, [l4_timeout_t](#) timeout, [l4_utcb_t](#) *utcb) [L4_NOTHROW](#)
Unmask and wait for this IRQ.
- [l4_msgtag_t l4_irq_wait](#) ([l4_cap_idx_t](#) irq, [l4_umword_t](#) *label, [l4_timeout_t](#) to) [L4_NOTHROW](#)
Unmask IRQ and wait for any message.
- [l4_msgtag_t l4_irq_wait_u](#) ([l4_cap_idx_t](#) irq, [l4_umword_t](#) *label, [l4_timeout_t](#) timeout, [l4_utcb_t](#) *utcb) [L4_NOTHROW](#)
Unmask IRQ and (open) wait for any message.
- [l4_msgtag_t l4_irq_unmask](#) ([l4_cap_idx_t](#) irq) [L4_NOTHROW](#)
Unmask IRQ.
- [l4_msgtag_t l4_irq_unmask_u](#) ([l4_cap_idx_t](#) irq, [l4_utcb_t](#) *utcb) [L4_NOTHROW](#)
Unmask this IRQ.

14.1.10.8.1 Detailed Description

C IRQ interface, see [L4::Irq](#) for the C++ interface.

The IRQ interface provides access to abstract interrupts provided by the microkernel. Interrupts may be

- hardware interrupts provided by the platform interrupt controller,
- virtual device interrupts provided by the microkernel's virtual devices (virtual serial or trace buffer) or
- virtual interrupts that can be triggered by user programs (IRQs) via [l4_irq_trigger\(\)](#).

For hardware and virtual device interrupts the `Irq` object must be bound to an interrupt source, see [Interrupt controller](#). To receive interrupts, the `Irq` object must be bound to a thread, see [l4_rcv_ep_bind_thread\(\)](#) and [l4_rcv_ep_bind_snd_destination\(\)](#).

IRQ objects can be created using a factory, see the [Factory](#) API (use [l4_factory_create_irq\(\)](#)).

Include File

```
#include <l4/sys/irq.h>
```

For the C++ interface refer to the [L4::Irq](#) API for an overview.

14.1.10.8.2 Enumeration Type Documentation

14.1.10.8.2.1 L4_irq_mode

```
enum L4\_irq\_mode
```

Interrupt attributes.

Enumerator

L4_IRQ_F_NONE	Flow types. None
L4_IRQ_F_SET_MODE	Valid flag, if not set, the set_mode operation does nothing.
L4_IRQ_F_LEVEL	Level triggered.
L4_IRQ_F_EDGE	Edge triggered.
L4_IRQ_F_POS	Positive trigger.
L4_IRQ_F_NEG	Negative trigger.
L4_IRQ_F_BOTH	Both edges trigger.
L4_IRQ_F_LEVEL_HIGH	Level high trigger.
L4_IRQ_F_LEVEL_LOW	Level low trigger.
L4_IRQ_F_POS_EDGE	Positive edge trigger.
L4_IRQ_F_NEG_EDGE	Negative edge trigger.
L4_IRQ_F_BOTH_EDGE	Both edges trigger.
L4_IRQ_F_MASK	Mask.
L4_IRQ_F_SET_WAKEUP	Wakeup source? Use irq as wakeup source
L4_IRQ_F_CLEAR_WAKEUP	Do not use irq as wakeup source.

Definition at line 70 of file [icu.h](#).

14.1.10.8.3 Function Documentation

14.1.10.8.3.1 l4_irq_bind_vcpu()

```
l4_msgtag_t l4_irq_bind_vcpu (
    l4_cap_idx_t irq,
    l4_cap_idx_t thread,
    l4_umword_t cfg) [inline]
```

Bind a thread to this Irq for vCPU interrupt forwarding.

If the interrupt is triggered, the kernel will directly inject the interrupt into the guest. This requires that the thread is currently in extended vCPU user mode. Otherwise the interrupt will stay pending and gets injected on the next vCPU user mode transition. Optionally a doorbell Irq can be registered on the thread (see `Thread::register_doorbell_irq()`) that is triggered in this case.

If a guest has acknowledged the interrupt but has not yet issued an EOI (i.e. the interrupt is in "active" state), it is not possible to bind the Irq to a new thread object. Either wait for the guest to issue the EOI or detach() from the current thread. In this case the interrupt will stay active in the guest and it is the responsibility of the VMM to handle the eventual EOI of the guest.

Parameters

<i>irq</i>	The IRQ object that shall be bound.
<i>thread</i>	Thread object this Irq shall be bound to.
<i>cfg</i>	Architecture specific interrupt configuration.

Returns

Syscall return tag

Return values

<code>-L4_EPERM</code>	Insufficient permissions; see precondition.
<code>-L4_EBUSY</code>	Cannot bind to new thread because interrupt is active on previous thread and guest has to issue end-of-interrupt first.
<code>-L4_ENOSYS</code>	The kernel does not support direct interrupt forwarding.

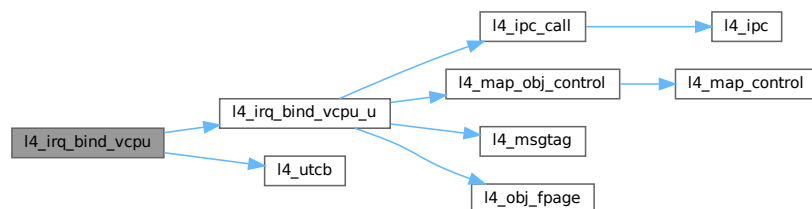
Precondition

The capabilities `irq` and `thread` both must have the permission [L4_CAP_FPAGE_S](#).

Definition at line 300 of file [irq.h](#).

References [l4_irq_bind_vcpu_u\(\)](#), [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.8.3.2 l4_irq_bind_vcpu_u()

```

l4_msgtag_t l4_irq_bind_vcpu_u (
    l4_cap_idx_t irq,
    l4_cap_idx_t thread,
    l4_umword_t cfg,
    l4_utcb_t * utcb) [inline]

```

Bind a thread to this Irq for vCPU interrupt forwarding.

Parameters

<i>irq</i>	The IRQ object that shall be bound. If the interrupt is triggered, the kernel will directly inject the interrupt into the guest. This requires that the thread is currently in extended vCPU user mode. Otherwise the interrupt will stay pending and gets injected on the next vCPU user mode transition. Optionally a doorbell Irq can be registered on the thread (see <code>Thread::register_doorbell_irq()</code>) that is triggered in this case.
------------	--

If a guest has acknowledged the interrupt but has not yet issued an EOI (i.e. the interrupt is in "active" state), it is not possible to bind the Irq to a new thread object. Either wait for the guest to issue the EOI or `detach()` from the current thread. In this case the interrupt will stay active in the guest and it is the responsibility of the VMM to handle the eventual EOI of the guest.

Parameters

<i>thread</i>	Thread object this Irq shall be bound to.
<i>cfg</i>	Architecture specific interrupt configuration.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Returns

Syscall return tag

Return values

<code>-L4_EPERM</code>	Insufficient permissions; see precondition.
<code>-L4_EBUSY</code>	Cannot bind to the new thread because interrupt is active on previous thread and guest has to issue end-of-interrupt first.
<code>-L4_ENOSYS</code>	The kernel does not support direct interrupt forwarding.

Precondition

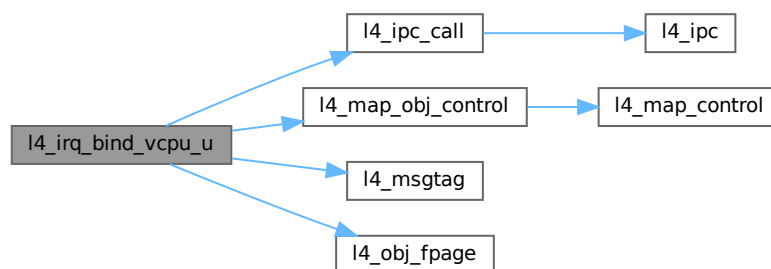
The invoked Irq capability and the capability `thread` both must have the permission [L4_CAP_FPAGE_S](#).

Definition at line 250 of file [irq.h](#).

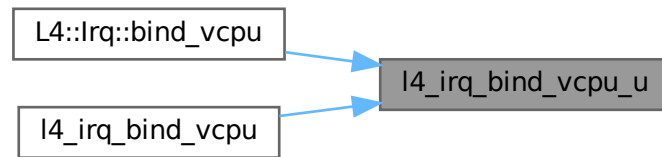
References [L4_CAP_FPAGE_RWS](#), [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), [l4_map_obj_control\(\)](#), [l4_msgtag\(\)](#), [L4_NOTHROW](#), [l4_obj_fpage\(\)](#), [L4_PROTO_IRQ_SENDER](#), [l4_msg_regs_t::mr](#), and [l4_fpage_t::raw](#).

Referenced by [L4::Irq::bind_vcpu\(\)](#), and [l4_irq_bind_vcpu\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.10.8.3.3 l4_irq_detach()

```
l4_msgtag_t l4_irq_detach (
    l4_cap_idx_t irq) [inline]
```

Detach from an interrupt source.

Parameters

<i>irq</i>	The IRQ object that shall be detached.
------------	--

Returns

Syscall return tag

Return values

0	Successfully detached, there was no interrupt pending.
1	Successfully detached, there was an interrupt pending.
2	Successfully detached, an active vIRQ was abandoned.
-L4_EPERM	Insufficient permissions; see precondition.

Precondition

The capability `irq` must have the permission `L4_CAP_FPAGE_S`.

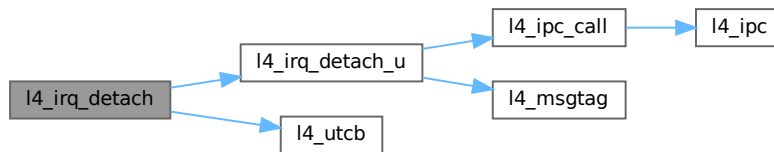
Examples

[examples/sys/isr/main.c](#).

Definition at line 294 of file [irq.h](#).

References [l4_irq_detach_u\(\)](#), [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.8.3.4 l4_irq_detach_u()

```

l4_msgtag_t l4_irq_detach_u (
    l4_cap_idx_t irq,
    l4_utcb_t * utcb) [inline]
  
```

Detach from this interrupt.

Parameters

<i>irq</i>	The IRQ object that shall be detached.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Returns

Syscall return tag

Return values

0	Successfully detached, there was no interrupt pending.
1	Successfully detached, there was an interrupt pending.
2	Successfully detached, an active vIRQ was abandoned.
-L4_EPERM	Insufficient permissions; see precondition.

Precondition

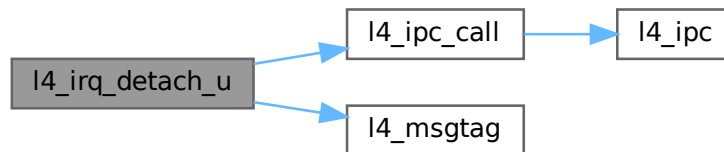
The invoked Irq capability must have the permission [L4_CAP_FPAGE_S](#).

Definition at line 242 of file [irq.h](#).

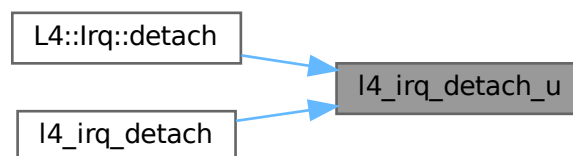
References [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), [l4_msgtag\(\)](#), [L4_NOTHROW](#), and [L4_PROTO_IRQ_SENDER](#).

Referenced by [L4::Irq::detach\(\)](#), and [l4_irq_detach\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**14.1.10.8.3.5 l4_irq_receive()**

```

l4_msgtag_t l4_irq_receive (
    l4_cap_idx_t irq,
    l4_timeout_t to) [inline]
  
```

Unmask and wait for specified IRQ.

Parameters

<i>irq</i>	The IRQ object that shall be unmasked.
<i>to</i>	Timeout.

Returns

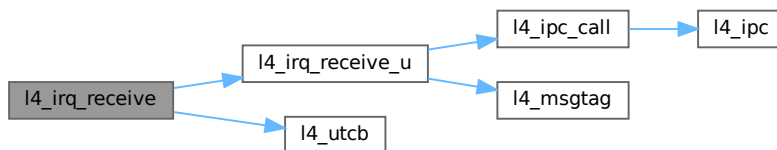
Syscall return tag

Definition at line 313 of file [irq.h](#).

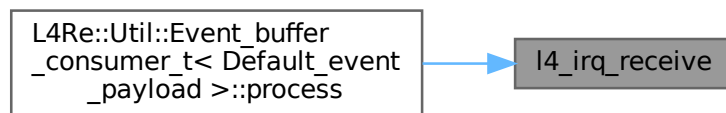
References [l4_irq_receive_u\(\)](#), [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Referenced by [L4Re::Util::Event_buffer_consumer_t< Default_event_payload >::process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**14.1.10.8.3.6 l4_irq_receive_u()**

```

l4_msgtag_t l4_irq_receive_u (
    l4_cap_idx_t irq,
    l4_timeout_t timeout,
    l4_utcb_t * utcb) [inline]
  
```

Unmask and wait for this IRQ.

Parameters

<i>irq</i>	The IRQ object that shall be unmasked.
<i>timeout</i>	Timeout.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Returns

Syscall return tag

Note

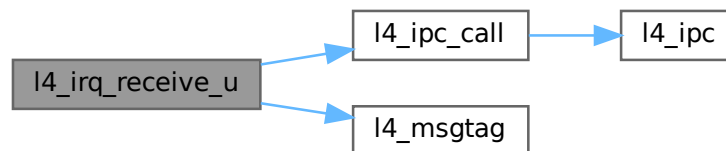
If this is the function normally used for your IRQs consider using [L4::Semaphore](#) instead of [L4::Irq](#).

Definition at line 270 of file [irq.h](#).

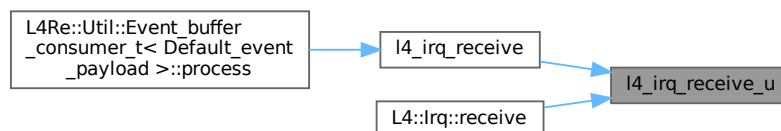
References [l4_ipc_call\(\)](#), [l4_msgtag\(\)](#), [L4_NOTHROW](#), and [L4_PROTO_IRQ](#).

Referenced by [l4_irq_receive\(\)](#), and [L4::Irq::receive\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**14.1.10.8.3.7 l4_irq_trigger()**

```
l4_msgtag_t l4_irq_trigger (
    l4_cap_idx_t irq) [inline]
```

Trigger an IRQ.

Parameters

<i>irq</i>	The IRQ object that shall be triggered.
------------	---

Returns

Syscall return tag.

Note that this function is a send only operation, i.e. there is no return value except for a failed send operation. Especially `l4_error()` will return an error value from the message tag which still contains the IRQ protocol used for the send operation.

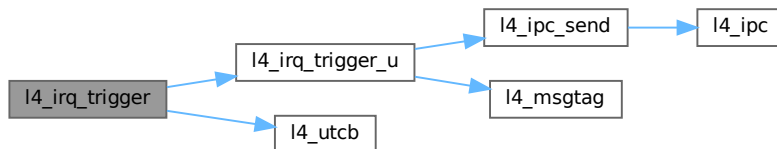
Use `l4_ipc_error()` to check for (send) errors.

Definition at line 307 of file `irq.h`.

References `l4_irq_trigger_u()`, `L4_NOTHROW`, and `l4_utcb()`.

Referenced by `l4_semaphore_up()`.

Here is the call graph for this function:



Here is the caller graph for this function:

**14.1.10.8.3.8 l4_irq_trigger_u()**

```

l4_msgtag_t l4_irq_trigger_u (
    l4_cap_idx_t irq,
    l4_utcb_t * utcb) [inline]
  
```

Trigger the object.

Parameters

<i>irq</i>	The IRQ object that shall be triggered.
------------	---

<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .
-------------	--

Returns

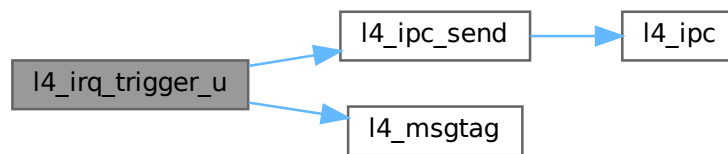
Syscall return tag for a send-only operation, this means there is no return value except [L4_MSGTAG_ERROR](#) indicating success or failure of the send operation. Use [l4_ipc_error\(\)](#) to check for errors and **do not** use [l4_error\(\)](#).

Definition at line 263 of file [irq.h](#).

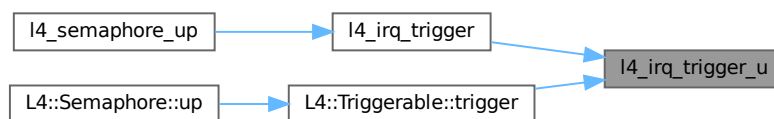
References [L4_IPC_BOTH_TIMEOUT_0](#), [l4_ipc_send\(\)](#), [l4_msgtag\(\)](#), [L4_NOTHROW](#), and [L4_PROTO_IRQ](#).

Referenced by [l4_irq_trigger\(\)](#), and [L4::Triggerable::trigger\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.10.8.3.9 l4_irq_unmask()

```

l4_msgtag_t l4_irq_unmask (
    l4_cap_idx_t irq) [inline]
  
```

Unmask IRQ.

Parameters

<i>irq</i>	The IRQ object that shall be unmasked.
------------	--

Returns

Syscall return tag

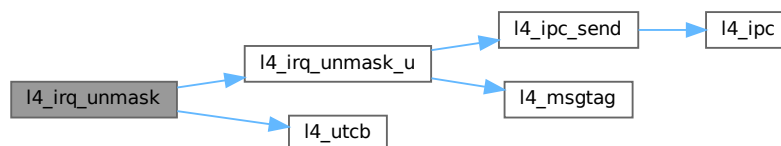
Note

[l4_irq_wait\(\)](#) and [l4_irq_receive\(\)](#) are doing the unmask themselves.

Definition at line 326 of file [irq.h](#).

References [l4_irq_unmask_u\(\)](#), [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.8.3.10 l4_irq_unmask_u()

```

l4_msgtag_t l4_irq_unmask_u (
    l4_cap_idx_t irq,
    l4_utcb_t * utcb) [inline]
  
```

Unmask this IRQ.

Parameters

<i>irq</i>	The IRQ object that shall be unmasked.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Returns

Syscall return tag for a send-only operation, this means there is no return value except [L4_MSGTAG_ERROR](#) indicating success or failure of the send operation. Use [l4_ipc_error\(\)](#) to check for errors and **do not** use [l4_error\(\)](#).

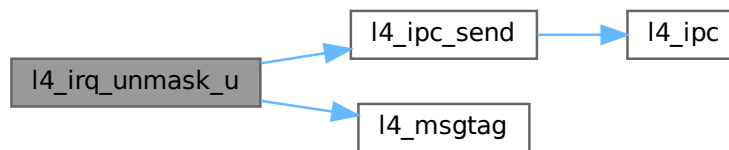
`Irq::wait()` and `Irq::receive()` operations already include an `unmask()`, do not use an extra `unmask()` in these cases.

Definition at line 286 of file `irq.h`.

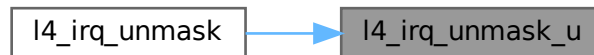
References `L4_IPC_NEVER`, `l4_ipc_send()`, `l4_msgtag()`, `L4_NOTHROW`, and `L4_PROTO_IRQ`.

Referenced by `l4_irq_unmask()`.

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.10.8.3.11 l4_irq_wait()

```

l4_msgtag_t l4_irq_wait (
    l4_cap_idx_t irq,
    l4_unword_t * label,
    l4_timeout_t to) [inline]
  
```

Unmask IRQ and wait for any message.

Parameters

<i>irq</i>	The IRQ object that shall be unmasked.
<i>label</i>	Receive label.
<i>to</i>	Timeout.

Returns

Syscall return tag

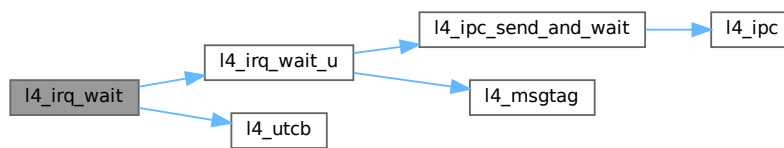
Examples

[examples/sys/isr/main.c](#).

Definition at line 319 of file [irq.h](#).

References [l4_irq_wait_u\(\)](#), [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:

**14.1.10.8.3.12 l4_irq_wait_u()**

```

l4_msgtag_t l4_irq_wait_u (
    l4_cap_idx_t irq,
    l4_umword_t * label,
    l4_timeout_t timeout,
    l4_utcb_t * utcb) [inline]
  
```

Unmask IRQ and (open) wait for any message.

Parameters

<i>irq</i>	The IRQ object that shall be unmasked.
<i>label</i>	The <i>protected label</i> shall be received here.
<i>timeout</i>	Timeout.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Returns

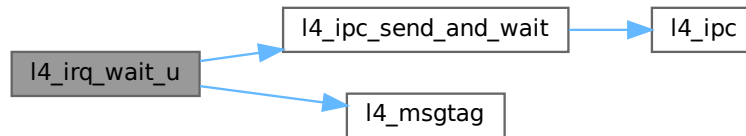
Syscall return tag

Definition at line 277 of file [irq.h](#).

References [l4_ipc_send_and_wait\(\)](#), [l4_msgtag\(\)](#), [L4_NOTHROW](#), and [L4_PROTO_IRQ](#).

Referenced by [l4_irq_wait\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.10.9 Platform Control C API

C interface for controlling platform-wide properties, see [L4::Platform_control](#) for the C++ interface.

Collaboration diagram for Platform Control C API:



Functions

- `l4_msgtag_t l4_platform_ctl_set_task_asid (l4_cap_idx_t pfc, l4_cap_idx_t task, l4_umword_t asid) L4_NOTHROW`
Set ASID of task.
- `l4_msgtag_t l4_platform_ctl_system_suspend (l4_cap_idx_t pfc, l4_umword_t extras) L4_NOTHROW`
Enter suspend to RAM.
- `l4_msgtag_t l4_platform_ctl_system_shutdown (l4_cap_idx_t pfc, l4_umword_t reboot) L4_NOTHROW`
Shutdown or reboot the system.
- `l4_msgtag_t l4_platform_ctl_cpu_allow_shutdown (l4_cap_idx_t pfc, l4_umword_t phys_id, l4_umword_t enable) L4_NOTHROW`

Allow a CPU to be shut down.

- [l4_msgtag_t l4_platform_ctl_cpu_enable](#) ([l4_cap_idx_t](#) pfc, [l4_umword_t](#) phys_id) [L4_NOTHROW](#)

Enable an offline CPU.

- [l4_msgtag_t l4_platform_ctl_cpu_disable](#) ([l4_cap_idx_t](#) pfc, [l4_umword_t](#) phys_id) [L4_NOTHROW](#)

Disable an online CPU.

14.1.10.9.1 Detailed Description

C interface for controlling platform-wide properties, see [L4::Platform_control](#) for the C++ interface.

Include File

```
#include <l4/sys/platform_control.h>
```

The API allows a client to suspend, reboot or shutdown the system.

For the C++ interface refer to [L4::Platform_control](#)

14.1.10.9.2 Function Documentation

14.1.10.9.2.1 l4_platform_ctl_cpu_allow_shutdown()

```
l4\_msgtag\_t l4_platform_ctl_cpu_allow_shutdown (
    l4\_cap\_idx\_t pfc,
    l4\_umword\_t phys_id,
    l4\_umword\_t enable) [inline]
```

Allow a CPU to be shut down.

Parameters

<i>pfc</i>	Capability selector for the platform-control object.
<i>phys_id</i>	Physical CPU id of CPU (e.g. local APIC id) to enable.
<i>enable</i>	Allow shutdown when 1, disallow when 0.

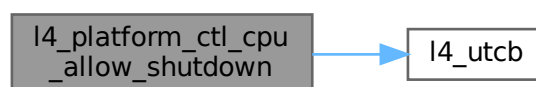
Returns

Syscall return tag

Definition at line 242 of file [platform_control.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.9.2.2 l4_platform_ctl_cpu_disable()

```
l4_msgtag_t l4_platform_ctl_cpu_disable (
    l4_cap_idx_t pfc,
    l4_umword_t phys_id) [inline]
```

Disable an online CPU.

Parameters

<i>pfc</i>	Capability to the platform control object.
<i>phys_id</i>	Physical CPU id of CPU (e.g. local APIC id) to disable.

Returns

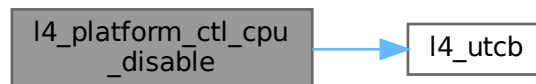
System call message tag

This function is currently only supported on the ARM EXYNOS platform.

Definition at line 281 of file [platform_control.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:

**14.1.10.9.2.3 l4_platform_ctl_cpu_enable()**

```
l4_msgtag_t l4_platform_ctl_cpu_enable (
    l4_cap_idx_t pfc,
    l4_umword_t phys_id) [inline]
```

Enable an offline CPU.

Parameters

<i>pfc</i>	Capability to the platform control object.
<i>phys_id</i>	Physical CPU id of CPU (e.g. local APIC id) to enable.

Returns

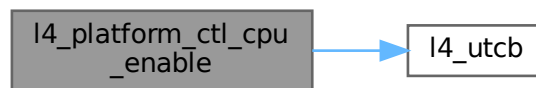
System call message tag

This function is currently only supported on the ARM EXYNOS platform.

Definition at line 274 of file [platform_control.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:

**14.1.10.9.2.4 l4_platform_ctl_set_task_asid()**

```

l4_msgtag_t l4_platform_ctl_set_task_asid (
    l4_cap_idx_t pfc,
    l4_cap_idx_t task,
    l4_umword_t asid) [inline]
  
```

Set ASID of task.

On Cortex-R52 platforms, it might be necessary to control the VMID of a task or virtual machine explicitly. The IOMPU on such platforms will use it for further access control of device memory accesses. A privileged component can use this call to control the value.

The caller must have write permissions to the destination task.

Parameters

<i>pfc</i>	Capability selector for the platform-control object.
<i>task</i>	Capability selector of destination task
<i>asid</i>	New ASID value

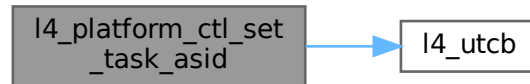
Returns

Syscall return tag

Definition at line 62 of file [__platform_control-arm.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:

**14.1.10.9.2.5 l4_platform_ctl_system_shutdown()**

```

l4_msgtag_t l4_platform_ctl_system_shutdown (
    l4_cap_idx_t pfc,
    l4_umword_t reboot) [inline]
  
```

Shutdown or reboot the system.

Parameters

<i>pfc</i>	Capability selector for the platform-control object.
<i>reboot</i>	Shutdown when 0, or reboot when 1.

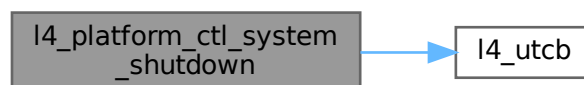
Returns

Syscall return tag

Definition at line 221 of file [platform_control.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.9.2.6 l4_platform_ctl_system_suspend()

```
l4_msgtag_t l4_platform_ctl_system_suspend (
    l4_cap_idx_t pfc,
    l4_umword_t extras) [inline]
```

Enter suspend to RAM.

Precondition

Must only be invoked on the boot CPU. Furthermore it must be ensured that the invoking thread is not migrated to a different CPU during the suspend.

Parameters

<i>pfc</i>	Capability selector for the platform-control object.
<i>extras</i>	Some extra platform-specific information needed to enter suspend to RAM. On x86 platforms and when using the Platform_control object provided by Fiasco, the value defines the sleep state. The sleep states are defined in the ACPI table. Other platforms as well as lo's Platform_control object don't make use of this value at the moment.

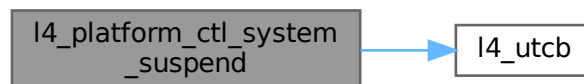
Returns

Syscall return tag

Definition at line 214 of file [platform_control.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.10 Scheduler

C interface of the Scheduler kernel object, see [L4::Scheduler](#) for the C++ interface.

Collaboration diagram for Scheduler:



Data Structures

- struct [l4_sched_cpu_set_t](#)
CPU sets.
- struct [l4_sched_param_t](#)
Scheduler parameter set.

Typedefs

- typedef struct [l4_sched_cpu_set_t](#) **[l4_sched_cpu_set_t](#)**
CPU sets.
- typedef struct [l4_sched_param_t](#) **[l4_sched_param_t](#)**
Scheduler parameter set.

Enumerations

- enum [l4_scheduler_classes](#) { [L4_SCHEDULER_CLASS_FIXED_PRIO](#) = 1UL << 1 , [L4_SCHEDULER_CLASS_WFQ](#) = 1UL << 2 }
Supported scheduler classes.
- enum [l4_scheduler_ops](#) { [L4_SCHEDULER_INFO_OP](#) = 0UL , [L4_SCHEDULER_RUN_THREAD_OP](#) = 1UL , [L4_SCHEDULER_IDLE_TIME_OP](#) = 2UL }
Operations on the Scheduler object.

Functions

- [l4_sched_cpu_set_t](#) [l4_sched_cpu_set](#) ([l4_umword_t](#) offset, unsigned char granularity, [l4_umword_t](#) map=1) [L4_NOTHROW](#)
- [l4_msgtag_t](#) [l4_scheduler_info](#) ([l4_cap_idx_t](#) scheduler, [l4_umword_t](#) *cpu_max, [l4_sched_cpu_set_t](#) *cpus) [L4_NOTHROW](#))
Get scheduler information.
- [l4_msgtag_t](#) [l4_scheduler_info_with_classes](#) ([l4_cap_idx_t](#) scheduler, [l4_umword_t](#) *cpu_max, [l4_sched_cpu_set_t](#) *cpus, [l4_umword_t](#) *sched_classes) [L4_NOTHROW](#))
Get scheduler information.
- [l4_sched_param_t](#) [l4_sched_param](#) (unsigned prio, [l4_umword_t](#) quantum=0) [L4_NOTHROW](#)
Construct scheduler parameter.
- [l4_msgtag_t](#) [l4_scheduler_run_thread](#) ([l4_cap_idx_t](#) scheduler, [l4_cap_idx_t](#) thread, [l4_sched_param_t](#) const *sp) [L4_NOTHROW](#))
Run a thread on a Scheduler.
- [l4_msgtag_t](#) [l4_scheduler_idle_time](#) ([l4_cap_idx_t](#) scheduler, [l4_sched_cpu_set_t](#) const *cpus, [l4_kernel_clock_t](#) *us) [L4_NOTHROW](#))
Query the idle time (in μ s) of a CPU.
- int [l4_scheduler_is_online](#) ([l4_cap_idx_t](#) scheduler, [l4_umword_t](#) cpu) [L4_NOTHROW](#)
Query if a CPU is online.

14.1.10.10.1 Detailed Description

C interface of the Scheduler kernel object, see [L4::Scheduler](#) for the C++ interface.

The Scheduler interface allows a client to manage CPU resources. The API provides functions to query scheduler information, check the online state of CPUs, query CPU idle time and to start threads on defined CPU sets.

The scheduler offers a virtual device IRQ which triggers when the number of online cores changes, e.g. due to hotplug events. In contrast to hardware IRQs, this IRQ implements a limited functionality:

- Only IRQ line 0 is supported, no MSI vectors.
- The IRQ is edge-triggered and the IRQ mode cannot be changed.
- As the IRQ is edge-triggered, it does not have to be explicitly unmasked.

It depends on the platform, which hotplug events actually trigger the IRQ. Many platforms only support triggering the IRQ when a CPU core different from the boot CPU goes online.

Include File

```
#include <l4/sys/scheduler.h>
```

14.1.10.10.2 Enumeration Type Documentation

14.1.10.10.2.1 L4_scheduler_classes

```
enum L4_scheduler_classes
```

Supported scheduler classes.

Enumerator

L4_SCHEDULER_CLASS_FIXED_PRIO	Fixed-priority scheduler.
L4_SCHEDULER_CLASS_WFQ	Weighted fair queuing scheduler.

Definition at line 46 of file [scheduler.h](#).

14.1.10.10.2.2 L4_scheduler_ops

```
enum L4_scheduler_ops
```

Operations on the Scheduler object.

Enumerator

L4_SCHEDULER_INFO_OP	Query infos about the scheduler.
----------------------	----------------------------------

L4_SCHEDULER_RUN_THREAD_OP	Run a thread on this scheduler.
L4_SCHEDULER_IDLE_TIME_OP	Query idle time for the scheduler.

Definition at line 269 of file [scheduler.h](#).

14.1.10.10.3 Function Documentation

14.1.10.10.3.1 l4_sched_cpu_set()

```
l4_sched_cpu_set_t l4_sched_cpu_set (
    l4_umword_t offset,
    unsigned char granularity,
    l4_umword_t map = 1) [inline]
```

Parameters

<i>offset</i>	Offset. Must be a multiple of 2 ^{granularity} .
<i>granularity</i>	Granularity in log2 notation.
<i>map</i>	Bitmap of CPUs, defaults to 1 in C++.

Returns

CPU set.

Examples

[examples/sys/migrate/thread_migrate.cc](#).

Definition at line 279 of file [scheduler.h](#).

References [l4_sched_cpu_set_t::gran_offset](#), [L4_NOTHROW](#), and [l4_sched_cpu_set_t::map](#).

Referenced by [l4_sched_param\(\)](#).

Here is the caller graph for this function:



14.1.10.10.3.2 l4_sched_param()

```
l4_sched_param_t l4_sched_param (
    unsigned prio,
    l4_umword_t quantum = 0) [inline]
```

Construct scheduler parameter.

Parameters

<i>prio</i>	Thread priority (1-255).
<i>quantum</i>	Timeslice in micro seconds.

The `l4_sched_param_t::affinity` of the returned value contains all CPUs.

Examples

`examples/sys/aliens/main.c`, `examples/sys/migrate/thread_migrate.cc`, `examples/sys/singlestep/main.c`, `examples/sys/start-with-exc/main.c`, and `examples/sys/utcb-ipc/main.c`.

Definition at line 289 of file `scheduler.h`.

References `l4_sched_param_t::affinity`, `L4_NOTHROW`, `l4_sched_cpu_set()`, `l4_sched_param_t::prio`, and `l4_sched_param_t::quantum`.

Here is the call graph for this function:



14.1.10.10.3.3 l4_scheduler_idle_time()

```

l4_msgtag_t l4_scheduler_idle_time (
    l4_cap_idx_t scheduler,
    l4_sched_cpu_set_t const * cpus,
    l4_kernel_clock_t * us) [inline]
  
```

Query the idle time (in μ s) of a CPU.

Parameters

	<i>scheduler</i>	Scheduler object.
	<i>cpus</i>	Set of CPUs to query. Only the idle time of the first selected CPU in <code>cpus.map</code> is queried.
out	<i>us</i>	Idle time of queried CPU in μ s.

Return values

<i>0</i>	Success.
<i>-L4_EINVAL</i>	Invalid CPU requested in cpu set.

This function retrieves the idle time in μ s of the first selected CPU in `cpus.map`. The idle time is the accumulated time a CPU has spent in the idle thread since its last reset. To calculate a load estimate l one has to retrieve the idle time at the beginning ($i1$) and the end ($i2$) of a known time interval t . The load is then calculated as $l = 1 - (i2 - i1)/t$.

The idle time is only defined for online CPUs. Reading the idle time from offline CPUs is undefined and may result in either getting `-L4_EINVAL` or calculating an estimated (incorrect) load of 1.

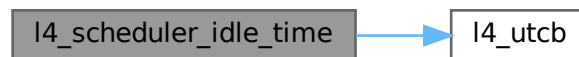
Note

The idle time statistics of remote CPUs is updated on context switch events only, hence may not be up-to-date when requested cross-CPU. To get up-to-date idle time you should use a thread running on the same CPU of which the idle time is requested.

Definition at line 403 of file [scheduler.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:

**14.1.10.10.3.4 l4_scheduler_info()**

```

l4_msgtag_t l4_scheduler_info (
    l4_cap_idx_t scheduler,
    l4_umword_t * cpu_max,
    l4_sched_cpu_set_t * cpus) [inline]
  
```

Get scheduler information.

Parameters

	<i>scheduler</i>	Scheduler object.
out	<i>cpu_max</i>	Maximum number of CPUs ever available. Optional, can be NULL.
in, out	<i>cpus</i>	<i>cpus.offset</i> is first CPU of interest. <i>cpusgranularity</i> (see l4_sched_cpu_set_t). <i>cpus.map</i> Bitmap of online CPUs. Must not be NULL.

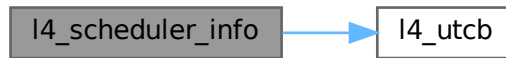
Return values

0	Success.
-L4_ERANGE	The given CPU offset is larger than the maximum number of CPUs.

Definition at line 381 of file [scheduler.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.10.3.5 l4_scheduler_info_with_classes()

```

l4_msgtag_t l4_scheduler_info_with_classes (
    l4_cap_idx_t scheduler,
    l4_umword_t * cpu_max,
    l4_sched_cpu_set_t * cpus,
    l4_umword_t * sched_classes) [inline]
  
```

Get scheduler information.

Parameters

	<i>scheduler</i>	Scheduler object.
out	<i>cpu_max</i>	Maximum number of CPUs ever available. Optional, can be NULL.
in, out	<i>cpus</i>	<i>cpus.offset</i> is first CPU of interest. <i>cpus.granularity</i> (see l4_sched_cpu_set_t). <i>cpus.map</i> Bitmap of online CPUs. Must not be NULL.
out	<i>sched_classes</i>	A bitmap of available scheduling classes (see L4_scheduler_classes). Optional, can be NULL.

Return values

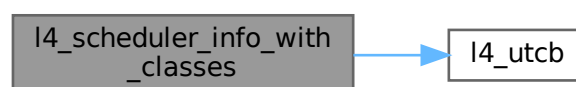
0	Success.
-L4_ERANGE	The given CPU offset is larger than the maximum number of CPUs.

This function delivers the same information as [l4_scheduler_info](#) plus the available scheduler classes (see [L4_scheduler_classes](#)).

Definition at line 388 of file [scheduler.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.10.3.6 l4_scheduler_is_online()

```
int l4_scheduler_is_online (
    l4_cap_idx_t scheduler,
    l4_umword_t cpu) [inline]
```

Query if a CPU is online.

Parameters

<i>scheduler</i>	Scheduler object.
<i>cpu</i>	CPU number whose online status should be queried.

Return values

<i>true</i>	The CPU is online.
<i>false</i>	The CPU is offline

Definition at line 410 of file [scheduler.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.10.3.7 l4_scheduler_run_thread()

```
l4_msgtag_t l4_scheduler_run_thread (
    l4_cap_idx_t scheduler,
    l4_cap_idx_t thread,
    l4_sched_param_t const * sp) [inline]
```

Run a thread on a Scheduler.

Parameters

<i>scheduler</i>	Scheduler object.
<i>thread</i>	Capability of the thread to run.
<i>sp</i>	Scheduling parameters.

Return values

0	Success.
-L4_EINVAL	Invalid size of the scheduling parameter.

This function launches a thread on a CPU determined by the scheduling parameter `sp.affinity`. A thread can be intentionally stopped by migrating it on an offline or an invalid CPU. The thread is only guaranteed to run if the CPU it is migrated to is currently online.

Note

If the target CPU is currently not online, there is no guarantee that the thread will ever run, even if the CPU comes online later on.

A scheduler may impose a policy with regard to selecting CPUs. However the scheduler is required to ensure the following two properties:

- Two threads with disjoint CPU sets must be scheduled to different CPUs.
- Two threads with identical CPU sets selecting only a single CPU must be scheduled to the same CPU.

Examples

[examples/sys/aliens/main.c](#), [examples/sys/singlestep/main.c](#), [examples/sys/start-with-exc/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

Definition at line 396 of file [scheduler.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

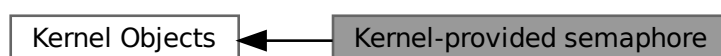
Here is the call graph for this function:



14.1.10.11 Kernel-provided semaphore

C semaphore interface, see [L4::Semaphore](#) for the C++ interface.

Collaboration diagram for Kernel-provided semaphore:



Functions

- [l4_msgtag_t l4_semaphore_up](#) ([l4_cap_idx_t](#) sem) [L4_NOTHROW](#)
Semaphore up operation (wrapper for trigger()).
- [l4_msgtag_t l4_semaphore_down](#) ([l4_cap_idx_t](#) sem, [l4_timeout_t](#) timeout) [L4_NOTHROW](#)
Semaphore down operation.

14.1.10.11.1 Detailed Description

C semaphore interface, see [L4::Semaphore](#) for the C++ interface.

Include File

```
#include <l4/sys/semaphore.h>
```

14.1.10.11.2 Function Documentation

14.1.10.11.2.1 l4_semaphore_down()

```
l4\_msgtag\_t l4_semaphore_down (
    l4\_cap\_idx\_t sem,
    l4\_timeout\_t timeout) [inline]
```

Semaphore down operation.

Parameters

<i>sem</i>	Semaphore object.
<i>timeout</i>	Timeout for blocking the semaphore down operation. Note: The receive timeout of this timeout-pair is significant for blocking, the send part is usually non-blocking.

Returns

Syscall return tag. Use [l4_error\(\)](#) to check for errors.

Return values

-L4_EPERM	Insufficient permissions; see precondition.
---------------------------	---

Precondition

The capability `sem` must have the permission `L4_CAP_FPAGE_S`.

This method decrements the semaphore counter by one, or blocks if the counter is already zero, until either a timeout or cancel condition hits or the counter is increased by an `up()` operation.

Definition at line 100 of file `semaphore.h`.

References `L4_NOTHROW`, and `l4_utcb()`.

Here is the call graph for this function:

**14.1.10.11.2.2 l4_semaphore_up()**

```
l4_msgtag_t l4_semaphore_up (
    l4_cap_idx_t sem) [inline]
```

Semaphore up operation (wrapper for `trigger()`).

Parameters

<code>sem</code>	Semaphore object.
------------------	-------------------

Returns

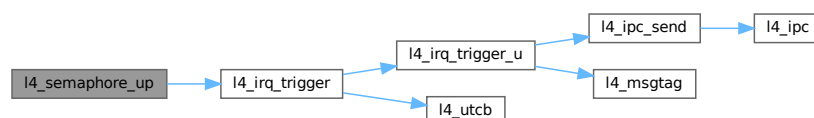
Send-only IPC message return tag. Use `l4_ipc_error()` to check for errors, do **not** use `l4_error()`.

Increases the semaphore counter by one if it is smaller than an unspecified limit. The unspecified limit is guaranteed to be at least $2^{31}-1$.

Definition at line 45 of file `semaphore.h`.

References `l4_irq_trigger()`, and `L4_NOTHROW`.

Here is the call graph for this function:



14.1.10.12 Task

C interface of the Task kernel object, see [L4::Task](#) for the C++ interface.

Collaboration diagram for Task:



Enumerations

- enum [l4_unmap_flags_t](#) { [L4_FP_ALL_SPACES](#) , [L4_FP_DELETE_OBJ](#) , [L4_FP_OTHER_SPACES](#) }
Flags for the unmap operation.

Functions

- [l4_msgtag_t l4_task_vgicc_map](#) ([l4_cap_idx_t](#) task, [l4_fpage_t](#) vgicc_fpage) [L4_NOTHROW](#)
Map the GIC virtual CPU interface page to the task in case Fiasco detected a GIC version 2.
- [l4_msgtag_t l4_task_map](#) ([l4_cap_idx_t](#) dst_task, [l4_cap_idx_t](#) src_task, [l4_fpage_t](#) snd_fpage, [l4_umword_t](#) snd_base) [L4_NOTHROW](#)
Map resources available in the source task to a destination task.
- [l4_msgtag_t l4_task_unmap](#) ([l4_cap_idx_t](#) task, [l4_fpage_t](#) fpage, [l4_umword_t](#) map_mask) [L4_NOTHROW](#)
Revoke rights from the task.
- [l4_msgtag_t l4_task_unmap_batch](#) ([l4_cap_idx_t](#) task, [l4_fpage_t](#) const *fpages, unsigned num_fpages, [l4_umword_t](#) map_mask) [L4_NOTHROW](#)
Revoke rights from a task.
- [l4_msgtag_t l4_task_delete_obj](#) ([l4_cap_idx_t](#) task, [l4_cap_idx_t](#) obj) [L4_NOTHROW](#)
Release capability and delete object.
- [l4_msgtag_t l4_task_release_cap](#) ([l4_cap_idx_t](#) task, [l4_cap_idx_t](#) cap) [L4_NOTHROW](#)
Release object capability.
- [l4_msgtag_t l4_task_cap_valid](#) ([l4_cap_idx_t](#) task, [l4_cap_idx_t](#) cap) [L4_NOTHROW](#)
Check whether a capability is present (refers to an object).
- [l4_msgtag_t l4_task_cap_equal](#) ([l4_cap_idx_t](#) task, [l4_cap_idx_t](#) cap_a, [l4_cap_idx_t](#) cap_b) [L4_NOTHROW](#)
Test whether two capabilities point to the same object with the same permissions (only considering selected permissions).
- [l4_msgtag_t l4_task_add_ku_mem](#) ([l4_cap_idx_t](#) task, [l4_fpage_t](#) *ku_mem) [L4_NOTHROW](#)
Add kernel-user memory.

14.1.10.12.1 Detailed Description

C interface of the Task kernel object, see [L4::Task](#) for the C++ interface.

A task represents a combination of the address spaces provided by the [L4Re](#) micro kernel. A task consists of at least a memory address space and an object address space. On IA32 there is also an IO-port address space.

Task objects are created using the [Factory](#) interface.

Include File

```
#include <l4/sys/task.h>
```

14.1.10.12.2 Enumeration Type Documentation

14.1.10.12.2.1 l4_unmap_flags_t

enum [l4_unmap_flags_t](#)

Flags for the unmap operation.

See also

[L4::Task::unmap\(\)](#) and [l4_task_unmap\(\)](#)

Enumerator

L4_FP_ALL_SPACES	<p>Flag to tell the unmap operation to revoke permissions from all child mappings including the mapping in the invoked task.</p> <p>Note</p> <p>Object capabilities are not hierarchical – they have no children. The result of the map operation on an object capability is a copy of that capability in the object space of the destination task. An unmap operation on object capabilities is a no-op if this flag is not specified.</p> <p>See also</p> <p>L4::Task::unmap() l4_task_unmap()</p>
L4_FP_DELETE_OBJ	<p>Flag that indicates that an unmap operation on object capabilities shall try to delete the corresponding objects immediately. This flag implies the L4_FP_ALL_SPACES flag. The concept of deletion is only applicable to kernel objects. Therefore, for memory and I/O port capabilities, this flag has the same effect as L4_FP_ALL_SPACES alone.</p> <p>See also</p> <p>L4::Task::unmap() l4_task_unmap()</p> <p>Note</p> <p>Specifying L4_FP_DELETE_OBJ ^ L4_FP_ALL_SPACES is treated as L4_FP_OTHER_SPACES.</p>
L4_FP_OTHER_SPACES	<p>Counterpart to L4_FP_ALL_SPACES; revoke permissions from child mappings only.</p> <p>See also</p> <p>L4::Task::unmap() l4_task_unmap()</p>

Definition at line 169 of file [consts.h](#).

14.1.10.12.3 Function Documentation

14.1.10.12.3.1 l4_task_add_ku_mem()

```
l4_msgtag_t l4_task_add_ku_mem (
    l4_cap_idx_t task,
    l4_fpage_t * ku_mem) [inline]
```

Add kernel-user memory.

Parameters

	<i>task</i>	Capability selector of the task to add the memory to.
<i>in, out</i>	<i>ku_mem</i>	Flexpage describing the virtual area the memory goes to. On systems without MMU, the flexpage is adjusted to reflect the actually allocated physical address.

Returns

Syscall return tag

Kernel-user memory (*ku_mem*) is memory that is shared between the kernel and user-space. It is needed for the UTCB area of threads (see [l4_thread_control_bind\(\)](#)) and for (extended) vCPU state. Note that existing kernel-user memory cannot be unmapped or mapped somewhere else.

Note

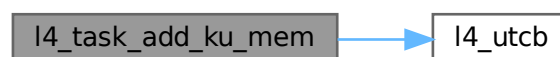
The amount of kernel-user memory that can be allocated at once is limited by the used kernel implementation. The minimum allocatable amount is one page ([L4_PAGESIZE](#)). A portable implementation should not depend on allocations greater than 16KiB to succeed.

This function is only guaranteed to work on [L4::Task](#) objects. It might or might not work on [L4::Vm](#) objects or on [L4Re::Dma_space](#) objects but there is no practical use for adding kernel-user memory to [L4::Vm](#) objects or to [L4Re::Dma_space](#) objects.

Definition at line 497 of file [task.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.12.3.2 l4_task_cap_equal()

```
l4_msgtag_t l4_task_cap_equal (
    l4_cap_idx_t task,
    l4_cap_idx_t cap_a,
    l4_cap_idx_t cap_b) [inline]
```

Test whether two capabilities point to the same object with the same permissions (only considering selected permissions).

Parameters

<i>task</i>	Capability selector for the destination task to do the lookup in.
<i>cap↔ _a</i>	Capability selector for the first capability to compare.
<i>cap↔ _b</i>	Capability selector for the second capability to compare.

Return values

<i>l4_msgtag_t::label()</i> = 1	The compared capabilities point to the same object with same considered permission.
<i>l4_msgtag_t::label()</i> = 0	The compared capabilities do not point to the same object or differ in the considered permission.

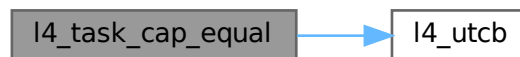
- For [L4::lpc_gate](#) objects, only the permissions [L4_CAP_FPAGE_W](#), [L4_CAP_FPAGE_S](#), and [L4_FPAGE_C_OBJ_RIGHT1](#) are considered for the comparison. Differences in other permissions are ignored.
- For other objects, only the permissions [L4_CAP_FPAGE_W](#) and [L4_CAP_FPAGE_S](#) are considered for the comparison. Differences in other permissions are ignored.

Note that having the [L4_CAP_FPAGE_R](#) permission is implicit in possessing the capability.

Definition at line 490 of file [task.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.12.3.3 l4_task_cap_valid()

```

l4_msgtag_t l4_task_cap_valid (
    l4_cap_idx_t task,
    l4_cap_idx_t cap) [inline]
  
```

Check whether a capability is present (refers to an object).

Parameters

<i>task</i>	Task to check the capability in.
<i>cap</i>	Valid capability to check for presence.

Return values

<code>l4_msgtag_t::label() > 0</code>	Capability is present (refers to an object).
<code>l4_msgtag_t::label() == 0</code>	No capability present (void object).

A capability is considered present when it refers to an existing kernel object.

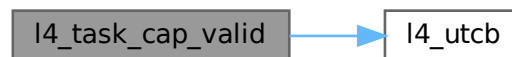
Precondition

`cap` must be a valid capability index (i.e. not `L4_INVALID_CAP` or the like).

Definition at line 484 of file [task.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.12.3.4 l4_task_delete_obj()

```

l4_msgtag_t l4_task_delete_obj (
    l4_cap_idx_t task,
    l4_cap_idx_t obj) [inline]
  
```

Release capability and delete object.

Parameters

<i>task</i>	Capability selector of destination task.
<i>obj</i>	Capability index of the object to delete.

Returns

Syscall return tag

If `obj` has the delete permission, initiates the deletion of the object. This implies that all capabilities for that object are gone afterwards. However, kernel-internally, objects are not destroyed until all other kernel objects holding a reference to it drop the reference. Hence, quota used by that object might not be freed immediately.

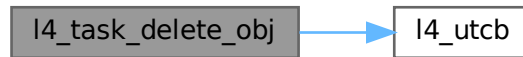
If `obj` does not have the delete permission, no error will be reported and only the capability `obj` is removed. (Note that, depending on the object's reference counter, this might still imply initiation of deletion.)

This operation is equivalent to [l4_task_unmap\(\)](#) with `L4_FP_DELETE_OBJ` flag.

Definition at line 463 of file [task.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.12.3.5 l4_task_map()

```

l4_msgtag_t l4_task_map (
    l4_cap_idx_t dst_task,
    l4_cap_idx_t src_task,
    l4_fpage_t snd_fpage,
    l4_umword_t snd_base) [inline]
  
```

Map resources available in the source task to a destination task.

Parameters

<i>dst_task</i>	Capability selector of the destination task.
<i>src_task</i>	Capability selector of the source task.
<i>snd_fpage</i>	Send flexpage that describes an area in the address space or object space of the source task.
<i>snd_base</i>	Send base that describes an offset in the receive window of the destination task. The lower bits contain additional map control flags (see l4_fpage_cacheability_opt_t for memory mappings, L4_obj_fpage_ctl for object mappings, and L4_MAP_ITEM_GRANT ; also see l4_map_control() and l4_map_obj_control()).

Returns

Syscall return tag. The function [l4_error\(\)](#) shall be used to test if the map operation was successful.

Return values

<i>L4_EOK</i>	Operation successful (but see notes below).
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i>-L4_EINVAL</i>	Invalid source task capability.
<i>-L4_IPC_SEMAPFAILED</i>	The map operation failed due to limited quota.

Precondition

The capability `dst_task` must have the permission [L4_CAP_FPAGE_W](#).

This method allows for asynchronous transfer of capabilities, memory mappings, and IO-port mappings (on IA32) from one task to another. The receive window is the whole address space of `dst_task`. By specifying proper rights in `snd_fpage` and `snd_base`, it is possible to remove rights during transfer.

Note

If the send flexpage is of type [L4_FPAGE_OBJ](#), the [L4_CAP_FPAGE_S](#) right is removed from the transferred capability unless both the source and destination task capabilities possess the [L4_CAP_FPAGE_S](#) right themselves.

Even with [l4_error\(\)](#) returning `L4_EOK` there might be cases where not all pages of the send flexpage were mapped respectively granted to the destination task, for instance, if the corresponding mapping in the destination task does already exist.

For more information on spaces and mappings, see [Spaces and Mappings](#). The flexpage API is described in more detail at [Flexpages](#).

Note

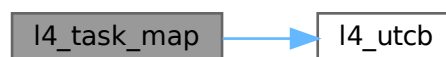
For peculiarities when using `grant`, see [L4_MAP_ITEM_GRANT](#).

Definition at line 433 of file [task.h](#).

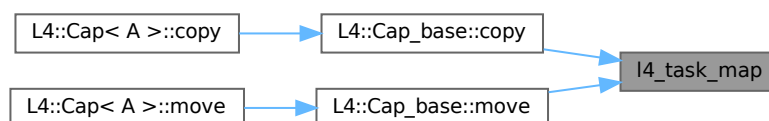
References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Referenced by [L4::Cap_base::copy\(\)](#), and [L4::Cap_base::move\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.10.12.3.6 l4_task_release_cap()

```
l4_msgtag_t l4_task_release_cap (  
    l4_cap_idx_t task,  
    l4_cap_idx_t cap) [inline]
```

Release object capability.

Parameters

<i>task</i>	Capability selector of destination task
<i>cap</i>	Capability selector of object to release

Returns

Syscall return tag

This operation unmaps the capability from the specified task. This operation is equivalent to unmapping a single object capability by specifying all object rights as unmap mask.

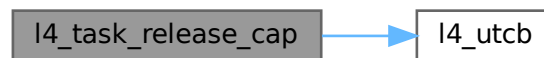
Note

If the reference counter of the kernel object referenced by `cap` goes down to zero, deletion of the object is initiated. Objects are not destroyed until all other kernel objects holding a reference to it drop the reference.

Definition at line 478 of file [task.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.12.3.7 l4_task_unmap()

```

l4_msgtag_t l4_task_unmap (
    l4_cap_idx_t task,
    l4_fpage_t fpage,
    l4_umword_t map_mask)  [inline]
  
```

Revoke rights from the task.

Parameters

<i>task</i>	Capability selector of destination task
<i>fpage</i>	Flexpage that describes an area in one capability space of the destination task and the rights to revoke.
<i>map_mask</i>	Unmap mask, see l4_unmap_flags_t

Returns

Syscall return tag

This method allows to revoke rights from the destination task. The rights to revoke are specified in the flexpage, see [l4_fpage_rights\(\)](#). For a flexpage describing IO ports or memory, it also revokes rights from all the tasks that got the rights delegated from the destination task (i.e., this operation does a recursive rights revocation). The capability is unmapped if certain rights are specified, see below for details. It is guaranteed that the rights revocation is completed before this function returns.

Note that this function cannot be used to revoke the reference counting permission (see [L4_FPAGE_C_REF_CNT](#)) or the IPC-gate server permission (see [L4_FPAGE_C_IPCGATE_SVR](#)) from object capabilities.

It depends on the platform and the object type which rights need to be specified in the `rights` field of `fpage` to unmap a capability:

- An object capability is unmapped if and only if the [L4_CAP_FPAGE_R](#) right bit is set.
- An IO port is unmapped if and only if any right bit is set.
- Memory is unmapped if and only if the [L4_FPAGE_RO](#) right bit is set.

Note

Depending on the page-table features supported by the hardware, revocation of certain rights from a memory capability can be a no-op (i.e., the rights are not revoked). Further, revocation of certain rights may grant other rights which were not present before. For instance, on an architecture without support for NX, revoking X does nothing. For another example, revoking only X from an execute-only page grants read permission (because the mapping remains present in the page table).

If the reference counter of a kernel object referenced in `fpage` goes down to zero (as a result of deleting capabilities), the deletion of the object is initiated. Objects are not destroyed until all other kernel objects holding a reference to it drop the reference.

Examples

[examples/sys/utcb-ipc/main.c](#).

Definition at line [440](#) of file [task.h](#).

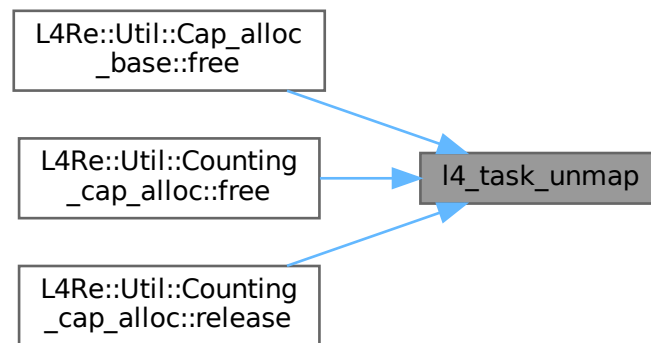
References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Referenced by [L4Re::Util::Cap_alloc_base::free\(\)](#), [L4Re::Util::Counting_cap_alloc< COUNTERTYPE, Dbg >::free\(\)](#), and [L4Re::Util::Counting_cap_alloc< COUNTERTYPE, Dbg >::release\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.10.12.3.8 l4_task_unmap_batch()

```

l4_msgtag_t l4_task_unmap_batch (
    l4_cap_idx_t task,
    l4_fpage_t const * fpages,
    unsigned num_fpages,
    l4_umword_t map_mask) [inline]
  
```

Revoke rights from a task.

Parameters

<i>task</i>	Capability selector of destination task
<i>fpages</i>	An array of flexpages. Each item describes an area in one capability space of the destination task.
<i>num_fpages</i>	The size of the fpages array in elements (number of fpages sent).
<i>map_mask</i>	Unmap mask, see l4_unmap_flags_t

Returns

Syscall return tag

Revoke rights specified in an array of flexpages, see [l4_task_unmap](#) for details.

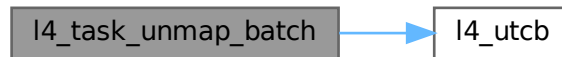
Precondition

The caller needs to take care that `num_fpages` is not bigger than `L4_UTCB_GENERIC_DATA_SIZE - 2`.

Definition at line 447 of file [task.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:

**14.1.10.12.3.9 l4_task_vgicc_map()**

```

l4_msgtag_t l4_task_vgicc_map (
    l4_cap_idx_t task,
    l4_fpage_t vgicc_fpage) [inline]
  
```

Map the GIC virtual CPU interface page to the task in case Fiasco detected a GIC version 2.

Parameters

<i>task</i>	Capability selector of destination task
<i>vgicc_fpage</i>	Flexpage that describes an area in the address space of the destination task to map the vGICC page to

Returns

Syscall return tag

Definition at line 46 of file [__task-arm.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

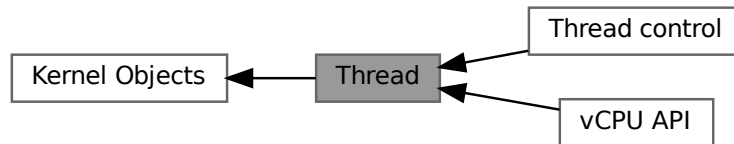
Here is the call graph for this function:



14.1.10.13 Thread

C Thread object interface, see [L4::Thread](#) for the C++ interface.

Collaboration diagram for Thread:



Topics

- Thread control ??
API for Thread Control method.
- vCPU API ??
vCPU API.

Enumerations

- enum [L4_thread_control_flags](#) { [L4_THREAD_CONTROL_SET_PAGER](#) = 0x0010000 , [L4_THREAD_CONTROL_BIND_TASK](#) = 0x0200000 , [L4_THREAD_CONTROL_ALIEN](#) = 0x0400000 , [L4_THREAD_CONTROL_SET_EXC_HANDLER](#) = 0x1000000 }
Flags for the thread control operation.
- enum [L4_thread_control_mr_indices](#) { [L4_THREAD_CONTROL_MR_IDX_FLAGS](#) = 0 , [L4_THREAD_CONTROL_MR_IDX_PAGER](#) = 1 , [L4_THREAD_CONTROL_MR_IDX_EXC_HANDLER](#) = 2 , [L4_THREAD_CONTROL_MR_IDX_FLAG_VALS](#) = 4 , [L4_THREAD_CONTROL_MR_IDX_BIND_UTCB](#) = 5 , [L4_THREAD_CONTROL_MR_IDX_BIND_TASK](#) = 6 }
Indices for the values in the message register for thread control.
- enum [L4_thread_ex_regs_flags](#) { [L4_THREAD_EX_REGS_CANCEL](#) = 0x10000UL , [L4_THREAD_EX_REGS_TRIGGER_EXC](#) = 0x20000UL , [L4_THREAD_EX_REGS_ARCH_MASK](#) = 0xff000000UL }
- enum [L4_thread_ex_regs_flags_arm](#) { [L4_THREAD_EX_REGS_ARM_SET_EL_MASK](#) = 0x3 << 24 , [L4_THREAD_EX_REGS_ARM_SET_EL_KEEP](#) = 0x0 << 24 , [L4_THREAD_EX_REGS_ARM_SET_EL_EL0](#) = 0x1 << 24 , [L4_THREAD_EX_REGS_ARM_SET_EL_EL1](#) = 0x2 << 24 }
Arm specific [L4::Thread::ex_regs\(\)](#) flags.
- enum [L4_thread_ex_regs_flags_arm64](#) { [L4_THREAD_EX_REGS_ARM64_SET_EL_MASK](#) = 0x3 << 24 , [L4_THREAD_EX_REGS_ARM64_SET_EL_KEEP](#) = 0x0 << 24 , [L4_THREAD_EX_REGS_ARM64_SET_EL_EL0](#) = 0x1 << 24 , [L4_THREAD_EX_REGS_ARM64_SET_EL_EL1](#) = 0x2 << 24 }
Arm64 specific [L4::Thread::ex_regs\(\)](#) flags.

Functions

- `l4_msgtag_t l4_thread_ex_regs (l4_cap_idx_t thread, l4_addr_t ip, l4_addr_t sp, l4_umword_t flags) L4_NOTHROW`
Exchange basic thread registers.
- `l4_msgtag_t l4_thread_ex_regs_u (l4_cap_idx_t thread, l4_addr_t ip, l4_addr_t sp, l4_umword_t flags, l4_utcb_t *utcb) L4_NOTHROW`
Exchange basic thread registers.
- `l4_msgtag_t l4_thread_ex_regs_ret (l4_cap_idx_t thread, l4_addr_t *ip, l4_addr_t *sp, l4_umword_t *flags) L4_NOTHROW`
Exchange basic thread registers and return previous values.
- `l4_msgtag_t l4_thread_ex_regs_ret_u (l4_cap_idx_t thread, l4_addr_t *ip, l4_addr_t *sp, l4_umword_t *flags, l4_utcb_t *utcb) L4_NOTHROW`
Exchange basic thread registers and return previous values.
- `l4_msgtag_t l4_thread_yield (void) L4_NOTHROW`
Yield current time slice.
- `l4_msgtag_t l4_thread_switch (l4_cap_idx_t to_thread) L4_NOTHROW`
Switch to another thread (and donate the remaining time slice).
- `l4_msgtag_t l4_thread_stats_time (l4_cap_idx_t thread, l4_kernel_clock_t *us) L4_NOTHROW`
Get consumed time of thread in μ s.
- `l4_msgtag_t l4_thread_vcpu_resume_start (void) L4_NOTHROW`
vCPU return from event handler.
- `l4_msgtag_t l4_thread_vcpu_resume_commit (l4_cap_idx_t thread, l4_msgtag_t tag) L4_NOTHROW`
Commit vCPU resume.
- `l4_msgtag_t l4_thread_vcpu_control (l4_cap_idx_t thread, l4_addr_t vcpu_state) L4_NOTHROW`
Enable the vCPU feature for the thread.
- `l4_msgtag_t l4_thread_vcpu_control_u (l4_cap_idx_t thread, l4_addr_t vcpu_state, l4_utcb_t *utcb) L4_NOTHROW`
Enable the vCPU feature for the thread.
- `l4_msgtag_t l4_thread_vcpu_control_ext (l4_cap_idx_t thread, l4_addr_t ext_vcpu_state) L4_NOTHROW`
Enable the extended vCPU feature for the thread.
- `l4_msgtag_t l4_thread_vcpu_control_ext_u (l4_cap_idx_t thread, l4_addr_t ext_vcpu_state, l4_utcb_t *utcb) L4_NOTHROW`
Enable the extended vCPU feature for the thread.
- `l4_msgtag_t l4_thread_register_del_irq (l4_cap_idx_t thread, l4_cap_idx_t irq) L4_NOTHROW`
Register an IRQ that will trigger upon deletion events.
- `l4_msgtag_t l4_thread_modify_sender_start (void) L4_NOTHROW`
Start a thread sender modification sequence.
- `int l4_thread_modify_sender_add (l4_umword_t match_mask, l4_umword_t match, l4_umword_t del_bits, l4_umword_t add_bits, l4_msgtag_t *tag) L4_NOTHROW`
Add a modification pattern to a sender modification sequence.
- `l4_msgtag_t l4_thread_modify_sender_commit (l4_cap_idx_t thread, l4_msgtag_t tag) L4_NOTHROW`
Apply (commit) a sender modification sequence.
- `l4_msgtag_t l4_thread_register_doorbell_irq (l4_cap_idx_t thread, l4_cap_idx_t irq) L4_NOTHROW`
Register an IRQ that will trigger when a forwarded virtual interrupt is pending.
- `l4_msgtag_t l4_thread_arm_set_tpidruro (l4_cap_idx_t thread, l4_addr_t tpidruro) L4_NOTHROW`
Set the TPIDRURO thread specific register.

14.1.10.13.1 Detailed Description

C Thread object interface, see [L4::Thread](#) for the C++ interface.

An [L4](#) thread is a thread of execution in the [L4](#) context. Usually user-level and kernel threads are mapped 1:1 to each other. Thread kernel objects are created using a factory, see [Factory](#) ([l4_factory_create_thread\(\)](#)).

Amongst other things an [L4](#) thread encapsulates:

- CPU state
 - General-purpose registers
 - Program counter
 - Stack pointer
- FPU state
- Scheduling parameters, see the [Scheduler](#) API
- Execution state
 - Blocked, Runnable, Running

Thread objects provide an API for

- Thread configuration and manipulation
- Thread switching.

The thread control functions are used to control various aspects of a thread. See [l4_thread_control_start\(\)](#) for more information.

On ARM newly created threads run in EL0 by default and the exception level can be changed there with [ex_regs\(\)](#).

Include File

```
#include <l4/sys/thread.h>
```

For the C++ interface refer to [L4::Thread](#).

14.1.10.13.2 Enumeration Type Documentation

14.1.10.13.2.1 L4_thread_control_flags

```
enum L4_thread_control_flags
```

Flags for the thread control operation.

Enumerator

L4_THREAD_CONTROL_SET_PAGER	The pager will be given.
-----------------------------	--------------------------

L4_THREAD_CONTROL_BIND_TASK	The task to bind the thread to will be given.
L4_THREAD_CONTROL_ALIEN	Alien state of the thread is set.
L4_THREAD_CONTROL_SET_EXC_HANDLER	The exception handler of the thread will be given.

Definition at line 761 of file [thread.h](#).

14.1.10.13.2.2 L4_thread_control_mr_indices

enum [L4_thread_control_mr_indices](#)

Indices for the values in the message register for thread control.

Enumerator

L4_THREAD_CONTROL_MR_IDX_FLAGS	See also L4_thread_control_flags .
L4_THREAD_CONTROL_MR_IDX_PAGER	Index for pager cap.
L4_THREAD_CONTROL_MR_IDX_EXC_HANDLER	Index for exception handler.
L4_THREAD_CONTROL_MR_IDX_FLAG_VALS	Index for feature values.
L4_THREAD_CONTROL_MR_IDX_BIND_UTCB	Index for UTCB address for bind.
L4_THREAD_CONTROL_MR_IDX_BIND_TASK	Index for task flexpage for bind.

Definition at line 782 of file [thread.h](#).

14.1.10.13.2.3 L4_thread_ex_regs_flags

enum [L4_thread_ex_regs_flags](#)

Flags for the thread ex-regs operation.

Enumerator

L4_THREAD_EX_REGS_CANCEL	Cancel ongoing IPC in the thread.
L4_THREAD_EX_REGS_TRIGGER_EXCEPTION	Trigger artificial exception in thread.
L4_THREAD_EX_REGS_ARCH_MASK	Arch specific flags.

Definition at line 797 of file [thread.h](#).

14.1.10.13.2.4 L4_thread_ex_regs_flags_arm

enum [L4_thread_ex_regs_flags_arm](#)

Arm specific [L4::Thread::ex_regs\(\)](#) flags.

Only one option must be used in calls to [L4::Thread::ex_regs\(\)](#). Using more than one option results in undefined behaviour.

Enumerator

L4_THREAD_EX_REGS_ARM_SET_EL_MASK	Exception level set mask.
L4_THREAD_EX_REGS_ARM_SET_EL_KEEP	Keep current exception level of thread (default).
L4_THREAD_EX_REGS_ARM_SET_EL_EL0	Set exception level of thread to EL0 (usr mode).
L4_THREAD_EX_REGS_ARM_SET_EL_EL1	Set exception level of thread to EL1 (sys mode).

Definition at line 39 of file [thread.h](#).

14.1.10.13.2.5 L4_thread_ex_regs_flags_arm64

enum [L4_thread_ex_regs_flags_arm64](#)

Arm64 specific [L4::Thread::ex_regs\(\)](#) flags.

Only one option must be used in calls to [L4::Thread::ex_regs\(\)](#). Using more than one option results in undefined behaviour.

Enumerator

L4_THREAD_EX_REGS_ARM64_SET_EL_MASK	Exception level set mask.
L4_THREAD_EX_REGS_ARM64_SET_EL_KEEP	Keep current exception level of thread (default).
L4_THREAD_EX_REGS_ARM64_SET_EL_EL0	Set exception level of thread to EL0.
L4_THREAD_EX_REGS_ARM64_SET_EL_EL1	Set exception level of thread to EL1t.

Definition at line 46 of file [thread.h](#).

14.1.10.13.3 Function Documentation

14.1.10.13.3.1 l4_thread_arm_set_tpidruro()

```
l4_msgtag_t l4_thread_arm_set_tpidruro (
    l4_cap_idx_t thread,
    l4_addr_t tpidruro) [inline]
```

Set the TPIDRURO thread specific register.

Parameters

<i>thread</i>	Thread to manipulate
<i>tpidruro</i>	The value to be set

Returns

System call return tag

Note

When this function is invoked for a thread currently executing on a different core, then the changed register content will not be visible to that thread until a thread switch happens on that core.

Definition at line 72 of file [thread.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:

**14.1.10.13.3.2 l4_thread_ex_regs()**

```

l4_msgtag_t l4_thread_ex_regs (
    l4_cap_idx_t thread,
    l4_addr_t ip,
    l4_addr_t sp,
    l4_umword_t flags) [inline]
  
```

Exchange basic thread registers.

Parameters

<i>thread</i>	Capability selector of the thread to manipulate.
<i>ip</i>	New instruction pointer, use ~0UL to leave the instruction pointer unchanged.
<i>sp</i>	New stack pointer, use ~0UL to leave the stack pointer unchanged.
<i>flags</i>	Ex-regs flags, see L4_thread_ex_regs_flags .

Returns

System call return tag

This method allows to manipulate and start a thread. The basic functionality is to set the instruction pointer and the stack pointer of a thread. Additionally, this method allows also to cancel ongoing IPC operations and to force the thread to raise an artificial exception (see [flags](#)). If the thread is in an IPC operation or if [L4_THREAD_EX_REGS_TRIGGER_EXCEPTION](#) forces an IPC then changes in IP and SP take effect directly after returning from this IPC. On ARM this method allows to change the exception level, see [L4_thread_ex_regs_flags_arm](#) and [L4_thread_ex_regs_flags_arm64](#).

The thread is started using [l4_scheduler_run_thread\(\)](#). However, if at the time [l4_scheduler_run_thread\(\)](#) is called, the instruction pointer of the thread is invalid, a later call to [l4_thread_ex_regs\(\)](#) with a valid instruction pointer might start the thread.

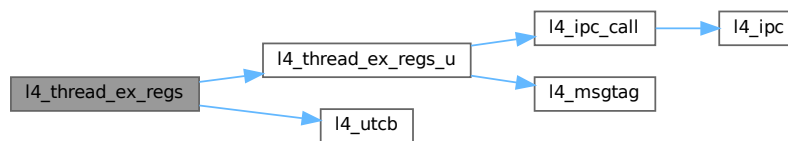
Examples

[examples/sys/aliens/main.c](#), [examples/sys/singlestep/main.c](#), [examples/sys/start-with-exc/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

Definition at line 942 of file [thread.h](#).

References [L4_NOTHROW](#), [l4_thread_ex_regs_u\(\)](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.13.3.3 l4_thread_ex_regs_ret()

```

l4_msgtag_t l4_thread_ex_regs_ret (
    l4_cap_idx_t thread,
    l4_addr_t * ip,
    l4_addr_t * sp,
    l4_umword_t * flags) [inline]
  
```

Exchange basic thread registers and return previous values.

Parameters

	<i>thread</i>	Capability selector of the thread to manipulate.
in, out	<i>ip</i>	New instruction pointer, use ~0UL to leave the instruction pointer unchanged, return previous instruction pointer.
in, out	<i>sp</i>	New stack pointer, use ~0UL to leave the stack pointer unchanged, returns previous stack pointer.
in, out	<i>flags</i>	Ex-regs flags, see L4_thread_ex_regs_flags , return previous CPU flags of the thread.

Returns

System call return tag

This method allows to manipulate and start a thread. The basic functionality is to set the instruction pointer and the stack pointer of a thread. Additionally, this method allows also to cancel ongoing IPC operations and to force the thread to raise an artificial exception (see `flags`). If the thread is in an IPC operation or if [L4_THREAD_EX_REGS_TRIGGER_EXCEPTION](#) forces an IPC then changes in IP and SP take effect directly after returning from this IPC. On ARM this method allows to change the exception level, see [L4_thread_ex_regs_flags_arm](#) and [L4_thread_ex_regs_flags_arm64](#).

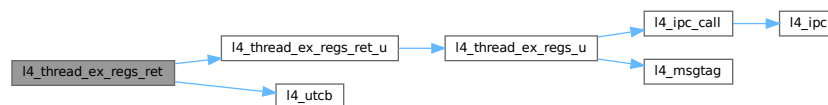
The thread is started using `l4_scheduler_run_thread()`. However, if at the time `l4_scheduler_run_thread()` is called, the instruction pointer of the thread is invalid, a later call to `l4_thread_ex_regs()` with a valid instruction pointer might start the thread.

Returned values are valid only if function returns successfully.

Definition at line 949 of file `thread.h`.

References `L4_NOTHROW`, `l4_thread_ex_regs_ret_u()`, and `l4_utcb()`.

Here is the call graph for this function:



14.1.10.13.3.4 l4_thread_ex_regs_ret_u()

```

l4_msgtag_t l4_thread_ex_regs_ret_u (
    l4_cap_idx_t thread,
    l4_addr_t * ip,
    l4_addr_t * sp,
    l4_umword_t * flags,
    l4_utcb_t * utcb) [inline]
  
```

Exchange basic thread registers and return previous values.

Parameters

	<i>thread</i>	Capability selector of the thread to manipulate.
in, out	<i>ip</i>	New instruction pointer, use <code>~0UL</code> to leave the instruction pointer unchanged, return previous instruction pointer.
in, out	<i>sp</i>	New stack pointer, use <code>~0UL</code> to leave the stack pointer unchanged, returns previous stack pointer.
in, out	<i>flags</i>	Ex-regs flags, see <code>L4_thread_ex_regs_flags</code> , return previous CPU flags of the thread.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to <code>l4_utcb</code> .

Returns

System call return tag. [out] parameters are only valid if the function returns successfully. Use `l4_error()` to check.

This method allows to manipulate and start a thread. The basic functionality is to set the instruction pointer and the stack pointer of a thread. Additionally, this method allows also to cancel ongoing IPC operations and to force the thread to raise an artificial exception (see `flags`). If the thread is in an IPC operation or if `L4_THREAD_EX_REGS_TRIGGER_EXCEPTION` forces an IPC then changes in IP and SP take effect directly after returning from this IPC. On ARM this method allows to change the exception level, see `L4_thread_ex_regs_flags_arm` and `L4_thread_ex_regs_flags_arm64`.

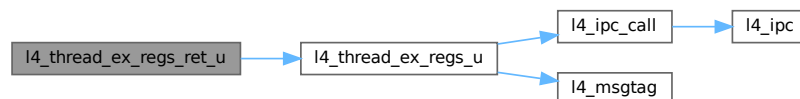
The thread is started using `L4::Scheduler::run_thread()`. However, if at the time `L4::Scheduler::run_thread()` is called, the instruction pointer of the thread is invalid, a later call to `ex_regs()` with a valid instruction pointer might start the thread.

Definition at line 823 of file `thread.h`.

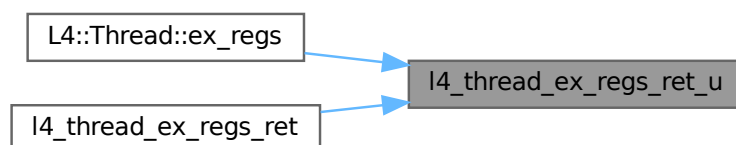
References `L4_NOTHROW`, `l4_thread_ex_regs_u()`, and `l4_msg_regs_t::mr`.

Referenced by `L4::Thread::ex_regs()`, and `l4_thread_ex_regs_ret()`.

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.10.13.3.5 l4_thread_ex_regs_u()

```

l4_msgtag_t l4_thread_ex_regs_u (
    l4_cap_idx_t thread,
    l4_addr_t ip,
    l4_addr_t sp,
    l4_umword_t flags,
    l4_utcb_t * utcb) [inline]
  
```

Exchange basic thread registers.

Parameters

<i>thread</i>	Capability selector of the thread to manipulate.
<i>ip</i>	New instruction pointer, use <code>~0UL</code> to leave the instruction pointer unchanged.
<i>sp</i>	New stack pointer, use <code>~0UL</code> to leave the stack pointer unchanged.
<i>flags</i>	Ex-regs flags, see L4_thread_ex_regs_flags .
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

System call return tag.

This method allows to manipulate and start a thread. The basic functionality is to set the instruction pointer and the stack pointer of a thread. Additionally, this method allows also to cancel ongoing IPC operations and to force the thread to raise an artificial exception (see `flags`). If the thread is in an IPC operation or if [L4_THREAD_EX_REGS_TRIGGER_EXCEPTION](#) forces an IPC then changes in IP and SP take effect directly after returning from this IPC. On ARM this method allows to change the exception level, see [L4_thread_ex_regs_flags_arm](#) and [L4_thread_ex_regs_flags_arm64](#).

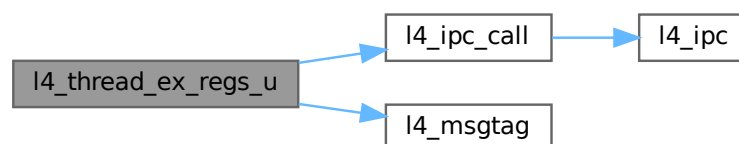
The thread is started using [L4::Scheduler::run_thread\(\)](#). However, if at the time [L4::Scheduler::run_thread\(\)](#) is called, the instruction pointer of the thread is invalid, a later call to `ex_regs()` with a valid instruction pointer might start the thread.

Definition at line 812 of file [thread.h](#).

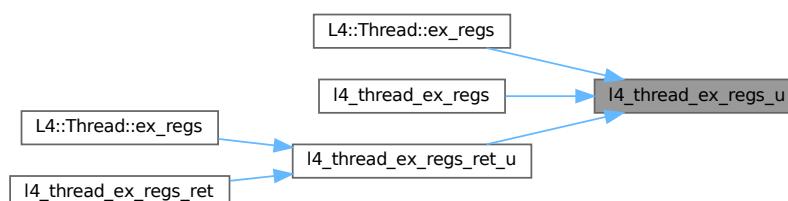
References [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), [l4_msgtag\(\)](#), [L4_NOTHROW](#), [L4_PROTO_THREAD](#), [L4_THREAD_EX_REGS_OP](#), and [l4_msg_regs_t::mr](#).

Referenced by [L4::Thread::ex_regs\(\)](#), [l4_thread_ex_regs\(\)](#), and [l4_thread_ex_regs_ret_u\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.10.13.3.6 l4_thread_modify_sender_add()

```
int l4_thread_modify_sender_add (
    l4_umword_t match_mask,
    l4_umword_t match,
    l4_umword_t del_bits,
    l4_umword_t add_bits,
    l4_msgtag_t * tag) [inline]
```

Add a modification pattern to a sender modification sequence.

Parameters

<i>tag</i>	Tag received from l4_thread_modify_sender_start() or previous l4_thread_modify_sender_add() calls from the same sequence.
<i>match_mask</i>	Bitmask of bits to match the label.
<i>match</i>	Bitmask that must be equal to the label after applying <i>match_mask</i> .
<i>del_bits</i>	Bits to be deleted from the label.
<i>add_bits</i>	Bits to be added to the label.

Returns

0 on success, <0 on error

In pseudo code: if ((sender_label & match_mask) == match) { sender_label = (sender_label & ~del_bits) | add_bits; }

Only the first match is applied.

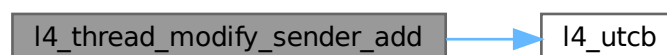
See also

[l4_thread_modify_sender_start](#)
[l4_thread_modify_sender_commit](#)

Definition at line 1116 of file [thread.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.13.3.7 `l4_thread_modify_sender_commit()`

```
l4_msgtag_t l4_thread_modify_sender_commit (
    l4_cap_idx_t thread,
    l4_msgtag_t tag) [inline]
```

Apply (commit) a sender modification sequence.

The modification rules are applied to all IPCs to the thread (whether directly or by IPC gate) that are already in flight, that is that the sender is already blocking on.

Note

Modifying the senders of a thread running on a different CPU core is not supported.

To ensure that no in-flight senders are missed, either the thread itself must execute `modify_senders`, or the thread executing the `modify_senders` must synchronize with the target thread. This synchronization must ensure the following:

1. Before `modify_senders` is executed the target thread must execute at least shortly (so that pending DRQs are handled).
2. The target thread must pause its IPC dispatch, until `modify_senders` is completed. In other words, the target thread must not be receive ready, because otherwise an IPC message with an unmodified label can be transferred to its UTCB or vCPU state.

See also

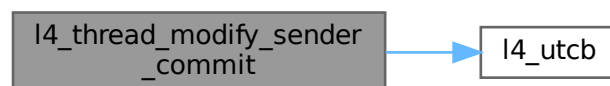
[l4_thread_modify_sender_start](#)

[l4_thread_modify_sender_add](#)

Definition at line 1127 of file [thread.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.13.3.8 `l4_thread_modify_sender_start()`

```
l4_msgtag_t l4_thread_modify_sender_start (
    void ) [inline]
```

Start a thread sender modification sequence.

Add modification rules with [l4_thread_modify_sender_add\(\)](#) and commit with [l4_thread_modify_sender_commit\(\)](#). Do not touch the UTCB between [l4_thread_modify_sender_start\(\)](#) and [l4_thread_modify_sender_commit\(\)](#).

This mechanism shall be used to change the source object labels of every pending IPC of an IPC gate or an IRQ if the labels in such pending IPC become invalid for the receiving thread, potentially because:

- an IPC gate / IRQ was unbound from a thread, or
- an IPC gate / IRQ was removed, or
- the label of an IPC gate /IRQ bound to a thread was changed.

It is not required to perform the `modify_sender` mechanism after an IPC gate or an IRQ was bound to a thread for the first time.

See also

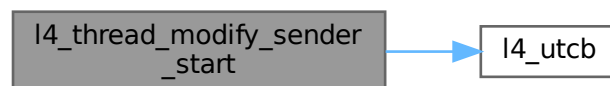
[l4_thread_modify_sender_add](#)

[l4_thread_modify_sender_commit](#)

Definition at line 1110 of file [thread.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.13.3.9 `l4_thread_register_del_irq()`

```
l4_msgtag_t l4_thread_register_del_irq (
    l4_cap_idx_t thread,
    l4_cap_idx_t irq) [inline]
```

Register an IRQ that will trigger upon deletion events.

Parameters

<i>thread</i>	Thread to register IRQ for.
<i>irq</i>	Capability selector for the IRQ object to be triggered.

Returns

System call return tag containing the return code.

Return values

<code>-L4_EPERM</code>	Insufficient permissions; see precondition.
------------------------	---

Precondition

The capability `irq` must have the permission `L4_CAP_FPAGE_W`.

In case the `irq` is already bound to an interrupt source, it is unbound first. When `irq` is deleted, it will be deregistered first. A registered deletion `Irq` can only be deregistered by deleting the `Irq` or the thread.

List of deletion events:

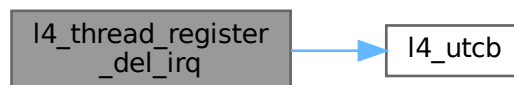
- deletion of one or several IPC gates bound to this thread.

When the deletion event is delivered, there is no indication about which IPC gate was deleted.

Definition at line 1037 of file `thread.h`.

References `L4_NOTHROW`, and `l4_utcb()`.

Here is the call graph for this function:



14.1.10.13.3.10 l4_thread_register_doorbell_irq()

```

l4_msgtag_t l4_thread_register_doorbell_irq (
    l4_cap_idx_t thread,
    l4_cap_idx_t irq) [inline]
  
```

Register an IRQ that will trigger when a forwarded virtual interrupt is pending.

Parameters

<i>thread</i>	Thread to register IRQ for.
<i>irq</i>	Capability selector for the IRQ object to be triggered.

Returns

System call return tag containing the return code.

Return values

<code>-L4_EPERM</code>	Insufficient permissions; see precondition.
------------------------	---

Precondition

The capability `irq` must have the permission [L4_CAP_FPAGE_W](#).

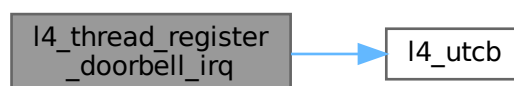
See [l4_irq_bind_vcpu\(\)](#) for more details about how interrupts can be forwarded directly by the kernel to extended vCPU user mode.

In case the `irq` is already bound to an interrupt source, it is unbound first. When `irq` is deleted, it will be deregistered first. A registered deletion `l4_irq` can only be deregistered by deleting the `l4_irq` or the thread.

Definition at line 1146 of file [thread.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.13.3.11 l4_thread_stats_time()

```

l4_msgtag_t l4_thread_stats_time (
    l4_cap_idx_t thread,
    l4_kernel_clock_t * us) [inline]
  
```

Get consumed time of thread in μ s.

Parameters

	<i>thread</i>	Thread to get the consumed time from.
out	<i>us</i>	Consumed time in μ s.

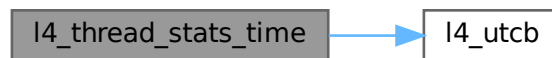
Returns

system call return tag

Definition at line 1005 of file [thread.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:

**14.1.10.13.3.12 l4_thread_switch()**

```
l4_msgtag_t l4_thread_switch (
    l4_cap_idx_t to_thread) [inline]
```

Switch to another thread (and donate the remaining time slice).

Parameters

<i>to_thread</i>	The thread to switch to.
------------------	--------------------------

Returns

system call return tag

Definition at line 996 of file [thread.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.13.3.13 `l4_thread_vcpu_control()`

```
l4_msgtag_t l4_thread_vcpu_control (
    l4_cap_idx_t thread,
    l4_addr_t vcpu_state) [inline]
```

Enable the vCPU feature for the thread.

Parameters

<i>thread</i>	Capability selector of the thread for which the vCPU feature shall be enabled.
<i>vcpu_state</i>	The virtual address where the kernel shall store the vCPU state in case of vCPU exits. The address must be a valid kernel-user-memory address (see l4_task_add_ku_mem()).

Returns

Syscall return tag.

This function enables the vCPU feature of the `thread`.

The kernel-user memory area starting at `vcpu_state` must be at least 128-byte aligned and must cover the size of [l4_vcpu_state_t](#).

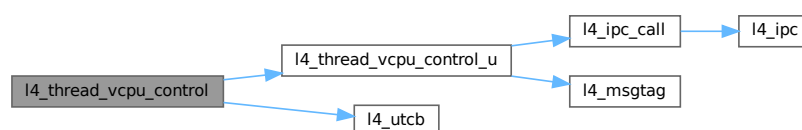
Note

Disabling of the vCPU feature is optional and currently not supported.

Definition at line 1054 of file [thread.h](#).

References [L4_NOTHROW](#), [l4_thread_vcpu_control_u\(\)](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.13.3.14 `l4_thread_vcpu_control_ext()`

```
l4_msgtag_t l4_thread_vcpu_control_ext (
    l4_cap_idx_t thread,
    l4_addr_t ext_vcpu_state) [inline]
```

Enable the extended vCPU feature for the thread.

Parameters

<i>thread</i>	Capability selector of the thread for which the extended vCPU feature shall be enabled.
<i>ext_vcpu_state</i>	The virtual address where the kernel shall store the vCPU state in case of vCPU exits. The address must be a valid kernel-user-memory address (see l4_task_add_ku_mem()).

Returns

Syscall return tag.

The extended vCPU feature allows the use of hardware-virtualization features such as Intel's VT or AMD's SVM.

This function enables the extended vCPU feature of the `thread`. Enabling the extended vCPU feature also enables the vCPU feature.

The kernel-user memory area starting at `ext_vcpu_state` must be at least 4 KiB aligned and must cover a size of `L4_PAGESIZE`. It includes the data of [l4_vcpu_state_t](#) at offset 0, the extended vCPU state at offset `L4_VCPU_OFFSET_EXT_STATE`, and, on some platforms, the extended vCPU information at offset `L4_VCPU_OFFSET_EXT_INFOS`.

Note

Enabling the extended vCPU feature for a thread running on a different CPU core is currently not supported.

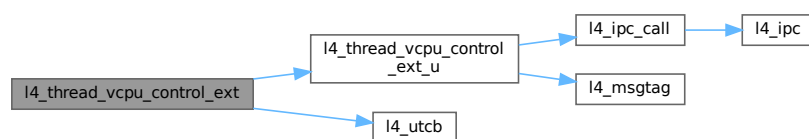
Disabling of the extended vCPU feature is currently not supported.

Upgrading from non-extended vCPU feature to extended vCPU feature is currently not supported.

Definition at line 1069 of file [thread.h](#).

References [L4_NOTHROW](#), [l4_thread_vcpu_control_ext_u\(\)](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.13.3.15 l4_thread_vcpu_control_ext_u()

```

l4_msgtag_t l4_thread_vcpu_control_ext_u (
    l4_cap_idx_t thread,
    l4_addr_t ext_vcpu_state,
    l4_utcb_t * utcb) [inline]

```

Enable the extended vCPU feature for the thread.

Parameters

<i>thread</i>	Capability selector of the thread for which the extended vCPU feature shall be enabled.
<i>ext_vcpu_state</i>	The virtual address where the kernel shall store the vCPU state in case of vCPU exits. The address must be a valid kernel-user-memory address (see L4::Task::add_ku_mem()).
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

Syscall return tag.

The extended vCPU feature allows the use of hardware-virtualization features such as Intel's VT-x (VMX) or AMD's AMD-V (SVM).

This function enables the extended vCPU feature of `this` thread. Enabling the extended vCPU feature also enables the vCPU feature.

The kernel-user memory area starting at `ext_vcpu_state` must be at least 4 KiB aligned and must cover a size of `L4_PAGESIZE`. It includes the data of [l4_vcpu_state_t](#) at offset 0, the extended vCPU state at offset `L4_VCPU_OFFSET_EXT_STATE`, and, on some platforms, the extended vCPU information at offset `L4_VCPU_OFFSET_EXT_INFOS`.

On Intel's VT-x (VMX), the extended vCPU state is [l4_vm_vmx_vcpu_vmcs_t](#) and the extended vCPU information is [l4_vm_vmx_vcpu_infos_t](#). Furthermore, the extended vCPU state needs to be associated with a vCPU context (see [l4_vm_vmx_set_hw_vmcs\(\)](#)).

On AMD's AMD-V (SVM), the extended vCPU state is [l4_vm_svm_vmcb_t](#).

Note

Enabling the extended vCPU feature for a thread running on a different CPU core is currently not supported.

Disabling of the extended vCPU feature is currently not supported.

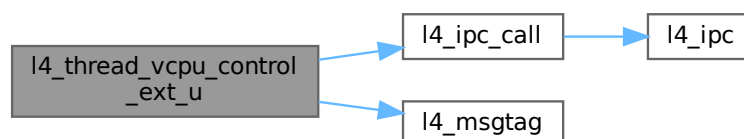
Upgrading from non-extended vCPU feature to extended vCPU feature is currently not supported.

Definition at line 1059 of file [thread.h](#).

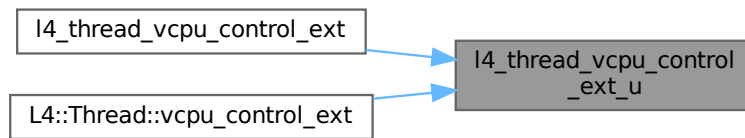
References [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), [l4_msgtag\(\)](#), [L4_NOTHROW](#), [L4_PROTO_THREAD](#), and [l4_msg_regs_t::mr](#).

Referenced by [l4_thread_vcpu_control_ext\(\)](#), and [L4::Thread::vcpu_control_ext\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.10.13.3.16 l4_thread_vcpu_control_u()

```
l4_msgtag_t l4_thread_vcpu_control_u (
    l4_cap_idx_t thread,
    l4_addr_t vcpu_state,
    l4_utcb_t * utcb) [inline]
```

Enable the vCPU feature for the thread.

Parameters

<i>thread</i>	Capability selector of the thread for which the vCPU feature shall be enabled.
<i>vcpu_state</i>	A virtual address pointing to a l4_vcpu_state_t . It must be a valid kernel-user-memory address (see L4::Task::add_ku_mem()).
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

Syscall return tag.

This function enables the vCPU feature of `this` thread

The kernel-user memory starting at `vcpu_state` must be at least 128-byte aligned and must cover the size of [l4_vcpu_state_t](#).

The asynchronous IPC handling is described at [vCPU API](#).

Note

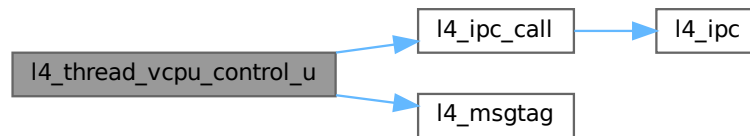
Disabling of the vCPU feature is optional and currently not supported.

Definition at line 1044 of file [thread.h](#).

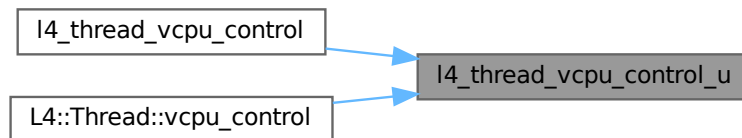
References [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), [l4_msgtag\(\)](#), [L4_NOTHROW](#), [L4_PROTO_THREAD](#), [L4_THREAD_VCPU_CONTROL_ON](#) and [l4_msg_regs_t::mr](#).

Referenced by [l4_thread_vcpu_control\(\)](#), and [L4::Thread::vcpu_control\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.10.13.3.17 l4_thread_vcpu_resume_commit()

```

l4_msgtag_t l4_thread_vcpu_resume_commit (
    l4_cap_idx_t thread,
    l4_msgtag_t tag) [inline]
  
```

Commit vCPU resume.

Parameters

<i>thread</i>	Thread to be resumed, the invalid cap can be used for the current thread.
<i>tag</i>	Tag to use, returned by l4_thread_vcpu_resume_start()

Returns

Syscall return tag containing one of the following return codes.

Return values

0	Indicates a VM exit, provided that <code>thread</code> is in extended vCPU mode with virtual interrupts cleared.
---	--

1	Indicates an incoming IPC message, provided that the <code>thread</code> is in extended vCPU mode with virtual interrupts cleared.
-L4_EPERM	The user task capability set in the vCPU state is missing the L4_CAP_FPAGE_S right. On Intel's VT-x (VMX): The vCPU context capability set in the extended vCPU state is missing the L4_CAP_FPAGE_S right.
-L4_ENOENT	The user task capability set in the vCPU state is invalid.
-L4_EINVAL	<code>thread</code> is not the current running thread, or does not have the vCPU feature enabled. On Intel's VT-x (VMX): No vCPU context associated with the extended vCPU state.
-L4_EBUSY	On Intel's VT-x (VMX): The vCPU context associated with the extended vCPU state is already active on a different CPU.
-L4_ENODEV	On Intel's VT-x (VMX): The vCPU context associated with the extended vCPU state cannot be initialized or activated.
<0	A supplied mapping failed.

All flexpages in the UTCB (added with [l4_sndfpage_add\(\)](#) after [l4_thread_vcpu_resume_start\(\)](#)) are unconditionally mapped into the user task configured in the vCPU state.

To resume into another address space, the capability to the target [Task](#) (or [L4::Vm](#)) must be set in [l4_vcpu_state_t::user_task](#) together with [L4_VCPU_F_USER_MODE](#). The capability selector must have all lower bits clear (see [L4_CAP_MASK](#)). The kernel adds the [L4_SYSF_SEND](#) flag there to indicate that the capability has been referenced in the kernel. Consecutive resumes will not reference the task capability again until all lower bits are cleared again. To release a task use a different task capability or use an invalid capability with the [L4_SYSF_REPLY](#) flag set.

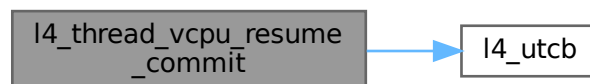
See also

[l4_vcpu_state_t](#)

Definition at line 1017 of file [thread.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.13.3.18 `l4_thread_vcpu_resume_start()`

```
l4_msgtag_t l4_thread_vcpu_resume_start (
    void ) [inline]
```

vCPU return from event handler.

Returns

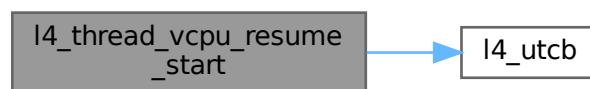
Message tag to be used for [l4_sndfpage_add\(\)](#) and [l4_thread_vcpu_resume_commit\(\)](#)

The vCPU resume functionality is split in multiple functions to allow the specification of additional send-flexpages using [l4_sndfpage_add\(\)](#).

Definition at line 1011 of file [thread.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:

**14.1.10.13.3.19 l4_thread_yield()**

```
l4_msgtag_t l4_thread_yield (  
    void ) [inline]
```

Yield current time slice.

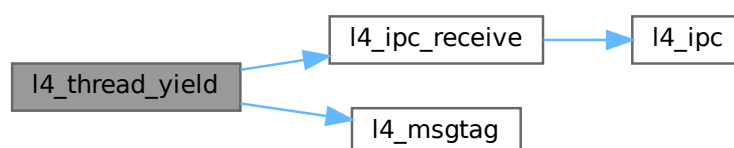
Returns

system call return tag

Definition at line 891 of file [thread.h](#).

References [L4_INVALID_CAP](#), [L4_IPC_BOTH_TIMEOUT_0](#), [l4_ipc_receive\(\)](#), [l4_msgtag\(\)](#), and [L4_NOTHROW](#).

Here is the call graph for this function:



14.1.10.13.4 Thread control

API for Thread Control method.

Collaboration diagram for Thread control:



Functions

- void [l4_thread_control_start](#) (void) [L4_NOTHROW](#)
Start a thread control API sequence.
- void [l4_thread_control_pager](#) ([l4_cap_idx_t](#) pager) [L4_NOTHROW](#)
Set the pager.
- void [l4_thread_control_exc_handler](#) ([l4_cap_idx_t](#) exc_handler) [L4_NOTHROW](#)
Set the exception handler.
- void [l4_thread_control_bind](#) ([l4_utcb_t](#) *thread_utcb, [l4_cap_idx_t](#) task) [L4_NOTHROW](#)
Bind the thread to a task.
- void [l4_thread_control_alien](#) (int on) [L4_NOTHROW](#)
Enable alien mode.
- [l4_msgtag_t](#) [l4_thread_control_commit](#) ([l4_cap_idx_t](#) thread) [L4_NOTHROW](#)
Commit the thread control parameters.

14.1.10.13.4.1 Detailed Description

API for Thread Control method.

The thread control API provides access to almost any parameter of a thread object. The API is based on a single invocation of the thread object. However, because of the huge amount of parameters, the API provides a set of functions to set specific parameters of a thread and a commit function to commit the thread control call (see [l4_thread_control_commit\(\)](#)).

A thread control operation must always start with [l4_thread_control_start\(\)](#) and be committed with [l4_thread_control_commit\(\)](#). All other thread control parameter setter functions must be called between these two functions.

An example for a sequence of thread control API calls can be found below.

```

l4_thread_control_start();
l4_thread_control_pager(pager_cap);
l4_thread_control_bind (thread_utcb, task);
l4_thread_control_commit(thread_cap);
  
```

14.1.10.13.4.2 Function Documentation

l4_thread_control_alien()

```
void l4_thread_control_alien (  
    int on) [inline]
```

Enable alien mode.

Parameters

<i>on</i>	Boolean value defining the state of the feature.
-----------	--

For a thread in alien mode the kernel produces just an exception IPC for each IPC and exception caused by the alien thread instead of handling these events regularly. (Page faults of alien threads and interrupts occurring while the alien thread is running are always handled regularly.) While the alien thread is blocking, the exception handler can inspect and modify the state of the alien thread and potentially also the system call arguments. If the exception handler replies with [L4_PROTO_ALLOW_SYSCALL](#) as message tag, the kernel handles the next IPC or exception of the alien thread in a regular way. If the exception handler leaves certain thread state unchanged (in particular the instruction pointer), this will be the IPC or exception that caused the call of the exception handler. For a regularly processed IPC or exception of the alien thread the kernel also performs an exception IPC on kernel exit.

This feature can be used to attach a debugger to a thread and trace all object invocations and their results. It could also be used to handle other systems that use the same syscall instruction as [L4Re](#).

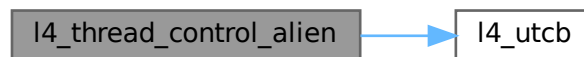
Examples

[examples/sys/aliens/main.c](#), and [examples/sys/singlestep/main.c](#).

Definition at line 981 of file [thread.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



`l4_thread_control_bind()`

```
void l4_thread_control_bind (
    l4_utcb_t * thread_utcb,
    l4_cap_idx_t task) [inline]
```

Bind the thread to a task.

Parameters

<i>thread_utcb</i>	The thread's UTCB address within the task it shall be bound to. The address must be aligned (architecture dependent; at least word aligned) and it must point to at least L4_UTCB_OFFSET bytes of kernel-user memory.
<i>task</i>	The task the thread shall be bound to.

Precondition

The thread must not be bound to a task yet.

The capability `task` must have the permission [L4_CAP_FPAGE_S](#), otherwise the later call to [l4_thread_control_commit\(\)](#) will fail with [L4_EPERM](#).

A thread may execute code in the context of a task if and only if the thread is bound to the task. To actually start execution, [l4_thread_ex_regs\(\)](#) needs to be used. Execution in the context of the task means that the code has access to all the task's resources (and only those). The executed code itself must be one of those resources. A thread can be bound at most once to a task.

Note

The UTCBs of different threads in the same task should not overlap in order to prevent data corruption.

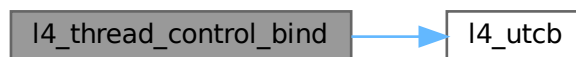
Examples

[examples/sys/aliens/main.c](#), [examples/sys/singlestep/main.c](#), [examples/sys/start-with-exc/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

Definition at line 975 of file [thread.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:

**`l4_thread_control_commit()`**

```
l4_msgtag_t l4_thread_control_commit (
    l4_cap_idx_t thread) [inline]
```

Commit the thread control parameters.

Parameters

<i>thread</i>	Capability selector of target thread to commit to.
---------------	--

Returns

Syscall return tag containing one of the following return codes.

Return values

<i>L4_EOK</i>	Operation successful.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i>-L4_EINVAL</i>	Malformed thread control parameters.

Precondition

The capability `thread` must have the permission `L4_CAP_FPAGE_S`. When using `l4_thread_control_bind()`, also the respective task capability must have the permission `L4_CAP_FPAGE_S`.

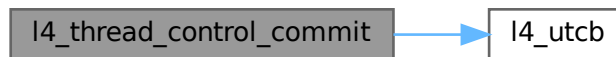
Examples

[examples/sys/aliens/main.c](#), [examples/sys/singlestep/main.c](#), [examples/sys/start-with-exc/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

Definition at line 987 of file [thread.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



`l4_thread_control_exc_handler()`

```
void l4_thread_control_exc_handler (
    l4_cap_idx_t exc_handler) [inline]
```

Set the exception handler.

Parameters

<i>exc_handler</i>	Capability selector invoked to send an exception IPC.
--------------------	---

Note

The exception-handler capability selector is interpreted in the task the thread is bound to (executes in).

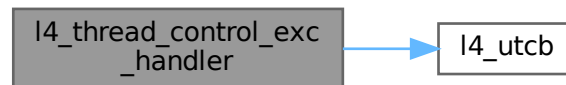
Examples

[examples/sys/aliens/main.c](#), [examples/sys/singlestep/main.c](#), [examples/sys/start-with-exc/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

Definition at line 968 of file [thread.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:

**l4_thread_control_pager()**

```
void l4_thread_control_pager (
    l4_cap_idx_t pager) [inline]
```

Set the pager.

Parameters

<i>pager</i>	Capability selector invoked to send a page-fault IPC.
--------------	---

Note

The pager capability selector is interpreted in the task the thread is bound to (executes in).

Examples

[examples/sys/aliens/main.c](#), [examples/sys/singlestep/main.c](#), [examples/sys/start-with-exc/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

Definition at line 962 of file [thread.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



l4_thread_control_start()

```
void l4_thread_control_start (
    void ) [inline]
```

Start a thread control API sequence.

This function starts a sequence of thread control API functions. After this functions any of following functions may be called in any order.

- [l4_thread_control_pager\(\)](#)
- [l4_thread_control_exc_handler\(\)](#)
- [l4_thread_control_bind\(\)](#)
- [l4_thread_control_alien\(\)](#)

To commit the changes to the thread [l4_thread_control_commit\(\)](#) must be called in the end.

Note

The thread control API calls store the parameters for the thread in the UTCB of the caller (see [l4_utcb\(\)](#)), this means between [l4_thread_control_start\(\)](#) and [l4_thread_control_commit\(\)](#) no functions that modify the UTCB contents must be called.

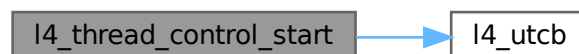
Examples

[examples/sys/aliens/main.c](#), [examples/sys/singlestep/main.c](#), [examples/sys/start-with-exc/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

Definition at line 956 of file [thread.h](#).

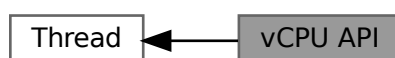
References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:

**14.1.10.13.5 vCPU API**

vCPU API.

Collaboration diagram for vCPU API:



Data Structures

- struct [l4_vcpu_state_t](#)
State of a vCPU.
- struct [l4_vcpu_regs_t](#)
vCPU registers.
- struct [l4_vcpu_ipc_regs_t](#)
vCPU message registers.

Typedefs

- typedef struct [l4_vcpu_state_t](#) [l4_vcpu_state_t](#)
State of a vCPU.
- typedef struct [l4_vcpu_regs_t](#) [l4_vcpu_regs_t](#)
vCPU registers.
- typedef struct [l4_vcpu_ipc_regs_t](#) [l4_vcpu_ipc_regs_t](#)
vCPU message registers.
- typedef [l4_exc_regs_t](#) [l4_vcpu_regs_t](#)
vCPU registers.
- typedef struct [l4_vcpu_ipc_regs_t](#) [l4_vcpu_ipc_regs_t](#)
vCPU message registers.
- typedef struct [l4_vcpu_regs_t](#) [l4_vcpu_regs_t](#)
vCPU registers.
- typedef struct [l4_vcpu_ipc_regs_t](#) [l4_vcpu_ipc_regs_t](#)
vCPU message registers.
- typedef struct [l4_vcpu_regs_t](#) [l4_vcpu_regs_t](#)
vCPU registers.
- typedef struct [l4_vcpu_ipc_regs_t](#) [l4_vcpu_ipc_regs_t](#)
vCPU message registers.

Enumerations

- enum [L4_vcpu_state_flags](#) {
[L4_VCPU_F_IRQ](#) = 0x01 , [L4_VCPU_F_PAGE_FAULTS](#) = 0x02 , [L4_VCPU_F_EXCEPTIONS](#) = 0x04 ,
[L4_VCPU_F_USER_MODE](#) = 0x20 ,
[L4_VCPU_F_FPU_ENABLED](#) = 0x80 }
State flags of a vCPU.
- enum [L4_vcpu_sticky_flags](#) { [L4_VCPU_SF_IRQ_PENDING](#) = 0x01 }
Sticky flags of a vCPU.
- enum [L4_vcpu_state_offset](#) { [L4_VCPU_OFFSET_EXT_STATE](#) = 0x180 , [L4_VCPU_OFFSET_EXT_INFOS](#) = 0x100 }
Offsets for vCPU state layouts.
- enum [L4_vcpu_state_offset](#) { [L4_VCPU_OFFSET_EXT_STATE](#) = 0x280 , [L4_VCPU_OFFSET_EXT_INFOS](#) = 0x200 }
Offsets for vCPU state layouts.
- enum [L4_vcpu_state_offset](#) { [L4_VCPU_OFFSET_EXT_STATE](#) = 0x400 , [L4_VCPU_OFFSET_EXT_INFOS](#) = 0x200 }
Offsets for vCPU state layouts.
- enum [L4_vcpu_state_offset](#) { [L4_VCPU_OFFSET_EXT_STATE](#) = 0x400 , [L4_VCPU_OFFSET_EXT_INFOS](#) = 0x200 }
Offsets for vCPU state layouts.

14.1.10.13.5.1 Detailed Description

vCPU API.

The vCPU API in [L4Re](#) implements virtual processors (vCPUs) on top of [L4::Thread](#). This API can be used for user level threading, operating system rehosting (see L4Linux) and virtualization.

You switch a thread into vCPU operation with [L4::Thread::vcpu_control](#).

In vCPU mode, incoming IPC can be redirected to a handler function. If an IPC is sent to the vCPU, the thread's normal execution is interrupted and the handler called. Which kind of IPC is redirected is specified by the [L4_vcpu_state_flags](#) set in the [l4_vcpu_state_t::state](#) field of [vcpu_state](#). All events enabled in the [vcpu_state](#) field are redirected to the handler. The handler is set via [l4_vcpu_state_t::entry_ip](#) and [l4_vcpu_state_t::entry_sp](#). IPC redirection works independent of "kernel" and "user" mode, but see [l4_vcpu_state_t::entry_sp](#). When the entry handler is called, the UTCB contains the result of the IPC and content normally found in CPU register is in [l4_vcpu_state_t::i](#).

Furthermore, the thread can execute in the context of different tasks, called the "kernel" and the "user" mode. The kernel task is the one to which the thread was originally bound via [L4::Thread::control\(\)](#). Execution starts in the kernel task and it is always switched to when the asynchronous IPC handler is invoked. When returning from the handler via [l4_thread_vcpu_resume_start\(\)](#) and [l4_thread_vcpu_resume_commit\(\)](#), a different user task can be specified by setting [l4_vcpu_state_t::user_task](#) and enabling the [L4_VCPU_F_USER_MODE](#) flag in [l4_vcpu_state_t::state](#). Note that the kernel may cache the user task internally, see [l4_thread_vcpu_resume_commit\(\)](#).

If the [L4_VCPU_F_USER_MODE](#) flag is enabled, the following flags will be automatically enabled in [l4_vcpu_state_t::state](#) on [L4::Thread::vcpu_resume_commit\(\)](#):

- [L4_VCPU_F_IRQ](#)
- [L4_VCPU_F_PAGE_FAULTS](#)
- [L4_VCPU_F_EXCEPTIONS](#)

When the kernel mode is entered, the following flags will be automatically disabled in [l4_vcpu_state_t::state](#):

- [L4_VCPU_F_IRQ](#)
- [L4_VCPU_F_PAGE_FAULTS](#)
- [L4_VCPU_F_USER_MODE](#)

Extended vCPU operation is used for hardware CPU virtualization. It can be enabled with [L4::Thread::vcpu_control_ext\(\)](#).

[vCPU Support Library](#) defines a convenience API for working with vCPUs.

See also

[vCPU Support Library](#)

14.1.10.13.5.2 Enumeration Type Documentation

L4_vcpu_state_flags

```
enum L4_vcpu_state_flags
```

State flags of a vCPU.

Enumerator

L4_VCPU_F_IRQ	<p>Receiving of IRQs and IPC enabled. While this flag is not set, the corresponding vCPU thread will not receive any IPC and threads attempting to send an IPC to this thread will block (according to the selected send timeout).</p> <p>Note</p> <p>On L4::Thread::vcpu_resume_commit() this flag is automatically enabled in l4_vcpu_state_t::state if L4_VCPU_F_USER_MODE is enabled.</p> <p>When the kernel mode is entered, this flags is automatically disabled in l4_vcpu_state_t::state.</p>
L4_VCPU_F_PAGE_FAULTS	<p>Page faults enabled. If this flag is set, a page fault switches to kernel mode (potentially causing a VM exit) and calls the entry handler. If this flag is not set, a page fault generates a page fault IPC to the pager of the vCPU thread.</p> <p>Note</p> <p>IPC redirection for page faults controlled by this flag works independent of "kernel" and "user" mode.</p> <p>On L4::Thread::vcpu_resume_commit() this flag is automatically enabled in l4_vcpu_state_t::state if L4_VCPU_F_USER_MODE is enabled.</p> <p>When the kernel mode is entered, this flags is automatically disabled in l4_vcpu_state_t::state.</p>
L4_VCPU_F_EXCEPTIONS	<p>Exceptions enabled. If this flag is set, then, on the event of an exception, the vCPU switches to kernel mode (potentially causing a VM exit) and calls the entry handler. If this flag is not set, an exception generates an exception IPC to the exception handler of the vCPU thread.</p> <p>Note</p> <p>IPC redirection for exceptions controlled by this flag works independent of "kernel" and "user" mode.</p> <p>On L4::Thread::vcpu_resume_commit() this flag is automatically enabled in l4_vcpu_state_t::state if L4_VCPU_F_USER_MODE is enabled.</p>
L4_VCPU_F_USER_MODE	<p>User task will be used. If set, the vCPU switches to user mode on next L4::Thread::vcpu_resume_commit(). If clear, the vCPU stays in "kernel" mode.</p> <p>Note</p> <p>When the kernel mode is entered, this flags is automatically disabled in l4_vcpu_state_t::state.</p>
L4_VCPU_F_FPU_ENABLED	<p>FPU enabled. This flag is only relevant if L4_VCPU_F_USER_MODE is set. Setting this flag allows code in vCPU mode to use the FPU. IF this flag is not set, any FPU operation will trigger a corresponding exception (FPU fault).</p>

Definition at line 101 of file [vcpu.h](#).

L4_vcpu_state_offset [1/4]

enum [L4_vcpu_state_offset](#)

Offsets for vCPU state layouts.

Enumerator

L4_VCPU_OFFSET_EXT_STATE	Offset where extended state begins.
L4_VCPU_OFFSET_EXT_INFOS	Offset where extended infos begin.

Definition at line 34 of file [__vcpu-arch.h](#).

L4_vcpu_state_offset [2/4]

```
enum L4_vcpu_state_offset
```

Offsets for vCPU state layouts.

Enumerator

L4_VCPU_OFFSET_EXT_STATE	Offset where extended state begins.
L4_VCPU_OFFSET_EXT_INFOS	Offset where extended infos begin.

Definition at line 35 of file [__vcpu-arch.h](#).

L4_vcpu_state_offset [3/4]

```
enum L4_vcpu_state_offset
```

Offsets for vCPU state layouts.

Enumerator

L4_VCPU_OFFSET_EXT_STATE	Offset where extended state begins.
L4_VCPU_OFFSET_EXT_INFOS	Offset where extended infos begin.

Definition at line 36 of file [__vcpu-arch.h](#).

L4_vcpu_state_offset [4/4]

```
enum L4_vcpu_state_offset
```

Offsets for vCPU state layouts.

Enumerator

L4_VCPU_OFFSET_EXT_STATE	Offset where extended state begins.
--------------------------	-------------------------------------

L4_VCPU_OFFSET_EXT_INFOS	Offset where extended infos begin.
--------------------------	------------------------------------

Definition at line 34 of file [__vcpu-arch.h](#).

L4_vcpu_sticky_flags

enum [L4_vcpu_sticky_flags](#)

Sticky flags of a vCPU.

Enumerator

L4_VCPU_SF_IRQ_PENDING	An event is pending: Either an IRQ or another thread attempts to send an IPC to this vCPU thread.
------------------------	---

Definition at line 167 of file [vcpu.h](#).

14.1.10.14 Thread groups

C thread group interface, see [L4::Thread_group](#) for the C++ interface.

Collaboration diagram for Thread groups:



Functions

- [l4_msgtag_t l4_thread_group_add](#) ([l4_cap_idx_t](#) tg, [l4_cap_idx_t](#) thread) [L4_NOTHROW](#)
Add thread to a thread group.
- [l4_msgtag_t l4_thread_group_remove](#) ([l4_cap_idx_t](#) tg, [l4_cap_idx_t](#) thread) [L4_NOTHROW](#)
Remove thread from a thread group.

14.1.10.14.1 Detailed Description

C thread group interface, see [L4::Thread_group](#) for the C++ interface.

An [L4](#) thread group is a collection of threads used as indirection for IPC gate and IRQ objects such that these objects can have multiple receivers, from which the kernel selects one according to a policy.

The primary use case for thread groups are multi-threaded servers and CPU core local IRQ / IPC delivery.

A thread can be bound to at most one thread group. Before binding a thread to a thread group, the thread must be bound to a task. All threads bound to the same thread group must belong to the same task.

14.1.10.14.2 Function Documentation

14.1.10.14.2.1 l4_thread_group_add()

```
l4_msgtag_t l4_thread_group_add (  
    l4_cap_idx_t tg,  
    l4_cap_idx_t thread) [inline]
```

Add thread to a thread group.

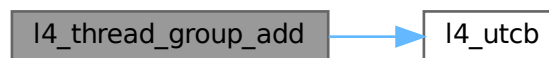
Parameters

<i>tg</i>	Thread group object.
<i>thread</i>	Thread to add to the thread group.

Definition at line 114 of file [thread_group.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.14.2.2 l4_thread_group_remove()

```
l4_msgtag_t l4_thread_group_remove (  
    l4_cap_idx_t tg,  
    l4_cap_idx_t thread) [inline]
```

Remove thread from a thread group.

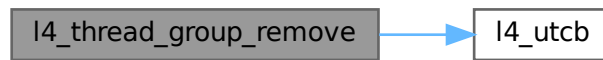
Parameters

<i>tg</i>	Thread group object.
<i>thread</i>	Thread to remove from the thread group.

Definition at line 121 of file [thread_group.h](#).

References [L4_NOTHROW](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



14.1.10.15 Virtual Console

C Virtual console interface for simple character based input and output, see [L4::Vcon](#) for the C++ interface.

Collaboration diagram for Virtual Console:



Data Structures

- struct [l4_vcon_attr_t](#)
Vcon attribute structure.

Typedefs

- typedef struct l4_vcon_attr_t [l4_vcon_attr_t](#)
Vcon attribute structure.

Enumerations

- enum [L4_vcon_size_consts](#) { [L4_VCON_WRITE_SIZE](#) = (L4_UTCB_GENERIC_DATA_SIZE - 2) * sizeof(l4_umword_t) , [L4_VCON_READ_SIZE](#) = (L4_UTCB_GENERIC_DATA_SIZE - 1) * sizeof(l4_umword_t) }
Size constants.
- enum [L4_vcon_i_flags](#) { [L4_VCON_INLCR](#) = 000100 , [L4_VCON_IGNCR](#) = 000200 , [L4_VCON_ICRNL](#) = 000400 }
Input flags.
- enum [L4_vcon_o_flags](#) { [L4_VCON_ONLCR](#) = 000004 , [L4_VCON_OCRNL](#) = 000010 , [L4_VCON_ONLRET](#) = 000040 }
Output flags.
- enum [L4_vcon_l_flags](#) { [L4_VCON_ICANON](#) = 000002 , [L4_VCON_ECHO](#) = 000010 }
Local flags.

Functions

- [l4_msgtag_t l4_vcon_send](#) ([l4_cap_idx_t](#) vcon, char const *buf, unsigned size) [L4_NOTHROW](#)
Send data to virtual console.
- [l4_msgtag_t l4_vcon_send_u](#) ([l4_cap_idx_t](#) vcon, char const *buf, unsigned size, [l4_utcb_t](#) *utcb) [L4_NOTHROW](#)
*Send data to *this* virtual console.*
- long [l4_vcon_write](#) ([l4_cap_idx_t](#) vcon, char const *buf, unsigned size) [L4_NOTHROW](#)
Write data to virtual console.
- long [l4_vcon_write_u](#) ([l4_cap_idx_t](#) vcon, char const *buf, unsigned size, [l4_utcb_t](#) *utcb) [L4_NOTHROW](#)
*Write data to *this* virtual console.*
- int [l4_vcon_read](#) ([l4_cap_idx_t](#) vcon, char *buf, unsigned size) [L4_NOTHROW](#)
Read data from virtual console.
- int [l4_vcon_read_u](#) ([l4_cap_idx_t](#) vcon, char *buf, unsigned size, [l4_utcb_t](#) *utcb) [L4_NOTHROW](#)
*Read data from *this* virtual console.*
- int [l4_vcon_read_with_flags](#) ([l4_cap_idx_t](#) vcon, char *buf, unsigned size) [L4_NOTHROW](#)
Read data from virtual console, extended version including flags.
- [l4_msgtag_t l4_vcon_set_attr](#) ([l4_cap_idx_t](#) vcon, [l4_vcon_attr_t](#) const *attr) [L4_NOTHROW](#)
Set attributes of a Vcon.
- [l4_msgtag_t l4_vcon_set_attr_u](#) ([l4_cap_idx_t](#) vcon, [l4_vcon_attr_t](#) const *attr, [l4_utcb_t](#) *utcb) [L4_NOTHROW](#)
*Set the attributes of *this* virtual console.*
- [l4_msgtag_t l4_vcon_get_attr](#) ([l4_cap_idx_t](#) vcon, [l4_vcon_attr_t](#) *attr) [L4_NOTHROW](#)
Get attributes of a Vcon.
- [l4_msgtag_t l4_vcon_get_attr_u](#) ([l4_cap_idx_t](#) vcon, [l4_vcon_attr_t](#) *attr, [l4_utcb_t](#) *utcb) [L4_NOTHROW](#)
*Get attributes of *this* virtual console.*
- void [l4_vcon_set_attr_raw](#) ([l4_vcon_attr_t](#) *attr) [L4_NOTHROW](#)
Set terminal attributes to disable all special processing.

14.1.10.15.1 Detailed Description

C Virtual console interface for simple character based input and output, see [L4::Vcon](#) for the C++ interface.

The interrupt for read events is provided by the virtual key interrupt which, in contrast to hardware IRQs, implements a limited functionality:

- Only IRQ line 0 is supported, no MSI vectors.
- The IRQ is edge-triggered and the IRQ mode cannot be changed.
- As the IRQ is edge-triggered, it does not have to be explicitly unmasked.

A server implementing the virtual console protocol has a queue for input events. When the first input event is added to the empty queue, the virtual key interrupt is triggered. Further events are added to the queue without generating further interrupts. The queue is emptied when a client reads all queued input events.

Include File

```
#include <l4/sys/vcon.h>
```

See [L4::Vcon](#) for the C++ interface.

14.1.10.15.2 Typedef Documentation

14.1.10.15.2.1 l4_vcon_attr_t

```
typedef struct l4_vcon_attr_t l4_vcon_attr_t
```

Vcon attribute structure.

The flags members can be a combination of their respective enums.

See also

[L4_vcon_i_flags](#)

[L4_vcon_o_flags](#)

[L4_vcon_l_flags](#)

14.1.10.15.3 Enumeration Type Documentation

14.1.10.15.3.1 L4_vcon_i_flags

```
enum L4_vcon_i_flags
```

Input flags.

Enumerator

L4_VCON_INLCR	Translate NL to CR.
L4_VCON_IGNCR	Ignore CR.
L4_VCON_ICRNL	Translate CR to NL if L4_VCON_IGNCR is not set.

Definition at line 208 of file [vcon.h](#).

14.1.10.15.3.2 L4_vcon_l_flags

```
enum L4_vcon_l_flags
```

Local flags.

Enumerator

L4_VCON_ICANON	Canonical mode.
L4_VCON_ECHO	Echo input.

Definition at line 230 of file [vcon.h](#).

14.1.10.15.3.3 L4_vcon_o_flags

```
enum L4_vcon_o_flags
```

Output flags.

Enumerator

L4_VCON_ONLCR	Translate NL to CR-NL.
L4_VCON_OCRNL	Translate CR to NL.
L4_VCON_ONLRET	Do not output CR.

Definition at line 219 of file [vcon.h](#).

14.1.10.15.3.4 L4_vcon_size_consts

```
enum L4_vcon_size_consts
```

Size constants.

Enumerator

L4_VCON_WRITE_SIZE	Maximum size that can be written with one l4_vcon_write call.
L4_VCON_READ_SIZE	Maximum size that can be read with one l4_vcon_read* call.

Definition at line 95 of file [vcon.h](#).

14.1.10.15.4 Function Documentation

14.1.10.15.4.1 l4_vcon_get_attr()

```
l4_msgtag_t l4_vcon_get_attr (
    l4_cap_idx_t vcon,
    l4_vcon_attr_t * attr) [inline]
```

Get attributes of a Vcon.

Parameters

	<i>vcon</i>	Vcon object.
out	<i>attr</i>	Attribute structure.

Returns

Syscall return tag

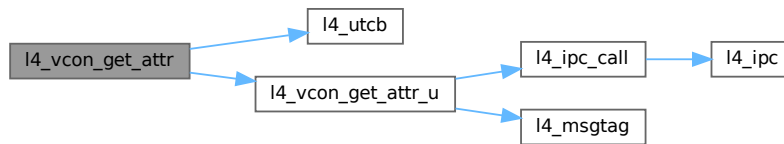
Examples

[examples/sys/isr/main.c](#).

Definition at line 435 of file [vcon.h](#).

References [L4_NOTHROW](#), [l4_utcb\(\)](#), and [l4_vcon_get_attr_u\(\)](#).

Here is the call graph for this function:



14.1.10.15.4.2 l4_vcon_get_attr_u()

```

l4_msgtag_t l4_vcon_get_attr_u (
    l4_cap_idx_t vcon,
    l4_vcon_attr_t * attr,
    l4_utcb_t * utcb) [inline]
  
```

Get attributes of this virtual console.

Parameters

	<i>vcon</i>	Capability index of the vcon object.
out	<i>attr</i>	Attribute structure. Contains the attributes after a successful call of this function.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

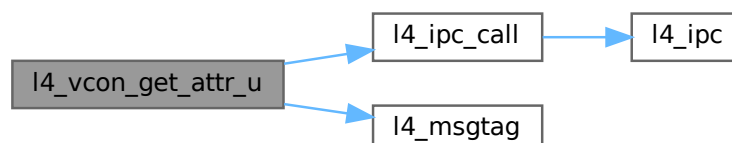
Syscall return tag.

Definition at line 417 of file [vcon.h](#).

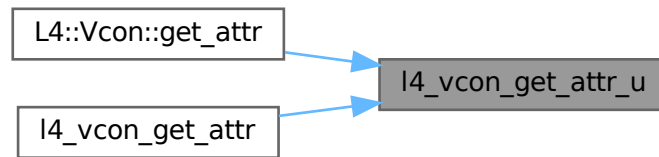
References [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), [l4_msgtag\(\)](#), [L4_NOTHROW](#), [L4_PROTO_LOG](#), [L4_VCON_GET_ATTR_OP](#), and [l4_msg_regs_t::mr](#).

Referenced by [L4::Vcon::get_attr\(\)](#), and [l4_vcon_get_attr\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.10.15.4.3 l4_vcon_read()

```
int l4_vcon_read (
    l4_cap_idx_t vcon,
    char * buf,
    unsigned size) [inline]
```

Read data from virtual console.

Parameters

	<i>vcon</i>	Vcon object.
out	<i>buf</i>	Pointer to data buffer.
	<i>size</i>	Size of buffer in bytes.

Return values

<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i>>size</i>	More bytes to read, <i>size</i> bytes are in the buffer <i>buf</i> .
<i><=size</i>	Number of bytes read.

Precondition

The capability *vcon* must have the permission [L4_CAP_FPAGE_W](#).

Note

Size must not exceed [L4_VCON_READ_SIZE](#).

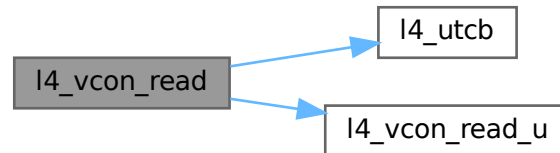
Examples

[examples/sys/isr/main.c](#).

Definition at line 391 of file [vcon.h](#).

References [L4_NOTHROW](#), [l4_utcb\(\)](#), and [l4_vcon_read_u\(\)](#).

Here is the call graph for this function:



14.1.10.15.4.4 l4_vcon_read_u()

```

int l4_vcon_read_u (
    l4_cap_idx_t vcon,
    char * buf,
    unsigned size,
    l4_utcb_t * utcb) [inline]
  
```

Read data from this virtual console.

Parameters

	<i>vcon</i>	Capability index of the vcon object.
out	<i>buf</i>	Pointer to data buffer.
	<i>size</i>	Size of the data buffer in bytes.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Return values

<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i>>size</i>	More bytes to read, <i>size</i> bytes are in the buffer <i>buf</i> .
<i><=size</i>	Number of bytes read.

Precondition

The invoked Vcon capability must have the permission [L4_CAP_FPAGE_W](#).

Note

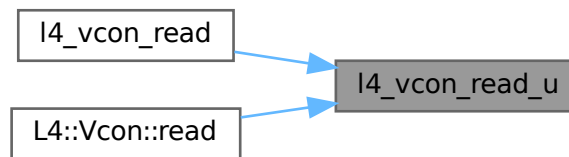
Size must not exceed [L4_VCON_READ_SIZE](#).

Definition at line 381 of file [vcon.h](#).

References [L4_NOTHROW](#), and [L4_VCON_READ_SIZE_MASK](#).

Referenced by [l4_vcon_read\(\)](#), and [L4::Vcon::read\(\)](#).

Here is the caller graph for this function:

**14.1.10.15.4.5 l4_vcon_read_with_flags()**

```
int l4_vcon_read_with_flags (
    l4_cap_idx_t vcon,
    char * buf,
    unsigned size) [inline]
```

Read data from virtual console, extended version including flags.

Parameters

	<i>vcon</i>	Vcon object.
out	<i>buf</i>	Pointer to data buffer.
	<i>size</i>	Size of buffer in bytes.

If this function returns a positive value the caller can check the [L4_VCON_READ_STAT_BREAK](#) flag bit for a break condition. The bytes read can be obtained by masking the return value with [L4_VCON_READ_SIZE_MASK](#).

If a break condition is signaled, it is always the first event in the transmitted content, i.e. all characters supplied by this read call follow the break condition.

`buf` might be a `NULL`, in this case the input data will be dropped.

Note

Size must not exceed [L4_VCON_READ_SIZE](#).

Return values

<code>-L4_EPERM</code>	Insufficient permissions; see precondition.
<code>>size</code>	More bytes to read, <code>size</code> bytes are in the buffer <code>buf</code> .
<code><=size</code>	Number of bytes read.

Precondition

The capability `vcon` must have the permission `L4_CAP_FPAGE_W`.

Definition at line 375 of file `vcon.h`.

References `L4_NOTHROW`, and `l4_utcb()`.

Here is the call graph for this function:



14.1.10.15.4.6 l4_vcon_send()

```

l4_msgtag_t l4_vcon_send (
    l4_cap_idx_t vcon,
    char const * buf,
    unsigned size) [inline]
  
```

Send data to virtual console.

Parameters

<code>vcon</code>	Vcon object.
<code>buf</code>	Pointer to data buffer.
<code>size</code>	Size of buffer in bytes.

Returns

Syscall return tag

Note

Size must not exceed [L4_VCON_WRITE_SIZE](#), a proper value of the `size` parameter is NOT checked. Also, this function is a send only operation, this means there is no return value except for a failed send operation. Use [l4_ipc_error\(\)](#) to check for send errors, and **do not** use [l4_error\(\)](#).

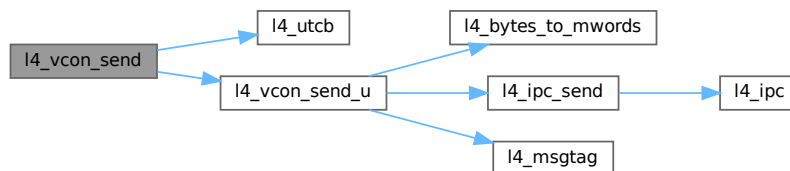
Examples

[examples/sys/utcb-ipc/main.c](#).

Definition at line 315 of file [vcon.h](#).

References [L4_NOTHROW](#), [l4_utcb\(\)](#), and [l4_vcon_send_u\(\)](#).

Here is the call graph for this function:

**14.1.10.15.4.7 l4_vcon_send_u()**

```

l4_msgtag_t l4_vcon_send_u (
    l4_cap_idx_t vcon,
    char const * buf,
    unsigned size,
    l4_utcb_t * utcb) [inline]
  
```

Send data to this virtual console.

Parameters

<i>vcon</i>	Capability index of the Vcon object.
<i>buf</i>	Pointer to the data buffer.
<i>size</i>	Size of the data buffer in bytes.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

Syscall return tag

Note

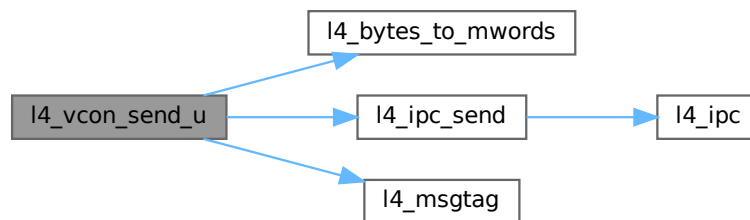
Size must not exceed [L4_VCON_WRITE_SIZE](#), a proper value of the `size` parameter is NOT checked. Also, this function is a send only operation, this means there is no return value except for a failed send operation. Use [l4_ipc_error\(\)](#) to check for send errors, do not use [l4_error\(\)](#), as [l4_error\(\)](#) will always return an error.

Definition at line 302 of file [vcon.h](#).

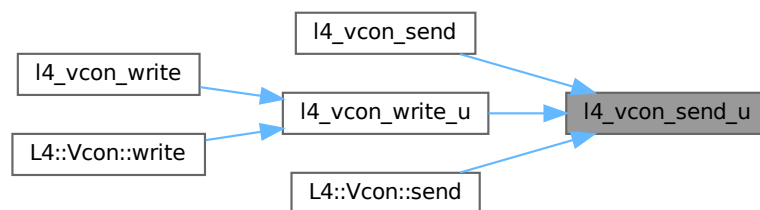
References [l4_bytes_to_mwords\(\)](#), [L4_IPC_NEVER](#), [l4_ipc_send\(\)](#), [l4_msgtag\(\)](#), [L4_MSGTAG_SCHEDULE](#), [L4_NOTHROW](#), [L4_PROTO_LOG](#), [L4_VCON_WRITE_OP](#), and [l4_msg_regs_t::mr](#).

Referenced by [l4_vcon_send\(\)](#), [l4_vcon_write_u\(\)](#), and [L4::Vcon::send\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.10.15.4.8 l4_vcon_set_attr()

```

l4_msgtag_t l4_vcon_set_attr (
    l4_cap_idx_t vcon,
    l4_vcon_attr_t const * attr) [inline]
  
```

Set attributes of a Vcon.

Parameters

<i>vcon</i>	Vcon object.
<i>attr</i>	Attribute structure.

Returns

Syscall return tag

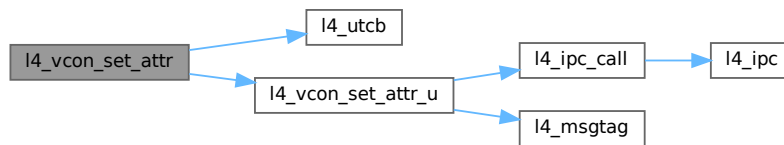
Examples

[examples/sys/isr/main.c](#).

Definition at line 411 of file [vcon.h](#).

References [L4_NOTHROW](#), [l4_utcb\(\)](#), and [l4_vcon_set_attr_u\(\)](#).

Here is the call graph for this function:



14.1.10.15.4.9 l4_vcon_set_attr_raw()

```
void l4_vcon_set_attr_raw (
    l4_vcon_attr_t * attr) [inline]
```

Set terminal attributes to disable all special processing.

Removes all flags that would mangle the read or written characters. Also disables echoing and any special processing of characters.

Parameters

<i>in, out</i>	<i>attr</i>	Attribute structure to update.
----------------	-------------	--------------------------------

Definition at line 441 of file [vcon.h](#).

References [L4_NOTHROW](#).

Referenced by [l4_vcon_attr_t::set_raw\(\)](#).

Here is the caller graph for this function:



14.1.10.15.4.10 l4_vcon_set_attr_u()

```
l4_msgtag_t l4_vcon_set_attr_u (
    l4_cap_idx_t vcon,
    l4_vcon_attr_t const * attr,
    l4_utcb_t * utcb) [inline]
```

Set the attributes of this virtual console.

Parameters

<i>vcon</i>	Capability index of the vcon object.
<i>attr</i>	Attribute structure with the attributes for the virtual console.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

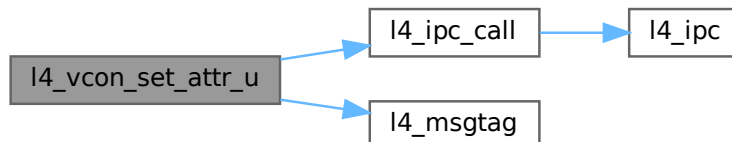
Syscall return tag.

Definition at line 397 of file [vcon.h](#).

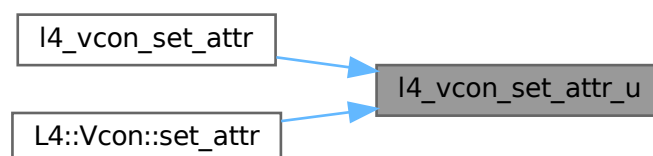
References [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), [l4_msgtag\(\)](#), [L4_NOTHROW](#), [L4_PROTO_LOG](#), [L4_VCON_SET_ATTR_OP](#), and [l4_msg_regs_t::mr](#).

Referenced by [l4_vcon_set_attr\(\)](#), and [L4::Vcon::set_attr\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.10.15.4.11 l4_vcon_write()

```
long l4_vcon_write (
    l4_cap_idx_t vcon,
    char const * buf,
    unsigned size) [inline]
```

Write data to virtual console.

Parameters

<i>vcon</i>	Vcon object.
<i>buf</i>	Pointer to data buffer.
<i>size</i>	Size of buffer in bytes.

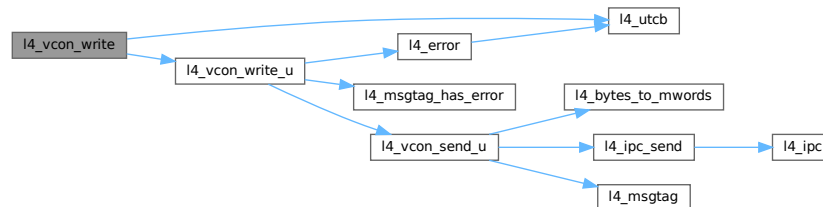
Return values

< 0	Error.
≥ 0	Number of bytes written to the virtual console

Definition at line 336 of file [vcon.h](#).

References [L4_NOTHROW](#), [l4_utcb\(\)](#), and [l4_vcon_write_u\(\)](#).

Here is the call graph for this function:

**14.1.10.15.4.12 l4_vcon_write_u()**

```
long l4_vcon_write_u (
    l4_cap_idx_t vcon,
    char const * buf,
    unsigned size,
    l4_utcb_t * utcb) [inline]
```

Write data to this virtual console.

Parameters

<i>vcon</i>	Capability index of the vcon object.
<i>buf</i>	Pointer to the data buffer.
<i>size</i>	Size of the data buffer in bytes.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Return values

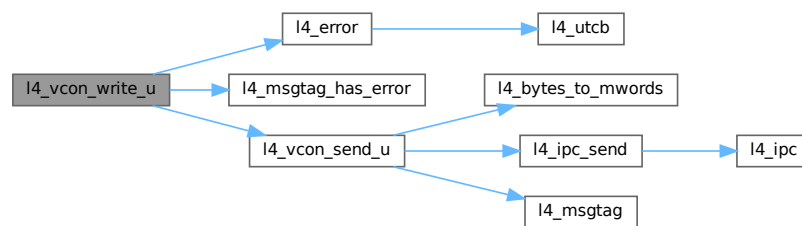
< 0	Error.
≥ 0	Number of bytes written to the virtual console.

Definition at line 321 of file [vcon.h](#).

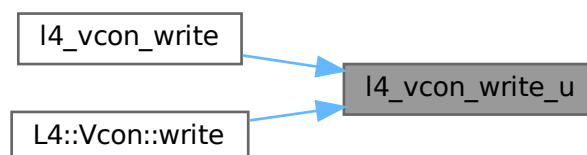
References [l4_error\(\)](#), [l4_msgtag_has_error\(\)](#), [L4_NOTHROW](#), [l4_vcon_send_u\(\)](#), and [L4_VCON_WRITE_SIZE](#).

Referenced by [l4_vcon_write\(\)](#), and [L4::Vcon::write\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.1.11 Kernel Interface Page

Kernel Interface Page.

Collaboration diagram for Kernel Interface Page:



Topics

- [Memory descriptors \(C version\)](#) ??
C Interface for KIP memory descriptors.

Data Structures

- class [L4::Kip::Mem_desc](#)
Memory descriptors stored in the kernel interface page.
- struct [l4_kernel_info_t](#)
L4 Kernel Interface Page.

Macros

- `#define L4_KERNEL_INFO_MAGIC (0x4BE6344CL) /* "L4μK" */`
Kernel Info Page identifier ("L4μK").

Typedefs

- typedef struct [l4_kernel_info_t](#) [l4_kernel_info_t](#)
L4 Kernel Interface Page.

Enumerations

- enum { [L4_KIP_OFFS_READ_US](#) = 0x900 , [L4_KIP_OFFS_READ_NS](#) = 0x980 }

Functions

- [l4_kernel_info_t](#) const * [l4_kip](#) (void) [L4_NOTHROW](#)
Get Kernel Info Page.
- [l4_umword_t](#) [l4_kip_version](#) ([l4_kernel_info_t](#) const *kip) [L4_NOTHROW](#)
Get the kernel version.
- const char * [l4_kip_version_string](#) ([l4_kernel_info_t](#) const *kip) [L4_NOTHROW](#)
Get the kernel version string.
- int [l4_kernel_info_version_offset](#) ([l4_kernel_info_t](#) const *kip) [L4_NOTHROW](#)
Return offset in bytes of version_strings relative to the KIP base.
- [l4_cpu_time_t](#) [l4_kip_clock](#) ([l4_kernel_info_t](#) const *kip) [L4_NOTHROW](#)
Return clock value from the KIP.
- [l4_umword_t](#) [l4_kip_clock_lw](#) ([l4_kernel_info_t](#) const *kip) [L4_NOTHROW](#)
Return least significant machine word of clock value from the KIP.
- [l4_uint64_t](#) [l4_kip_clock_ns](#) ([l4_kernel_info_t](#) const *kip) [L4_NOTHROW](#)
Return current clock using the KIP in nanoseconds.

14.1.11.1 Detailed Description

Kernel Interface Page.

C++ interface for the Kernel Interface Page:

Include File

```
#include <l4/sys/kip>
```

C interface for the Kernel Interface Page:

Include File

```
#include <l4/sys/kip.h>
```

14.1.11.2 Typedef Documentation

14.1.11.2.1 l4_kernel_info_t

```
typedef struct l4_kernel_info_t l4_kernel_info_t
```

[L4](#) Kernel Interface Page.

32-bit architecture may assume that the upper 32 bits of addresses is 0

14.1.11.3 Enumeration Type Documentation

14.1.11.3.1 anonymous enum

```
anonymous enum
```

Enumerator

L4_KIP_OFFS_READ_US	Offset of KIP code (provided by the kernel) for reading the KIP clock in microseconds. If the kernel is configured for a fine-grained KIP clock (CONFIG_SYNC_↔ TSC enabled for IA32, ARM_SYNC_CLOCK for ARM), this code provides the KIP clock with microseconds granularity and accuracy by reading the hardware clock used by the kernel and transforming this value into microseconds. Otherwise this code just reads the KIP clock value.
L4_KIP_OFFS_READ_NS	Offset of KIP code (provided by the kernel) for reading the time stamp counter and transforming this value into nanoseconds. If the kernel is configured for fine-grained KIP clock (CONFIG_SYNC enabled for IA32, ARM_SYNC_CLOCK for ARM), this code provides the KIP clock with nanoseconds granularity and accuracy by reading the hardware clock used by the kernel and transforming this value into nanoseconds. Otherwise this code just reads the KIP clock value and multiplies it by 1000.

Definition at line 95 of file [kip.h](#).

14.1.11.4 Function Documentation

14.1.11.4.1 l4_kernel_info_version_offset()

```
int l4_kernel_info_version_offset (  
    l4_kernel_info_t const * kip) [inline]
```

Return offset in bytes of version_strings relative to the KIP base.

Parameters

<i>kip</i>	Pointer to the kernel info page (KIP).
------------	--

Returns

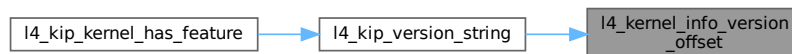
offset of version_strings relative to the KIP base address, in bytes.

Definition at line 238 of file [kip.h](#).

References [L4_NOTHROW](#).

Referenced by [l4_kip_version_string\(\)](#).

Here is the caller graph for this function:



14.1.11.4.2 l4_kip()

```
l4_kernel_info_t const * l4_kip (
    void ) [inline]
```

Get Kernel Info Page.

Returns

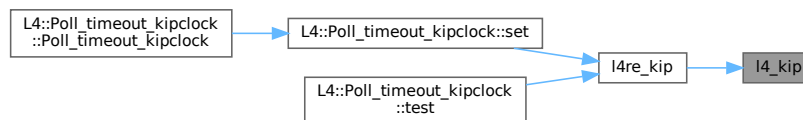
Pointer to Kernel Info Page (KIP) structure.

Definition at line 226 of file [kip.h](#).

References [L4_NOTHROW](#).

Referenced by [l4re_kip\(\)](#).

Here is the caller graph for this function:



14.1.11.4.3 l4_kip_clock()

```
l4_cpu_time_t l4_kip_clock (
    l4_kernel_info_t const * kip) [inline]
```

Return clock value from the KIP.

Parameters

<i>kip</i>	Pointer to the kernel info page (KIP).
------------	--

Returns

Value of the clock field in the KIP.

The KIP clock always contains the current (relative) time in micro seconds independently of the CPU frequency. The clock is only guaranteed to be accurate within the scheduling granularity announced in the KIP.

This function basically calls the KIP code for reading the KIP clock with microseconds resolution. The accuracy depends on the platform and the kernel configuration.

See also

[L4_KIP_OFFS_READ_US](#).

Examples

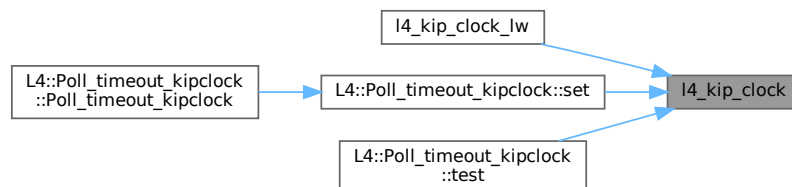
[examples/libs/shmc/prodcons.c](#).

Definition at line 242 of file [kip.h](#).

References [L4_KIP_OFFS_READ_US](#), and [L4_NOTHROW](#).

Referenced by [l4_kip_clock_lw\(\)](#), [L4::Poll_timeout_kipclock::set\(\)](#), and [L4::Poll_timeout_kipclock::test\(\)](#).

Here is the caller graph for this function:



14.1.11.4.4 l4_kip_clock_lw()

```
l4_umword_t l4_kip_clock_lw (
    l4_kernel_info_t const * kip) [inline]
```

Return least significant machine word of clock value from the KIP.

Parameters

<i>kip</i>	Pointer to the kernel info page (KIP).
------------	--

Returns

Lower machine word of clock value from the KIP.

Deprecated Use [l4_kip_clock\(\)](#) instead.

This function will always provide the least significant machine word of the clock value from the KIP, regardless of the kernel configuration.

Definition at line 261 of file [kip.h](#).

References [l4_kip_clock\(\)](#), and [L4_NOTHROW](#).

Here is the call graph for this function:



14.1.11.4.5 l4_kip_clock_ns()

```
l4_cpu_time_t l4_kip_clock_ns (
    l4_kernel_info_t const * kip) [inline]
```

Return current clock using the KIP in nanoseconds.

Parameters

<i>kip</i>	Pointer to the kernel info page (KIP).
------------	--

Returns

Value of the current clock in nanoseconds.

This function basically calls the KIP code for reading the KIP clock with nanoseconds resolution. The accuracy depends on the platform and the kernel configuration.

See also

[L4_KIP_OFFS_READ_NS](#).

Definition at line 252 of file [kip.h](#).

References [L4_KIP_OFFS_READ_NS](#), and [L4_NOTHROW](#).

14.1.11.4.6 l4_kip_version()

```
l4_umword_t l4_kip_version (  
    l4_kernel_info_t const * kip)  [inline]
```

Get the kernel version.

Parameters

<i>kip</i>	Kernel Info Page.
------------	-------------------

Returns

Kernel version string. 0 if KIP could not be mapped.

Definition at line 230 of file [kip.h](#).

References [L4_NOTHROW](#).

14.1.11.4.7 l4_kip_version_string()

```
const char * l4_kip_version_string (  
    l4_kernel_info_t const * kip)    [inline]
```

Get the kernel version string.

Parameters

<i>kip</i>	Kernel Info Page.
------------	-------------------

Returns

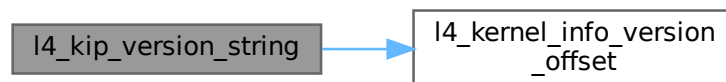
Kernel version string.

Definition at line 234 of file [kip.h](#).

References [l4_kernel_info_version_offset\(\)](#), and [L4_NOTHROW](#).

Referenced by [l4_kip_kernel_has_feature\(\)](#).

Here is the call graph for this function:



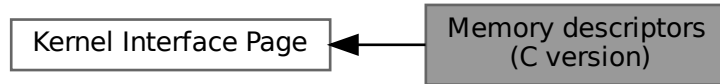
Here is the caller graph for this function:



14.1.11.5 Memory descriptors (C version)

C Interface for KIP memory descriptors.

Collaboration diagram for Memory descriptors (C version):



Data Structures

- struct `l4_kernel_info_mem_desc_t`
Memory descriptor data structure.

Typedefs

- typedef struct `l4_kernel_info_mem_desc_t` `l4_kernel_info_mem_desc_t`
Memory descriptor data structure.

Enumerations

- enum `l4_mem_type_t` {
`l4_mem_type_undefined` = 0x0 , `l4_mem_type_conventional` = 0x1 , `l4_mem_type_reserved` = 0x2 ,
`l4_mem_type_dedicated` = 0x3 ,
`l4_mem_type_shared` = 0x4 , `l4_mem_type_info` = 0xd , `l4_mem_type_bootloader` = 0xe , `l4_mem_type_archspecific`
= 0xf }
Type of a memory descriptor.
- enum `l4_mem_info_sub_type_t` { `l4_mem_info_acpi_rsd` = 0 }
Memory sub types for l4_mem_type_info descriptors.
- enum `l4_mem_archspecific_sub_type_common_t` { `l4_mem_archspecific_acpi_tables` = 3 , `l4_mem_archspecific_acpi_nvs`
= 4 }
Memory sub types for l4_mem_type_archspecific descriptors.

Functions

- `l4_kernel_info_mem_desc_t * l4_kernel_info_get_mem_descs (l4_kernel_info_t *kip) L4_NOTHROW`
Get pointer to memory descriptors from KIP.
- unsigned `l4_kernel_info_get_num_mem_descs (l4_kernel_info_t *kip) L4_NOTHROW`
Get number of memory descriptors in KIP.
- void `l4_kernel_info_set_mem_desc (l4_kernel_info_mem_desc_t *md, l4_addr_t start, l4_addr_t end, unsigned type, unsigned virt, unsigned sub_type) L4_NOTHROW`
Populate a memory descriptor.
- `l4_umword_t l4_kernel_info_get_mem_desc_start (l4_kernel_info_mem_desc_t *md) L4_NOTHROW`

Get start address of the region described by the memory descriptor.

- [l4_umword_t l4_kernel_info_get_mem_desc_end](#) ([l4_kernel_info_mem_desc_t *md](#)) [L4_NOTHROW](#)

Get end address of the region described by the memory descriptor.

- [l4_umword_t l4_kernel_info_get_mem_desc_type](#) ([l4_kernel_info_mem_desc_t *md](#)) [L4_NOTHROW](#)

Get type of the memory region.

- [l4_umword_t l4_kernel_info_get_mem_desc_subtype](#) ([l4_kernel_info_mem_desc_t *md](#)) [L4_NOTHROW](#)

Get sub-type of memory region.

- [l4_umword_t l4_kernel_info_get_mem_desc_is_virtual](#) ([l4_kernel_info_mem_desc_t *md](#)) [L4_NOTHROW](#)

Get virtual flag of the memory descriptor.

14.1.11.5.1 Detailed Description

C Interface for KIP memory descriptors.

Include File

```
#include <l4/sys/memdesc.h>
```

This module contains the C functions to access the memory descriptor in the kernel interface page (KIP).

14.1.11.5.2 Typedef Documentation

14.1.11.5.2.1 l4_kernel_info_mem_desc_t

```
typedef struct l4_kernel_info_mem_desc_t l4_kernel_info_mem_desc_t
```

Memory descriptor data structure.

Note

This data type is opaque, and must be accessed by the accessor functions defined in this module.

14.1.11.5.3 Enumeration Type Documentation

14.1.11.5.3.1 l4_mem_archspecific_sub_type_common_t

```
enum l4_mem_archspecific_sub_type_common_t
```

Memory sub types for l4_mem_type_archspecific descriptors.

Enumerator

l4_mem_archspecific_acpi_tables	Firmware ACPI tables.
l4_mem_archspecific_acpi_nvs	Firmware reserved address space.

Definition at line 59 of file [memdesc.h](#).

14.1.11.5.3.2 l4_mem_info_sub_type_t

```
enum l4_mem_info_sub_type_t
```

Memory sub types for l4_mem_type_info descriptors.

Enumerator

<code>l4_mem_info_acpi_rsdp</code>	Physical address of the ACPI root pointer.
------------------------------------	--

Definition at line 50 of file [memdesc.h](#).

14.1.11.5.3.3 `l4_mem_type_t`

enum `l4_mem_type_t`

Type of a memory descriptor.

Enumerator

<code>l4_mem_type_undefined</code>	Undefined, unused descriptor.
<code>l4_mem_type_conventional</code>	Conventional memory.
<code>l4_mem_type_reserved</code>	Reserved memory for kernel etc.
<code>l4_mem_type_dedicated</code>	Dedicated memory (some device memory).
<code>l4_mem_type_shared</code>	Shared memory (not implemented).
<code>l4_mem_type_info</code>	Info from the boot loader.
<code>l4_mem_type_bootloader</code>	Memory owned by the boot loader.
<code>l4_mem_type_archspecific</code>	Architecture specific memory (e.g., ACPI memory).

Definition at line 33 of file [memdesc.h](#).

14.1.11.5.4 Function Documentation

14.1.11.5.4.1 `l4_kernel_info_get_mem_desc_end()`

```
l4_umword_t l4_kernel_info_get_mem_desc_end (
    l4_kernel_info_mem_desc_t * md) [inline]
```

Get end address of the region described by the memory descriptor.

Returns

End address.

Definition at line 215 of file [memdesc.h](#).

References [L4_NOTHROW](#).

14.1.11.5.4.2 `l4_kernel_info_get_mem_desc_is_virtual()`

```
l4_umword_t l4_kernel_info_get_mem_desc_is_virtual (
    l4_kernel_info_mem_desc_t * md) [inline]
```

Get virtual flag of the memory descriptor.

Returns

1 if region is virtual memory, 0 if region is physical memory

Definition at line 236 of file [memdesc.h](#).

References [L4_NOTHROW](#).

14.1.11.5.4.3 l4_kernel_info_get_mem_desc_start()

```
l4_umword_t l4_kernel_info_get_mem_desc_start (  
    l4_kernel_info_mem_desc_t * md) [inline]
```

Get start address of the region described by the memory descriptor.

Returns

Start address.

Definition at line 208 of file [memdesc.h](#).

References [L4_NOTHROW](#).

14.1.11.5.4.4 l4_kernel_info_get_mem_desc_subtype()

```
l4_umword_t l4_kernel_info_get_mem_desc_subtype (  
    l4_kernel_info_mem_desc_t * md) [inline]
```

Get sub-type of memory region.

Returns

Sub-type.

The sub type is defined for architecture specific memory descriptors (see [l4_mem_type_archspecific](#)) and has architecture specific meaning.

Definition at line 229 of file [memdesc.h](#).

References [L4_NOTHROW](#).

14.1.11.5.4.5 l4_kernel_info_get_mem_desc_type()

```
l4_umword_t l4_kernel_info_get_mem_desc_type (  
    l4_kernel_info_mem_desc_t * md) [inline]
```

Get type of the memory region.

Returns

Type of the region (see [l4_mem_type_t](#)).

Definition at line 222 of file [memdesc.h](#).

References [L4_NOTHROW](#).

14.1.11.5.4.6 l4_kernel_info_get_num_mem_descs()

```
unsigned l4_kernel_info_get_num_mem_descs (
    l4_kernel_info_t * kip) [inline]
```

Get number of memory descriptors in KIP.

Returns

Number of memory descriptors.

Definition at line 186 of file [memdesc.h](#).

References [L4_NOTHROW](#).

14.1.11.5.4.7 l4_kernel_info_set_mem_desc()

```
void l4_kernel_info_set_mem_desc (
    l4_kernel_info_mem_desc_t * md,
    l4_addr_t start,
    l4_addr_t end,
    unsigned type,
    unsigned virt,
    unsigned sub_type) [inline]
```

Populate a memory descriptor.

Parameters

<i>md</i>	Pointer to memory descriptor
<i>start</i>	Start of region
<i>end</i>	End of region
<i>type</i>	Type of region
<i>virt</i>	1 if virtual region, 0 if physical region
<i>sub_type</i>	Sub type.

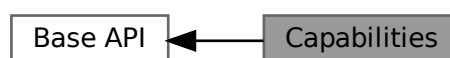
Definition at line 193 of file [memdesc.h](#).

References [L4_NOTHROW](#).

14.1.12 Capabilities

C interface for capabilities.

Collaboration diagram for Capabilities:



Typedefs

- typedef unsigned long [l4_cap_idx_t](#)
Capability selector type.

Enumerations

- enum [l4_cap_consts_t](#) {
 [L4_CAP_SHIFT](#) , [L4_CAP_SIZE](#) = 1UL << [L4_CAP_SHIFT](#) , [L4_CAP_OFFSET](#) , [L4_CAP_MASK](#) ,
 [L4_INVALID_CAP](#) , [L4_INVALID_CAP_BIT](#) = 1UL << ([L4_CAP_SHIFT](#) - 1) }
Constants related to capability selectors.
- enum [l4_default_caps_t](#) {
 [L4_BASE_TASK_CAP](#) , [L4_BASE_FACTORY_CAP](#) , [L4_BASE_THREAD_CAP](#) , [L4_BASE_PAGER_CAP](#) ,
 [L4_BASE_LOG_CAP](#) , [L4_BASE_ICU_CAP](#) , [L4_BASE_SCHEDULER_CAP](#) , [L4_BASE_IOMMU_CAP](#) ,
 [L4_BASE_DEBUGGER_CAP](#) , [L4_BASE_ARM_SMCCC_CAP](#) , [L4_BASE_CAPS_LAST_P1](#) , [L4_BASE_CAPS_LAST](#)
 = [L4_BASE_CAPS_LAST_P1](#) - 1 }
Default capabilities setup for the initial tasks.

Functions

- unsigned [l4_is_invalid_cap](#) ([l4_cap_idx_t](#) c) [L4_NOTHROW](#)
Test if a capability selector is the invalid capability.
- unsigned [l4_is_valid_cap](#) ([l4_cap_idx_t](#) c) [L4_NOTHROW](#)
Test if a capability selector is a valid selector.
- unsigned [l4_capability_equal](#) ([l4_cap_idx_t](#) c1, [l4_cap_idx_t](#) c2) [L4_NOTHROW](#)
Test if the capability indices of two capability selectors are equal.

14.1.12.1 Detailed Description

C interface for capabilities.

Add

```
#include <l4/sys/types.h>
#include <l4/sys/consts.h>
```

to your code to use the functions and definitions explained here.

14.1.12.2 Typedef Documentation

14.1.12.2.1 [l4_cap_idx_t](#)

```
typedef unsigned long l4\_cap\_idx\_t
```

Capability selector type.

A capability selector is either a (shifted) capability index or the invalid capability selector [L4_INVALID_CAP](#).

Usage of the invalid capability selector is defined only for invoking IPC (see [Object Invocation](#)): When IPC is invoked on [L4_INVALID_CAP](#), then it is resolved to a capability for the current thread with full permissions.

Otherwise, the API assumes that each argument of type [l4_cap_idx_t](#) is a capability index, i.e., $idx \ll \text{L4_CAP_SHIFT}$ for arbitrary idx . The behavior for other arguments is then undefined.

Examples

[examples/libs/shmc/prodcons.c](#), [examples/sys/aliens/main.c](#), [examples/sys/isr/main.c](#), [examples/sys/singlestep/main.c](#), [examples/sys/start-with-exc/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

Definition at line 336 of file [types.h](#).

14.1.12.3 Enumeration Type Documentation

14.1.12.3.1 l4_cap_consts_t

enum [l4_cap_consts_t](#)

Constants related to capability selectors.

Enumerator

L4_CAP_SHIFT	Capability index shift.
L4_CAP_SIZE	Deprecated Superseded by L4_CAP_OFFSET .
L4_CAP_OFFSET	Offset of two consecutive capability selectors.
L4_CAP_MASK	Mask to get only the relevant bits of an l4_cap_idx_t .
L4_INVALID_CAP	Invalid capability selector.

Definition at line [139](#) of file [consts.h](#).

14.1.12.3.2 l4_default_caps_t

enum [l4_default_caps_t](#)

Default capabilities setup for the initial tasks.

These capability selectors are setup per default by the micro kernel for the two initial tasks, the Root-Pager (Sigma0) and the Root-Task (Moe).

Attention

These constants do not have any particular meaning for applications started by Moe, see [Initial Environment](#) for this kind of information.

See also

[Initial Environment](#) for information useful for normal user applications.

Enumerator

L4_BASE_TASK_CAP	Capability selector for the current task.
L4_BASE_FACTORY_CAP	Capability selector for the factory.
L4_BASE_THREAD_CAP	Capability selector for the first thread.
L4_BASE_PAGER_CAP	Capability selector for the pager gate. For Sigma0, the pager is not present since it never raises page faults. For Moe, the pager is set to Sigma0.
L4_BASE_LOG_CAP	Capability selector for the log object. Present if the corresponding feature is turned on in the microkernel configuration.

L4_BASE_ICU_CAP	Capability selector for the base icu object.
L4_BASE_SCHEDULER_CAP	Capability selector for the scheduler cap.
L4_BASE_IOMMU_CAP	Capability selector for the IO-MMU cap. Present if the microkernel detected an IO-MMU.
L4_BASE_DEBUGGER_CAP	Capability selector for the debugger cap. Present if the corresponding feature is turned on in the microkernel configuration.
L4_BASE_ARM_SMCCC_CAP	Capability selector for the ARM SMCCC cap. Present if the microkernel detected an ARM SMC capable trusted execution environment.
L4_BASE_CAPS_LAST	Last capability index used for base capabilities.

Definition at line 324 of file [consts.h](#).

14.1.12.4 Function Documentation

14.1.12.4.1 l4_capability_equal()

```
unsigned l4_capability_equal (
    l4_cap_idx_t c1,
    l4_cap_idx_t c2) [inline]
```

Test if the capability indices of two capability selectors are equal.

Parameters

<i>c1</i>	Capability selector.
<i>c2</i>	Capability selector.

Return values

0	The index parts of the capability selectors differ.
1	The index parts of the capability selectors are equal.

Precondition

Both capability selectors must be valid (cf. [l4_is_valid_cap\(\)](#)) otherwise the return value is undefined.

Definition at line 397 of file [types.h](#).

References [L4_CAP_SHIFT](#), and [L4_NOTHROW](#).

14.1.12.4.2 l4_is_invalid_cap()

```
unsigned l4_is_invalid_cap (
    l4_cap_idx_t c) [inline]
```

Test if a capability selector is the invalid capability.

Parameters

<code>c</code>	Capability selector
----------------	---------------------

Return values

<code>0</code>	The capability selector is not the invalid capability.
<code>>0</code>	The capability selector is the invalid capability.

Examples

[examples/libs/l4re/c/ma+rm.c](#), [examples/sys/aliens/main.c](#), [examples/sys/isr/main.c](#), [examples/sys/singlestep/main.c](#), [examples/sys/start-with-exc/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

Definition at line 389 of file [types.h](#).

References [L4_NOTHROW](#).

14.1.12.4.3 l4_is_valid_cap()

```
unsigned l4_is_valid_cap (
    l4_cap_idx_t c) [inline]
```

Test if a capability selector is a valid selector.

Parameters

<code>c</code>	Capability selector
----------------	---------------------

Return values

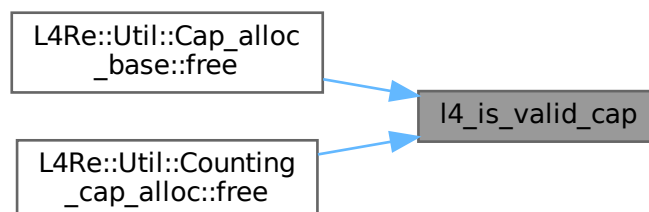
<code>0</code>	The capability selector is not valid.
<code>>0</code>	The capability selector is valid.

Definition at line 393 of file [types.h](#).

References [L4_NOTHROW](#).

Referenced by [L4Re::Util::Cap_alloc_base::free\(\)](#), and [L4Re::Util::Counting_cap_alloc< COUNTERTYPE, Dbg >::free\(\)](#).

Here is the caller graph for this function:



14.1.13 Memory operations.

Operations for memory access.

Collaboration diagram for Memory operations.:



Enumerations

- enum `L4_mem_op_widths` { `L4_MEM_WIDTH_1BYTE` = 0 , `L4_MEM_WIDTH_2BYTE` = 1 , `L4_MEM_WIDTH_4BYTE` = 2 }

Memory access width definitions.

Functions

- unsigned long `l4_mem_read` (unsigned long virtaddress, unsigned width)
Read user task memory from kernel privilege level.
- void `l4_mem_write` (unsigned long virtaddress, unsigned width, unsigned long value)
Write user task memory from kernel privilege level.

14.1.13.1 Detailed Description

Operations for memory access.

This module provides functionality to access user task memory from the kernel. This is needed for some devices that are only accessible from privileged processor mode. Only use this when absolutely required. This functionality is only available on the ARM architecture.

```
#include <l4/sys/mem_op.h>
```

14.1.13.2 Enumeration Type Documentation

14.1.13.2.1 L4_mem_op_widths

```
enum L4_mem_op_widths
```

Memory access width definitions.

Enumerator

L4_MEM_WIDTH_1BYTE	Access one byte (8-bit width).
L4_MEM_WIDTH_2BYTE	Access two bytes (16-bit width).
L4_MEM_WIDTH_4BYTE	Access four bytes (32-bit width).

Definition at line 40 of file [mem_op.h](#).

14.1.13.3 Function Documentation

14.1.13.3.1 l4_mem_read()

```
unsigned long l4_mem_read (  
    unsigned long virtaddress,  
    unsigned width) [inline]
```

Read user task memory from kernel privilege level.

Parameters

<i>virtaddress</i>	Virtual address in the calling task.
<i>width</i>	Width of access in bytes in log2,

See also

[L4_mem_op_widths](#)

Returns

Read value.

Upon an given invalid address or invalid width value the function does nothing.

Definition at line 130 of file [mem_op.h](#).

References [l4_mem_arm_op_call\(\)](#).

Here is the call graph for this function:



14.1.13.3.2 l4_mem_write()

```
void l4_mem_write (  
    unsigned long virtaddress,  
    unsigned width,  
    unsigned long value) [inline]
```

Write user task memory from kernel privilege level.

Parameters

<i>virtaddress</i>	Virtual address in the calling task.
<i>width</i>	Width of access in bytes in log2 (i.e. allowed values: 0, 1, 2)
<i>value</i>	Value to write.

Upon an given invalid address or invalid width value the function does nothing.

Definition at line 136 of file [mem_op.h](#).

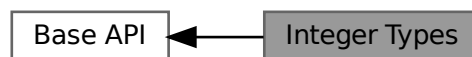
References [l4_mem_arm_op_call\(\)](#).

Here is the call graph for this function:



14.1.14 Integer Types

Collaboration diagram for Integer Types:



Files

- file [l4int.h](#)
Fixed sized integer types, generic version.
- file [l4int.h](#)
Fixed sized integer types, arm version.
- file [l4int.h](#)
Fixed sized integer types, arm version.
- file [l4int.h](#)
Fixed sized integer types, AMD64 version.
- file [l4int.h](#)
Fixed sized integer types, x86 version.

Macros

- `#define L4_MWORD_BITS 32`
Size of machine words in bits.
- `#define L4_MWORD_BITS 64`
Size of machine words in bits.
- `#define L4_MWORD_BITS 64`
Size of machine words in bits.
- `#define L4_MWORD_BITS 32`
Size of machine words in bits.

Typedefs

- `typedef signed char l4_int8_t`
Signed 8bit value.
- `typedef unsigned char l4_uint8_t`
Unsigned 8bit value.
- `typedef signed short int l4_int16_t`
Signed 16bit value.
- `typedef unsigned short int l4_uint16_t`
Unsigned 16bit value.
- `typedef signed int l4_int32_t`
Signed 32bit value.
- `typedef unsigned int l4_uint32_t`
Unsigned 32bit value.
- `typedef signed long long l4_int64_t`
Signed 64bit value.
- `typedef unsigned long long l4_uint64_t`
Unsigned 64bit value.
- `typedef unsigned long l4_addr_t`
Address type.
- `typedef signed long l4_mword_t`
Signed machine word.
- `typedef unsigned long l4_umword_t`
Unsigned machine word.
- `typedef l4_uint64_t l4_cpu_time_t`
CPU clock type.
- `typedef l4_uint64_t l4_kernel_clock_t`
Kernel clock type.
- `typedef unsigned int l4_size_t`
Unsigned size type.
- `typedef signed int l4_ssize_t`
Signed size type.
- `typedef unsigned long l4_size_t`
Unsigned size type.
- `typedef signed long l4_ssize_t`
Signed size type.
- `typedef unsigned long l4_size_t`
Unsigned size type.
- `typedef signed long l4_ssize_t`
Signed size type.
- `typedef unsigned int l4_size_t`
Unsigned size type.
- `typedef signed int l4_ssize_t`
Signed size type.

14.1.14.1 Detailed Description

Include File

```
#include <l4/sys/l4int.h>
```

14.2 EDID parsing functionality

Enumerations

- enum [Libedid_consts](#) { [Libedid_block_size](#) = 128 }
EDID constants.

Functions

- int [libedid_check_header](#) (const unsigned char *edid)
Check for valid EDID header.
- int [libedid_checksum](#) (const unsigned char *edid)
Calculates the EDID checksum.
- unsigned [libedid_version](#) (const unsigned char *edid)
Returns the EDID version number.
- unsigned [libedid_revision](#) (const unsigned char *edid)
Returns the EDID revision number.
- void [libedid_pnp_id](#) (const unsigned char *edid, unsigned char *id)
Extracts the display's PnP ID.
- void [libedid_preferred_resolution](#) (const unsigned char *edid, unsigned *w, unsigned *h)
Extract the display's preferred mode.
- unsigned [libedid_num_ext_blocks](#) (const unsigned char *edid)
Get the number of EDID extension blocks.
- unsigned [libedid_dump_standard_timings](#) (const unsigned char *edid)
Dump the standard timings to stdout.
- void [libedid_dump](#) (const unsigned char *edid)
Dump raw EDID data to stdout.

14.2.1 Detailed Description

14.2.2 Enumeration Type Documentation

14.2.2.1 Libedid_consts

```
enum Libedid\_consts
```

EDID constants.

Enumerator

<code>Libedid_block_size</code>	Size of one EDID block in bytes.
---------------------------------	----------------------------------

Definition at line 23 of file [edid.h](#).

14.2.3 Function Documentation

14.2.3.1 `libedid_check_header()`

```
int libedid_check_header (  
    const unsigned char * edid)
```

Check for valid EDID header.

Parameters

<i>edid</i>	Pointer to a 128byte EDID block
-------------	---------------------------------

Returns

0 if the header is correct, -EINVAL otherwise

14.2.3.2 `libedid_checksum()`

```
int libedid_checksum (  
    const unsigned char * edid)
```

Calculates the EDID checksum.

Parameters

<i>edid</i>	Pointer to a 128byte EDID block
-------------	---------------------------------

Returns

0 if checksum is correct, -EINVAL otherwise

14.2.3.3 `libedid_dump()`

```
void libedid_dump (  
    const unsigned char * edid)
```

Dump raw EDID data to stdout.

Parameters

<i>edid</i>	Pointer to a 128byte EDID block
-------------	---------------------------------

14.2.3.4 libedid_dump_standard_timings()

```
unsigned libedid_dump_standard_timings (  
    const unsigned char * edid)
```

Dump the standard timings to stdout.

Parameters

<i>edid</i>	Pointer to a 128byte EDID block
-------------	---------------------------------

Returns

Number of standard timings stored in EDID

14.2.3.5 libedid_num_ext_blocks()

```
unsigned libedid_num_ext_blocks (  
    const unsigned char * edid)
```

Get the number of EDID extension blocks.

Parameters

<i>edid</i>	Pointer to a 128byte EDID block
-------------	---------------------------------

Returns

Number of EDID extension blocks

14.2.3.6 libedid_pnp_id()

```
void libedid_pnp_id (  
    const unsigned char * edid,  
    unsigned char * id)
```

Extracts the display's PnP ID.

Parameters

	<i>edid</i>	Pointer to a 128byte EDID block
--	-------------	---------------------------------

out	<i>id</i>	Return the PnP id. Must point to 4 bytes.
-----	-----------	---

14.2.3.7 libedid_prefered_resolution()

```
void libedid_prefered_resolution (
    const unsigned char * edid,
    unsigned * w,
    unsigned * h)
```

Extract the display's preferred mode.

Parameters

	<i>edid</i>	Pointer to a 128byte EDID block
out	<i>w</i>	X resolution of preferred video mode in pixels.
out	<i>h</i>	Y resolution of preferred video mode in pixels.

14.2.3.8 libedid_revision()

```
unsigned libedid_revision (
    const unsigned char * edid)
```

Returns the EDID revision number.

Parameters

<i>edid</i>	Pointer to a 128 EDID block
-------------	-----------------------------

Returns

Revision number

14.2.3.9 libedid_version()

```
unsigned libedid_version (
    const unsigned char * edid)
```

Returns the EDID version number.

Parameters

<i>edid</i>	Pointer to a 128byte EDID block
-------------	---------------------------------

Returns

Version number

14.3 IO interface

Typedefs

- typedef `l4vbus_resource_t` `l4io_resource_t`
Resource descriptor.
- typedef `l4vbus_device_t` `l4io_device_t`
Device descriptor.

Enumerations

- enum `l4io_iomem_flags_t` {
`L4IO_MEM_NONCACHED` = 0 , `L4IO_MEM_CACHED` = 1 , `L4IO_MEM_USE_MTRR` = 2 , `L4IO_MEM_ATTR_MASK` = 0xf ,
`L4IO_MEM_WRITE_COMBINED` = `L4IO_MEM_USE_MTRR` | `L4IO_MEM_CACHED` , `L4IO_MEM_USE_RESERVED_AREA` = 0x40 << 8 , `L4IO_MEM_EAGER_MAP` = 0x80 << 8 }
Flags for IO memory.
- enum `l4io_device_types_t` {
`L4IO_DEVICE_INVALID` = 0 , `L4IO_DEVICE_PCI` , `L4IO_DEVICE_USB` , `L4IO_DEVICE_OTHER` ,
`L4IO_DEVICE_ANY` = ~0 }
Device types.
- enum `l4io_resource_types_t` {
`L4IO_RESOURCE_INVALID` = `L4VBUS_RESOURCE_INVALID` , `L4IO_RESOURCE_IRQ` = `L4VBUS_RESOURCE_IRQ` , `L4IO_RESOURCE_MEM` = `L4VBUS_RESOURCE_MEM` , `L4IO_RESOURCE_PORT` = `L4VBUS_RESOURCE_PORT` ,
`L4IO_RESOURCE_ANY` = ~0 }
Resource types.

Functions

- long `l4io_request_iomem` (`l4_addr_t` phys, unsigned long size, int flags, `l4_addr_t` *virt)
Request an IO memory region.
- long `l4io_request_iomem_region` (`l4_addr_t` phys, `l4_addr_t` virt, unsigned long size, int flags)
Request an IO memory region and map it to a specified region.
- long `l4io_release_iomem` (`l4_addr_t` virt, unsigned long size)
Release an IO memory region.
- long `l4io_request_ioport` (unsigned portnum, unsigned len)
Request an IO port region.
- long `l4io_release_ioport` (unsigned portnum, unsigned len)
Release an IO port region.
- int `l4io_lookup_device` (const char *devname, `l4io_device_handle_t` *dev_handle, `l4io_device_t` *dev, `l4io_resource_handle_t` *res_handle)
Find a device by name.
- int `l4io_lookup_resource` (`l4io_device_handle_t` devhandle, enum `l4io_resource_types_t` type, `l4io_resource_handle_t` *reshandle, `l4io_resource_t` *res)
Request a specific resource from a device description.
- `l4_addr_t` `l4io_request_resource_iomem` (`l4io_device_handle_t` devhandle, `l4io_resource_handle_t` *reshandle)
Request IO memory.
- int `l4io_has_resource` (enum `l4io_resource_types_t` type, `l4vbus_paddr_t` start, `l4vbus_paddr_t` end)
Check if a resource is available.

14.3.1 Detailed Description

14.3.2 Typedef Documentation

14.3.2.1 l4io_resource_t

```
typedef l4vbus_resource_t l4io_resource_t
```

Resource descriptor.

For IRQ types, the end field is not used, i.e. only a single interrupt can be described with a [l4io_resource_t](#)

Definition at line 67 of file [types.h](#).

14.3.3 Enumeration Type Documentation

14.3.3.1 l4io_device_types_t

```
enum l4io_device_types_t
```

Device types.

Enumerator

L4IO_DEVICE_INVALID	Invalid type.
L4IO_DEVICE_PCI	PCI device.
L4IO_DEVICE_USB	USB device.
L4IO_DEVICE_OTHER	Any other device without unique IDs.
L4IO_DEVICE_ANY	any type

Definition at line 36 of file [types.h](#).

14.3.3.2 l4io_iomem_flags_t

```
enum l4io_iomem_flags_t
```

Flags for IO memory.

Enumerator

L4IO_MEM_NONCACHED	Non-cache memory.
L4IO_MEM_CACHED	Cache memory.
L4IO_MEM_USE_MTRR	Use MTRR.
L4IO_MEM_USE_RESERVED_AREA	Use reserved area for mapping I/O memory. Flag only valid for l4io_request_iomem_region()

L4IO_MEM_EAGER_MAP	Eagerly map the I/O memory. Passthrough to the l4re-rm.
--------------------	---

Definition at line 14 of file [types.h](#).

14.3.3.3 l4io_resource_types_t

enum [l4io_resource_types_t](#)

Resource types.

Enumerator

L4IO_RESOURCE_INVALID	Invalid type.
L4IO_RESOURCE_IRQ	Interrupt resource.
L4IO_RESOURCE_MEM	I/O memory resource.
L4IO_RESOURCE_PORT	I/O port resource (x86 only).
L4IO_RESOURCE_ANY	any type

Definition at line 48 of file [types.h](#).

14.3.4 Function Documentation

14.3.4.1 l4io_has_resource()

```
int l4io_has_resource (
    enum l4io\_resource\_types\_t type,
    l4vbus\_paddr\_t start,
    l4vbus\_paddr\_t end)
```

Check if a resource is available.

Parameters

<i>type</i>	Type of resource
<i>start</i>	Minimal value.
<i>end</i>	Maximum value.

References [L4_INLINE](#).

14.3.4.2 l4io_lookup_device()

```
int l4io_lookup_device (
    const char * devname,
    l4io\_device\_handle\_t * dev_handle,
    l4io\_device\_t * dev,
    l4io\_resource\_handle\_t * res_handle)
```

Find a device by name.

Parameters

	<i>devname</i>	Name of device.
out	<i>dev_handle</i>	Device handle for found device, can be NULL.
out	<i>dev</i>	Device information, filled by the function, can be NULL.
out	<i>res_handle</i>	Resource handle, can be NULL.

Returns

0 on success, error code otherwise

References [L4_CV](#), and [L4_EXPORT](#).

14.3.4.3 l4io_lookup_resource()

```
int l4io_lookup_resource (
    l4io_device_handle_t devhandle,
    enum l4io_resource_types_t type,
    l4io_resource_handle_t * reshandle,
    l4io_resource_t * res)
```

Request a specific resource from a device description.

Parameters

	<i>devhandle</i>	Device handle.
	<i>type</i>	Type of resource to request (see l4io_resource_types_t).
in, out	<i>reshandle</i>	Resource handle, start with handle returned by device functions. The next resource handle is returned here.
out	<i>res</i>	Device descriptor.

Returns

0 on success, error code otherwise, esp. -L4_ENOENT if no more resources found

References [L4_CV](#), and [L4_EXPORT](#).

14.3.4.4 l4io_release_iomem()

```
long l4io_release_iomem (
    l4_addr_t virt,
    unsigned long size)
```

Release an IO memory region.

Parameters

<i>virt</i>	Virtual address of region to free, see l4io_request_iomem
<i>size</i>	Size of the region to release.

Returns

0 on success, <0 on error

References [L4_CV](#), and [L4_EXPORT](#).

14.3.4.5 l4io_release_ioport()

```
long l4io_release_ioport (  
    unsigned portnum,  
    unsigned len)
```

Release an IO port region.

Parameters

<i>portnum</i>	Start of port range to release
<i>len</i>	Length of range to request

Returns

0 on success, <0 on error

Note

X86 architecture only

References [L4_CV](#), [L4_EXPORT](#), and [L4_INLINE](#).

14.3.4.6 l4io_request_iomem()

```
long l4io_request_iomem (  
    l4_addr_t phys,  
    unsigned long size,  
    int flags,  
    l4_addr_t * virt)
```

Request an IO memory region.

Parameters

	<i>phys</i>	Physical address of the I/O memory region
--	-------------	---

	<i>size</i>	Size of the region in Bytes, granularity pages.
	<i>flags</i>	See l4io_iomem_flags_t
<i>in, out</i>	<i>virt</i>	Virtual address where the IO memory region should be mapped to. If the caller passes '0' a region in the caller's address space is searched and the virtual address is returned.

Return values

<i>0</i>	Success.
<i>-L4_ENOENT</i>	No area in the caller's address space could be found to map the IO memory region.
<i>-L4_EPERM</i>	Operation not allowed.
<i>-L4_EINVAL</i>	Invalid value.
<i>-L4_EADDRNOTAVAIL</i>	The requested virtual address is not available.
<i>-L4_ENOMEM</i>	The requested IO memory region could not be allocated.
<i><0</i>	IPC errors.

Note

This function uses [L4Re](#) functionality to reserve a part of the virtual address space of the caller.

References [L4_CV](#), and [L4_EXPORT](#).

14.3.4.7 l4io_request_iomem_region()

```
long l4io_request_iomem_region (
    l4_addr_t phys,
    l4_addr_t virt,
    unsigned long size,
    int flags)
```

Request an IO memory region and map it to a specified region.

Parameters

<i>phys</i>	Physical address of the I/O memory region
<i>virt</i>	Virtual address.
<i>size</i>	Size of the region in Bytes, granularity pages.
<i>flags</i>	See l4io_iomem_flags_t

Return values

<i>0</i>	Success.
<i>-L4_ENOENT</i>	No area could be found to map the IO memory region.
<i>-L4_EPERM</i>	Operation not allowed.

<code>-L4_EINVAL</code>	Invalid value.
<code>-L4_EADDRNOTAVAIL</code>	The requested virtual address is not available.
<code>-L4_ENOMEM</code>	The requested IO memory region could not be allocated.
<code><0</code>	IPC errors.

Note

This function uses [L4Re](#) functionality to reserve a part of the virtual address space of the caller.

References [L4_CV](#), and [L4_EXPORT](#).

14.3.4.8 l4io_request_ioport()

```
long l4io_request_ioport (
    unsigned portnum,
    unsigned len)
```

Request an IO port region.

Parameters

<i>portnum</i>	Start of port range to request
<i>len</i>	Length of range to request

Returns

0 on success, <0 on error

Note

X86 architecture only

References [L4_CV](#), and [L4_EXPORT](#).

14.3.4.9 l4io_request_resource_iomem()

```
l4\_addr\_t l4io_request_resource_iomem (
    l4io_device_handle_t devhandle,
    l4io_resource_handle_t * reshandle)
```

Request IO memory.

Parameters

	<i>devhandle</i>	Device handle.
--	------------------	----------------

<code>in, out</code>	<code>reshandle</code>	Resource handle from which IO memory should be requested. Upon successful completion 'reshandle' points to the device's next resource.
----------------------	------------------------	--

Return values

<code>0</code>	An error occurred. The value of 'reshandle' is undefined.
<code>>0</code>	The virtual address of the IO memory mapping.

References [L4_CV](#), and [L4_EXPORT](#).

14.4 IPC Helpers

Functions

- void [L4::throw_ipc_exception](#) ([L4::Cap](#)< void > const &o, [l4_msgtag_t](#) const &err, [l4_utcb_t](#) *utcb)
Throw an [L4](#) IPC error as exception.
- void [L4::throw_ipc_exception](#) (void const *o, [l4_msgtag_t](#) const &err, [l4_utcb_t](#) *utcb)
Throw an [L4](#) IPC error as exception.

14.4.1 Detailed Description

14.4.2 Function Documentation

14.4.2.1 [throw_ipc_exception\(\)](#) [1/2]

```
void L4::throw_ipc_exception (
    L4::Cap< void > const & o,
    l4\_msgtag\_t const & err,
    l4\_utcb\_t * utcb) [inline]
```

Throw an [L4](#) IPC error as exception.

Parameters

<code>o</code>	The client side object, for which the IPC was invoked.
<code>err</code>	The IPC result code (error code).
<code>utcb</code>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Definition at line 34 of file [ipc_helper](#).

References [l4_msgtag_t::has_error\(\)](#).

Referenced by [throw_ipc_exception\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.4.2.2 throw_ipc_exception() [2/2]

```

void L4::throw_ipc_exception (
    void const * o,
    l4_msgtag_t const & err,
    l4_utcb_t * utcb) [inline]
  
```

Throw an [L4](#) IPC error as exception.

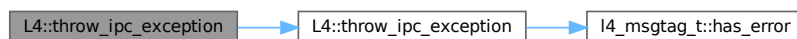
Parameters

<i>o</i>	The client side object, for which the IPC was invoked.
<i>err</i>	The IPC result code (error code).
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Definition at line [50](#) of file [ipc_helper](#).

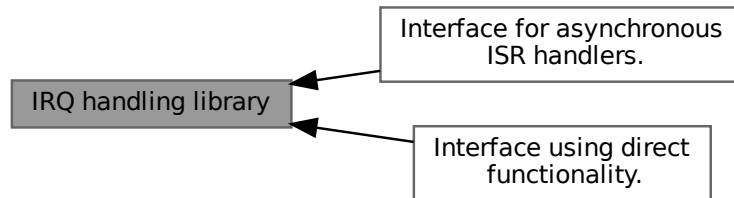
References [throw_ipc_exception\(\)](#).

Here is the call graph for this function:



14.5 IRQ handling library

Collaboration diagram for IRQ handling library:



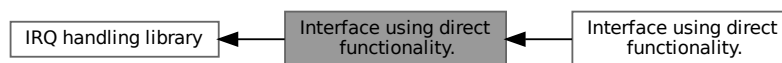
Topics

- Interface using direct functionality. ??
- Interface for asynchronous ISR handlers. ??
This interface has just two (main) functions.

14.5.1 Detailed Description

14.5.2 Interface using direct functionality.

Collaboration diagram for Interface using direct functionality.:



Topics

- Interface using direct functionality. ??

Functions

- `l4irq_t * l4irq_attach` (int irqnum)
Attach/connect to IRQ.
- `l4irq_t * l4irq_attach_ft` (int irqnum, unsigned mode)
Attach/connect to IRQ using given type.
- `l4irq_t * l4irq_attach_thread` (int irqnum, `l4_cap_idx_t` to_thread)
Attach/connect to IRQ.
- `l4irq_t * l4irq_attach_thread_ft` (int irqnum, `l4_cap_idx_t` to_thread, unsigned mode)
Attach/connect to IRQ using given type.
- `long l4irq_wait` (`l4irq_t *irq`)
Wait for specified IRQ.
- `long l4irq_unmask_and_wait_any` (`l4irq_t *unmask_irq`, `l4irq_t **ret_irq`)
Unmask a specific IRQ and wait for any attached IRQ.
- `long l4irq_wait_any` (`l4irq_t **irq`)
Wait for any attached IRQ.
- `long l4irq_unmask` (`l4irq_t *irq`)
Unmask a specific IRQ.
- `long l4irq_detach` (`l4irq_t *irq`)
Detach from IRQ.

14.5.2.1 Detailed Description

14.5.2.2 Function Documentation

14.5.2.2.1 `l4irq_attach()`

```
l4irq_t * l4irq_attach (
    int irqnum)
```

Attach/connect to IRQ.

Parameters

<code>irqnum</code>	IRQ number to request
---------------------	-----------------------

Returns

Pointer to `l4irq_t` structure, 0 on error

This `l4irq_attach` has to be called in the same thread as `l4irq_wait` and caller has to be a pthread thread.

Examples

[examples/libs/libirq/loop.c](#).

References [L4_CV](#).

14.5.2.2.2 `l4irq_attach_ft()`

```
l4irq_t * l4irq_attach_ft (
    int irqnum,
    unsigned mode)
```

Attach/connect to IRQ using given type.

Parameters

<i>irqnum</i>	IRQ number to request
<i>mode</i>	Interrupt type,

See also

[L4_irq_mode](#)

Returns

Pointer to `l4irq_t` structure, 0 on error

This `l4irq_attach` has to be called in the same thread as `l4irq_wait` and caller has to be a pthread thread.

References [L4_CV](#).

14.5.2.2.3 l4irq_attach_thread()

```
l4irq_t * l4irq_attach_thread (
    int irqnum,
    l4_cap_idx_t to_thread)
```

Attach/connect to IRQ.

Parameters

<i>irqnum</i>	IRQ number to request
<i>to_thread</i>	Attach IRQ to this specified thread.

Returns

Pointer to `l4irq_t` structure, 0 on error

The pointer to the IRQ structure is used as a label in the IRQ object.

References [L4_CV](#).

14.5.2.2.4 l4irq_attach_thread_ft()

```
l4irq_t * l4irq_attach_thread_ft (
    int irqnum,
    l4_cap_idx_t to_thread,
    unsigned mode)
```

Attach/connect to IRQ using given type.

Parameters

<i>irqnum</i>	IRQ number to request
<i>to_thread</i>	Attach IRQ to this specified thread.
<i>mode</i>	Interrupt type,

See also

[L4_irq_mode](#)

Returns

Pointer to `l4irq_t` structure, 0 on error

The pointer to the IRQ structure is used as a label in the IRQ object.

References [L4_CV](#).

14.5.2.2.5 l4irq_detach()

```
long l4irq_detach (  
    l4irq_t * irq)
```

Detach from IRQ.

Parameters

<i>irq</i>	IRQ data structure
------------	--------------------

Returns

0 on success, != 0 on error

References [L4_CV](#).

14.5.2.2.6 l4irq_unmask()

```
long l4irq_unmask (  
    l4irq_t * irq)
```

Unmask a specific IRQ.

Parameters

<i>irq</i>	IRQ data structure
------------	--------------------

Returns

0 on success, != 0 on error

This function is useful if a thread wants to wait for multiple IRQs using `l4_ipc_wait`.

References [L4_CV](#).

14.5.2.2.7 l4irq_unmask_and_wait_any()

```
long l4irq_unmask_and_wait_any (
    l4irq_t * unmask_irq,
    l4irq_t ** ret_irq)
```

Unmask a specific IRQ and wait for any attached IRQ.

Parameters

	<i>unmask_irq</i>	IRQ data structure for unmask.
out	<i>ret_irq</i>	Received interrupt.

Returns

0 on success, != 0 on error

References [L4_CV](#).

14.5.2.2.8 l4irq_wait()

```
long l4irq_wait (
    l4irq_t * irq)
```

Wait for specified IRQ.

Parameters

<i>irq</i>	IRQ data structure
------------	--------------------

Returns

0 on success, != 0 on error

Examples

[examples/libs/libirq/loop.c](#).

References [L4_CV](#).

14.5.2.2.9 l4irq_wait_any()

```
long l4irq_wait_any (
    l4irq_t ** irq)
```

Wait for any attached IRQ.

Return values

<i>irq</i>	Received interrupt.
------------	---------------------

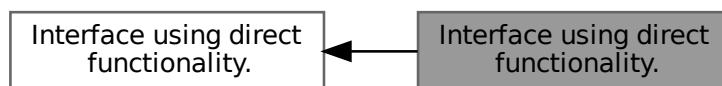
Returns

0 on success, != 0 on error

References [L4_CV](#).

14.5.2.3 Interface using direct functionality.

Collaboration diagram for Interface using direct functionality.:

**Functions**

- `l4irq_t * l4irq_attach_cap (l4_cap_idx_t irqcap)`
Attach/connect to IRQ.
- `l4irq_t * l4irq_attach_cap_ft (l4_cap_idx_t irqcap, unsigned mode)`
Attach/connect to IRQ using given type.
- `l4irq_t * l4irq_attach_thread_cap (l4_cap_idx_t irqcap, l4_cap_idx_t to_thread)`
Attach/connect to IRQ.
- `l4irq_t * l4irq_attach_thread_cap_ft (l4_cap_idx_t irqcap, l4_cap_idx_t to_thread, unsigned mode)`
Attach/connect to IRQ using given type.

14.5.2.3.1 Detailed Description**14.5.2.3.2 Function Documentation****14.5.2.3.2.1 l4irq_attach_cap()**

```
l4irq_t * l4irq_attach_cap (
    l4_cap_idx_t irqcap)
```

Attach/connect to IRQ.

Parameters

<i>irqcap</i>	IRQ capability
---------------	----------------

Returns

Pointer to `l4irq_t` structure, 0 on error

This `l4irq_attach` has to be called in the same thread as `l4irq_wait` and caller has to be a pthread thread.

References [L4_CV](#).

14.5.2.3.2.2 l4irq_attach_cap_ft()

```
l4irq_t * l4irq_attach_cap_ft (
    l4_cap_idx_t irqcap,
    unsigned mode)
```

Attach/connect to IRQ using given type.

Parameters

<i>irqcap</i>	IRQ capability
<i>mode</i>	Interrupt type,

See also

[L4_irq_mode](#)

Returns

Pointer to `l4irq_t` structure, 0 on error

This `l4irq_attach` has to be called in the same thread as `l4irq_wait` and caller has to be a pthread thread.

References [L4_CV](#).

14.5.2.3.2.3 l4irq_attach_thread_cap()

```
l4irq_t * l4irq_attach_thread_cap (
    l4_cap_idx_t irqcap,
    l4_cap_idx_t to_thread)
```

Attach/connect to IRQ.

Parameters

<i>irqcap</i>	IRQ capability
---------------	----------------

<i>to_thread</i>	Attach IRQ to this thread.
------------------	----------------------------

Returns

Pointer to `l4irq_t` structure, 0 on error

The pointer to the IRQ structure is used as a label in the IRQ object.

References [L4_CV](#).

14.5.2.3.2.4 l4irq_attach_thread_cap_ft()

```
l4irq_t * l4irq_attach_thread_cap_ft (
    l4_cap_idx_t irqcap,
    l4_cap_idx_t to_thread,
    unsigned mode)
```

Attach/connect to IRQ using given type.

Parameters

<i>irqcap</i>	IRQ capability
<i>to_thread</i>	Attach IRQ to this thread.
<i>mode</i>	Interrupt type,

See also

[L4_irq_mode](#)

Returns

Pointer to `l4irq_t` structure, 0 on error

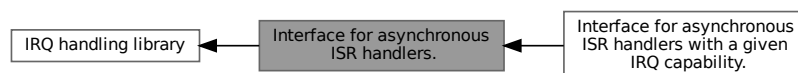
The pointer to the IRQ structure is used as a label in the IRQ object.

References [L4_CV](#).

14.5.3 Interface for asynchronous ISR handlers.

This interface has just two (main) functions.

Collaboration diagram for Interface for asynchronous ISR handlers.:



Topics

- Interface for asynchronous ISR handlers with a given IRQ capability. ??
This group is just an enhanced version to [l4irq_request\(\)](#) which takes a capability object instead of a plain number.

Functions

- `l4irq_t * l4irq_request` (int irqnum, void(*isr_handler)(void *), void *isr_data, int irq_thread_prio, unsigned mode)
Attach asynchronous ISR handler to IRQ.
- `long l4irq_release` (l4irq_t *irq)
Release asynchronous ISR handler and free resources.

14.5.3.1 Detailed Description

This interface has just two (main) functions.

`l4irq_request` to install a handler for an interrupt and `l4irq_release` to uninstall the handler again and release all resources associated with it.

14.5.3.2 Function Documentation

14.5.3.2.1 l4irq_release()

```
long l4irq_release (
    l4irq_t * irq)
```

Release asynchronous ISR handler and free resources.

Parameters

<i>irq</i>	IRQ data structure
------------	--------------------

Returns

0 success, != 0 failure

Examples

[examples/libs/libirq/async_isr.c](#).

References [L4_CV](#).

14.5.3.2.2 l4irq_request()

```
l4irq_t * l4irq_request (
    int irqnum,
    void(* isr_handler ) (void *),
    void * isr_data,
    int irq_thread_prio,
    unsigned mode)
```

Attach asynchronous ISR handler to IRQ.

Parameters

<i>irqnum</i>	IRQ number to request
<i>isr_handler</i>	Handler routine that is called when an interrupt triggers
<i>isr_data</i>	Pointer given as argument to <i>isr_handler</i>
<i>irq_thread_prio</i>	L4 thread priority of the ISR handler. Give -1 for same priority as creator.
<i>mode</i>	Interrupt type,

See also

[L4_irq_mode](#)

Returns

Pointer to `l4irq_t` structure, 0 on error

Examples

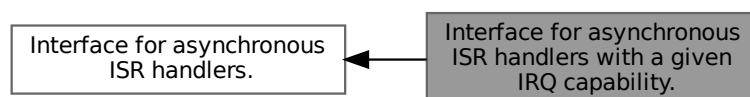
[examples/libs/libirq/async_isr.c](#).

References [L4_CV](#).

14.5.3.3 Interface for asynchronous ISR handlers with a given IRQ capability.

This group is just an enhanced version to [l4irq_request\(\)](#) which takes a capability object instead of a plain number.

Collaboration diagram for Interface for asynchronous ISR handlers with a given IRQ capability.:

**Functions**

- `l4irq_t * l4irq_request_cap (l4_cap_idx_t irqcap, void(*isr_handler)(void *), void *isr_data, int irq_thread_↔prio, unsigned mode)`

Attach asynchronous ISR handler to IRQ.

14.5.3.3.1 Detailed Description

This group is just an enhanced version to [l4irq_request\(\)](#) which takes a capability object instead of a plain number.

14.5.3.3.2 Function Documentation

14.5.3.3.2.1 l4irq_request_cap()

```
l4irq_t * l4irq_request_cap (
    l4_cap_idx_t irqcap,
    void(* isr_handler ) (void *),
    void * isr_data,
    int irq_thread_prio,
    unsigned mode)
```

Attach asynchronous ISR handler to IRQ.

Parameters

<i>irqcap</i>	IRQ capability
<i>isr_handler</i>	Handler routine that is called when an interrupt triggers
<i>isr_data</i>	Pointer given as argument to isr_handler
<i>irq_thread_prio</i>	L4 thread priority of the ISR handler. Give -1 for same priority as creator.
<i>mode</i>	Interrupt type,

See also

[L4_irq_mode](#)

Returns

Pointer to l4irq_t structure, 0 on error

14.6 L4 IPC Opcodes

List of protocol specific opcodes used for communication with [L4Re](#) and Kernel objects.

Enumerations

- enum [L4_icu_opcode](#) {
[L4_ICU_OP_BIND](#) , [L4_ICU_OP_UNBIND](#) , [L4_ICU_OP_INFO](#) , [L4_ICU_OP_MSI_INFO](#) ,
[L4_ICU_OP_UNMASK](#) , [L4_ICU_OP_MASK](#) , [L4_ICU_OP_SET_MODE](#) }
Opcodes to the ICU interface.
- enum [L4_ipc_gate_ops](#) { [L4_IPC_GATE_BIND_OP](#) = 0x10 , [L4_IPC_GATE_GET_INFO_OP](#) = 0x11 }
Operations on the IPC-gate.
- enum [L4_platform_ctl_ops](#) {
[L4_PLATFORM_CTL_SYS_SUSPEND_OP](#) = 0UL , [L4_PLATFORM_CTL_SYS_SHUTDOWN_OP](#) = 1UL ,
[L4_PLATFORM_CTL_CPU_ALLOW_SHUTDOWN_OP](#) = 2UL , [L4_PLATFORM_CTL_CPU_ENABLE_OP](#) =
3UL ,
[L4_PLATFORM_CTL_CPU_DISABLE_OP](#) = 4UL , [L4_PLATFORM_CTL_SET_TASK_ASID_OP](#) = 0x10UL }
Operations on platform-control objects.

- enum `L4_task_ops` {
`L4_TASK_MAP_OP` = 0UL , `L4_TASK_UNMAP_OP` = 1UL , `L4_TASK_CAP_INFO_OP` = 2UL ,
`L4_TASK_ADD_KU_MEM_OP` = 3UL ,
`L4_TASK_LDT_SET_X86_OP` = 0x11UL , `L4_TASK_MAP_VGICC_ARM_OP` = 0x12UL }
Operations on task objects.
- enum `L4_thread_ops` {
`L4_THREAD_CONTROL_OP` = 0UL , `L4_THREAD_EX_REGS_OP` = 1UL , `L4_THREAD_SWITCH_OP` =
2UL , `L4_THREAD_STATS_OP` = 3UL ,
`L4_THREAD_VCPU_RESUME_OP` = 4UL , `L4_THREAD_REGISTER_DELETE_IRQ_OP` = 5UL ,
`L4_THREAD_MODIFY_SENDER_OP` = 6UL , `L4_THREAD_VCPU_CONTROL_OP` = 7UL ,
`L4_THREAD_VCPU_CONTROL_EXT_OP` = `L4_THREAD_VCPU_CONTROL_OP` | 0x10000 , `L4_THREAD_REGISTER_DO`
= 8UL , `L4_THREAD_X86_GDT_OP` = 0x10UL , `L4_THREAD_ARM_TPIDRURO_OP` = 0x10UL ,
`L4_THREAD_AMD64_SET_SEGMENT_BASE_OP` = 0x12UL , `L4_THREAD_AMD64_GET_SEGMENT_INFO_OP`
= 0x13UL , `L4_THREAD_OPCODE_MASK` = 0xffff }
Operations on thread objects.
- enum `L4_vcon_ops` { `L4_VCON_WRITE_OP` = 0UL , `L4_VCON_READ_OP` = 1UL , `L4_VCON_SET_ATTR_OP`
= 2UL , `L4_VCON_GET_ATTR_OP` = 3UL }
Operations on vcon objects.

14.6.1 Detailed Description

List of protocol specific opcodes used for communication with [L4Re](#) and Kernel objects.

14.6.2 Enumeration Type Documentation

14.6.2.1 L4_icu_opcode

enum `L4_icu_opcode`

Opcodes to the ICU interface.

Enumerator

<code>L4_ICU_OP_BIND</code>	Bind opcode. See also l4_icu_bind()
<code>L4_ICU_OP_UNBIND</code>	Unbind opcode. See also l4_icu_unbind()
<code>L4_ICU_OP_INFO</code>	Info opcode. See also l4_icu_info()

L4_ICU_OP_MSI_INFO	Msi-info opcode. See also l4_icu_msi_info()
L4_ICU_OP_UNMASK	Unmask opcode. See also l4_icu_unmask()
L4_ICU_OP_MASK	Mask opcode. See also l4_icu_mask()
L4_ICU_OP_SET_MODE	Set-mode opcode. See also l4_icu_set_mode()

Definition at line 97 of file [icu.h](#).

14.6.2.2 L4_ipc_gate_ops

```
enum L4_ipc_gate_ops
```

Operations on the IPC-gate.

Enumerator

L4_IPC_GATE_BIND_OP	Bind operation.
L4_IPC_GATE_GET_INFO_OP	Info operation.

Definition at line 108 of file [ipc_gate.h](#).

14.6.2.3 L4_platform_ctl_ops

```
enum L4_platform_ctl_ops
```

Operations on platform-control objects.

See [L4_PROTO_PLATFORM_CTL](#) for the protocol type to use for messages to platform-control objects.

Enumerator

L4_PLATFORM_CTL_SYS_SUSPEND_OP	Suspend.
--------------------------------	----------

L4_PLATFORM_CTL_SYS_SHUTDOWN_OP	shutdown/reboot
L4_PLATFORM_CTL_CPU_ALLOW_SHUTDOWN_OP	allow CPU shutdown
L4_PLATFORM_CTL_CPU_ENABLE_OP	enable an offline CPU
L4_PLATFORM_CTL_CPU_DISABLE_OP	disable an online CPU
L4_PLATFORM_CTL_SET_TASK_ASID_OP	Arm: set task ASID.

Definition at line 159 of file [platform_control.h](#).

14.6.2.4 L4_task_ops

enum [L4_task_ops](#)

Operations on task objects.

Enumerator

L4_TASK_MAP_OP	Map.
L4_TASK_UNMAP_OP	Unmap.
L4_TASK_CAP_INFO_OP	Cap info.
L4_TASK_ADD_KU_MEM_OP	Add kernel-user memory.
L4_TASK_LDT_SET_X86_OP	x86: LDT set
L4_TASK_MAP_VGICC_ARM_OP	Arm: Map virtual GICC area.

Definition at line 341 of file [task.h](#).

14.6.2.5 L4_thread_ops

enum [L4_thread_ops](#)

Operations on thread objects.

Enumerator

L4_THREAD_CONTROL_OP	Control operation.
L4_THREAD_EX_REGS_OP	Exchange registers operation.
L4_THREAD_SWITCH_OP	Do a thread switch.
L4_THREAD_STATS_OP	Thread statistics.
L4_THREAD_VCPU_RESUME_OP	VCPU resume.
L4_THREAD_REGISTER_DELETE_IRQ_OP	Register an IPC-gate deletion IRQ.
L4_THREAD_MODIFY_SENDER_OP	Modify all senders IDs that match the given pattern.
L4_THREAD_VCPU_CONTROL_OP	Enable / disable VCPU feature.
L4_THREAD_REGISTER_DOORBELL_IRQ_OP	Register direct IRQ injection doorbell IRQ.
L4_THREAD_X86_GDT_OP	Gdt.

L4_THREAD_ARM_TPIDRURO_OP	Set TPIDRURO register.
L4_THREAD_AMD64_SET_SEGMENT_BASE_OP	Set segment base.
L4_THREAD_AMD64_GET_SEGMENT_INFO_OP	Get segment information.
L4_THREAD_OPCODE_MASK	Mask for opcodes.

Definition at line 732 of file [thread.h](#).

14.6.2.6 L4_vcon_ops

enum [L4_vcon_ops](#)

Operations on vcon objects.

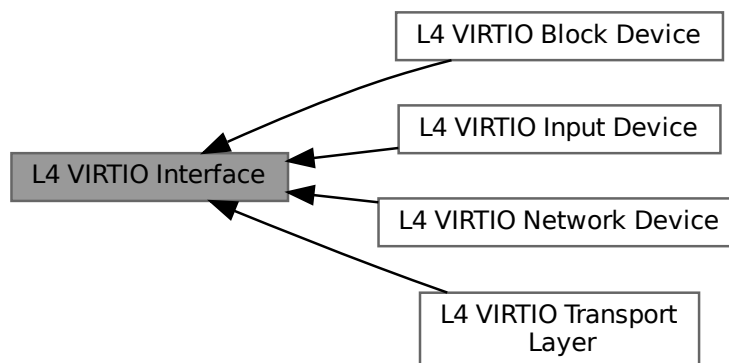
Enumerator

L4_VCON_WRITE_OP	Write.
L4_VCON_READ_OP	Read.
L4_VCON_SET_ATTR_OP	Get console attributes.
L4_VCON_GET_ATTR_OP	Set console attributes.

Definition at line 291 of file [vcon.h](#).

14.7 L4 VIRTIO Interface

Collaboration diagram for L4 VIRTIO Interface:



Topics

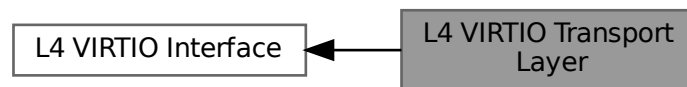
- [L4 VIRTIO Transport Layer](#) ??
L4 specific VIRTIO Transport layer.
- [L4 VIRTIO Block Device](#) ??
- [L4 VIRTIO Input Device](#) ??
- [L4 VIRTIO Network Device](#) ??

14.7.1 Detailed Description

14.7.2 L4 VIRTIO Transport Layer

[L4](#) specific VIRTIO Transport layer.

Collaboration diagram for L4 VIRTIO Transport Layer:



Namespaces

- namespace [L4virtio](#)
L4-VIRTIO Transport C++ API.

Data Structures

- struct [l4virtio_config_hdr_t](#)
L4-VIRTIO config header, provided in shared data space.
- struct [l4virtio_config_queue_t](#)
Queue configuration entry.

Typedefs

- typedef struct l4virtio_config_hdr_t [l4virtio_config_hdr_t](#)
L4-VIRTIO config header, provided in shared data space.
- typedef struct l4virtio_config_queue_t [l4virtio_config_queue_t](#)
Queue configuration entry.

Enumerations

- enum [L4_virtio_protocol](#)
L4-VIRTIO protocol number.
- enum [L4_virtio_opcodes](#) {
[L4VIRTIO_OP_SET_STATUS](#) = 0 , [L4VIRTIO_OP_CONFIG_QUEUE](#) = 1 , [L4VIRTIO_OP_REGISTER_DS](#) = 3 , [L4VIRTIO_OP_DEVICE_CONFIG](#) = 4 ,
[L4VIRTIO_OP_GET_DEVICE_IRQ](#) = 5 }
Opcodes to setup and configure a device.
- enum [L4virtio_device_ids](#) {
[L4VIRTIO_ID_NET](#) = 1 , [L4VIRTIO_ID_BLOCK](#) = 2 , [L4VIRTIO_ID_CONSOLE](#) = 3 , [L4VIRTIO_ID_RNG](#) = 4
,
[L4VIRTIO_ID_BALLOON](#) = 5 , [L4VIRTIO_ID_RPMMSG](#) = 7 , [L4VIRTIO_ID_SCSI](#) = 8 , [L4VIRTIO_ID_9P](#) = 9 ,
[L4VIRTIO_ID_RPROC_SERIAL](#) = 11 , [L4VIRTIO_ID_CAIF](#) = 12 , [L4VIRTIO_ID_GPU](#) = 16 , [L4VIRTIO_ID_INPUT](#) = 18 ,
[L4VIRTIO_ID_VSOCK](#) = 19 , [L4VIRTIO_ID_CRYPTIO](#) = 20 , [L4VIRTIO_ID_FS](#) = 26 , [L4VIRTIO_ID_SCMI](#) = 32 ,
[L4VIRTIO_ID_I2C](#) = 34 , [L4VIRTIO_ID_GPIO](#) = 41 , [L4VIRTIO_ID_SOCKET](#) = 0x9999 }
Virtio device IDs as reported in the driver's config space.
- enum [L4virtio_device_status](#) {
[L4VIRTIO_STATUS_ACKNOWLEDGE](#) = 1 , [L4VIRTIO_STATUS_DRIVER](#) = 2 , [L4VIRTIO_STATUS_DRIVER_OK](#) = 4 , [L4VIRTIO_STATUS_FEATURES_OK](#) = 8 ,
[L4VIRTIO_STATUS_DEVICE_NEEDS_RESET](#) = 0x40 , [L4VIRTIO_STATUS_FAILED](#) = 0x80 }
Virtio device status bits.
- enum [L4virtio_feature_bits](#) { [L4VIRTIO_FEATURE_VERSION_1](#) = 32 , [L4VIRTIO_FEATURE_CMD_CONFIG](#) = 160 }
L4virtio-specific feature bits.
- enum [L4_virtio_irq_status](#) { [L4VIRTIO_IRQ_STATUS_VRING](#) = 1 , [L4VIRTIO_IRQ_STATUS_CONFIG](#) = 2 }
VIRTIO IRQ status codes (l4virtio_config_hdr_t::irq_status).
- enum [L4_virtio_cmd](#) {
[L4VIRTIO_CMD_NONE](#) = 0x00000000 , [L4VIRTIO_CMD_SET_STATUS](#) = 0x01000000 , [L4VIRTIO_CMD_CFG_QUEUE](#) = 0x02000000 , [L4VIRTIO_CMD_CFG_CHANGED](#) = 0x04000000 ,
[L4VIRTIO_CMD_NOTIFY_QUEUE](#) = 0x08000000 , [L4VIRTIO_CMD_MASK](#) = 0xff000000 }
Virtio commands for device configuration.

Functions

- [L4_BEGIN_DECLS](#) [l4virtio_config_queue_t](#) * [l4virtio_config_queues](#) ([l4virtio_config_hdr_t](#) const *cfg)
Get the pointer to the first queue config.
- void * [l4virtio_device_config](#) ([l4virtio_config_hdr_t](#) const *cfg)
Get the pointer to the device configuration.
- void [l4virtio_set_feature](#) ([l4_uint32_t](#) *feature_map, unsigned feat)
Set the given feature bit in a feature map.
- void [l4virtio_clear_feature](#) ([l4_uint32_t](#) *feature_map, unsigned feat)
Clear the given feature bit in a feature map.
- unsigned [l4virtio_get_feature](#) ([l4_uint32_t](#) *feature_map, unsigned feat)
Check if the given bit in a feature map is set.
- int [l4virtio_set_status](#) ([l4_cap_idx_t](#) cap, unsigned status) [L4_NOTHROW](#)
- int [l4virtio_config_queue](#) ([l4_cap_idx_t](#) cap, unsigned queue) [L4_NOTHROW](#)
- int [l4virtio_register_ds](#) ([l4_cap_idx_t](#) cap, [l4_cap_idx_t](#) ds_cap, [l4_uint64_t](#) base, [l4_umword_t](#) offset, [l4_umword_t](#) size) [L4_NOTHROW](#)
- int [l4virtio_device_config_ds](#) ([l4_cap_idx_t](#) cap, [l4_cap_idx_t](#) config_ds, [l4_addr_t](#) *ds_offset) [L4_NOTHROW](#)
- int [l4virtio_device_notification_irq](#) ([l4_cap_idx_t](#) cap, unsigned index, [l4_cap_idx_t](#) irq) [L4_NOTHROW](#)

14.7.2.1 Detailed Description

[L4](#) specific VIRTIO Transport layer.

The [L4](#) specific VIRTIO Transport layer is based on [L4Re::Dataspace](#) as shared memory and [L4::lrq](#) for signaling. The VIRTIO configuration space is mostly based on a shared memory implementation too and accompanied by two IPC functions to synchronize the configuration between device and driver.

14.7.2.2 Typedef Documentation

14.7.2.2.1 l4virtio_config_queue_t

```
typedef struct l4virtio_config_queue_t l4virtio_config_queue_t
```

Queue configuration entry.

An array of such entries is available at the [l4virtio_config_hdr_t::queues_offset](#) in the config data space.

Consistency rules for the queue config are:

- A driver might read `num_max` at any time.
- A driver must write to `num`, `desc_addr`, `avail_addr`, and `used_addr` only when `ready` is zero (0). Values in these fields are validated and used by the device only after successfully setting `ready` to one (1), either by the IPC or by `L4VIRTIO_CMD_CFG_QUEUE`.
- The value of `device_notify_index` is valid only when `ready` is one.
- The driver might write to `device_notify_index` at any time, however the change is guaranteed to take effect after a successful `L4VIRTIO_CMD_CFG_QUEUE` or after a `config_queue` IPC. Note, the change might also have immediate effect, depending on the device implementation.

14.7.2.3 Enumeration Type Documentation

14.7.2.3.1 L4_virtio_cmd

```
enum L4_virtio_cmd
```

Virtio commands for device configuration.

Enumerator

L4VIRTIO_CMD_NONE	No command pending.
L4VIRTIO_CMD_SET_STATUS	Set the status register.
L4VIRTIO_CMD_CFG_QUEUE	Configure a queue.
L4VIRTIO_CMD_CFG_CHANGED	Device config changed.
L4VIRTIO_CMD_NOTIFY_QUEUE	Configure a queue.
L4VIRTIO_CMD_MASK	Mask to get command bits.

Definition at line 119 of file [virtio.h](#).

14.7.2.3.2 L4_virtio_irq_status

enum [L4_virtio_irq_status](#)

VIRTIO IRQ status codes (`l4virtio_config_hdr_t::irq_status`).

Note

`l4virtio_config_hdr_t::irq_status` is currently unused.

Enumerator

L4VIRTIO_IRQ_STATUS_VRING	VRING IRQ pending flag.
L4VIRTIO_IRQ_STATUS_CONFIG	CONFIG IRQ pending flag.

Definition at line 110 of file [virtio.h](#).

14.7.2.3.3 L4_virtio_opcodes

enum [L4_virtio_opcodes](#)

Opcodes to setup and configure a device.

Enumerator

L4VIRTIO_OP_SET_STATUS	Write device status register.
L4VIRTIO_OP_CONFIG_QUEUE	Configure queue.
L4VIRTIO_OP_REGISTER_DS	Register shared memory with device.
L4VIRTIO_OP_DEVICE_CONFIG	Get device config page.
L4VIRTIO_OP_GET_DEVICE_IRQ	Retrieve device notification IRQ.

Definition at line 52 of file [virtio.h](#).

14.7.2.3.4 L4virtio_device_ids

enum [L4virtio_device_ids](#)

Virtio device IDs as reported in the driver's config space.

Enumerator

L4VIRTIO_ID_NET	Virtual ethernet card.
L4VIRTIO_ID_BLOCK	General block device.
L4VIRTIO_ID_CONSOLE	Simple device for data IO via ports.
L4VIRTIO_ID_RNG	Entropy source.

L4VIRTIO_ID_BALLOON	Memory ballooning device.
L4VIRTIO_ID_RPMMSG	Device using rpmsg protocol.
L4VIRTIO_ID_SCSI	SCSI host device.
L4VIRTIO_ID_9P	Device using 9P transport protocol.
L4VIRTIO_ID_RPROC_SERIAL	Rproc serial device.
L4VIRTIO_ID_CAIF	Device using CAIF network protocol.
L4VIRTIO_ID_GPU	GPU.
L4VIRTIO_ID_INPUT	Input.
L4VIRTIO_ID_VSOCK	Vsock transport.
L4VIRTIO_ID_CRYPTIO	Crypto.
L4VIRTIO_ID_FS	FS.
L4VIRTIO_ID_SCMI	Scmi device.
L4VIRTIO_ID_I2C	I2C device.
L4VIRTIO_ID_GPIO	Gpio device.
L4VIRTIO_ID_SOCKET	Unofficial socket device.

Definition at line 62 of file [virtio.h](#).

14.7.2.3.5 L4virtio_device_status

enum [L4virtio_device_status](#)

Virtio device status bits.

Enumerator

L4VIRTIO_STATUS_ACKNOWLEDGE	Guest OS has found device.
L4VIRTIO_STATUS_DRIVER	Guest OS knows how to drive device.
L4VIRTIO_STATUS_DRIVER_OK	Driver is set up.
L4VIRTIO_STATUS_FEATURES_OK	Driver has acknowledged feature set.
L4VIRTIO_STATUS_DEVICE_NEEDS_RESET	Device detected fatal error.
L4VIRTIO_STATUS_FAILED	Driver detected fatal error.

Definition at line 87 of file [virtio.h](#).

14.7.2.3.6 L4virtio_feature_bits

enum [L4virtio_feature_bits](#)

L4virtio-specific feature bits.

Enumerator

L4VIRTIO_FEATURE_VERSION_1	Virtio protocol version 1 supported. Must be 1 for L4virtio .
L4VIRTIO_FEATURE_CMD_CONFIG	Status and queue config are set via cmd field instead of via IPC.

Definition at line 98 of file [virtio.h](#).

14.7.2.4 Function Documentation

14.7.2.4.1 l4virtio_config_queue()

```
int l4virtio_config_queue (
    l4_cap_idx_t cap,
    unsigned queue)
```

Parameters

<i>cap</i>	Capability to the VIRTIO host.
------------	--------------------------------

Trigger queue configuration of the given queue.

Usually all queues are configured when the status is written to running. However, in some cases queues shall be disabled or enabled dynamically, in this case this function triggers a reconfiguration from the shared memory register of the queue config.

Parameters

<i>queue</i>	Queue index for the queue to be configured.
--------------	---

Return values

0	on success.
-L4_EIO	The queue's status is invalid.
-L4_ERANGE	The queue index exceeds the number of queues.
-L4_EINVAL	Otherwise.

References [L4_CV](#), and [L4_NOTHROW](#).

14.7.2.4.2 l4virtio_config_queues()

```
L4_BEGIN_DECLS l4virtio_config_queue_t * l4virtio_config_queues (
    l4virtio_config_hdr_t const * cfg) [inline]
```

Get the pointer to the first queue config.

Parameters

<i>cfg</i>	Pointer to the config header.
------------	-------------------------------

Returns

pointer to queue config of queue 0.

Definition at line 252 of file [virtio.h](#).

References [l4virtio_config_hdr_t::queues_offset](#).

14.7.2.4.3 l4virtio_device_config()

```
void * l4virtio_device_config (
    l4virtio_config_hdr_t const * cfg) [inline]
```

Get the pointer to the device configuration.

Parameters

<i>cfg</i>	Pointer to the config header.
------------	-------------------------------

Returns

pointer to device configuration structure.

Definition at line 263 of file [virtio.h](#).

14.7.2.4.4 l4virtio_device_config_ds()

```
int l4virtio_device_config_ds (
    l4_cap_idx_t cap,
    l4_cap_idx_t config_ds,
    l4_addr_t * ds_offset)
```

Parameters

<i>cap</i>	Capability to the L4-VIRTIO host
------------	----------------------------------

Get the dataspace with the [L4virtio](#) configuration page.

Parameters

<i>config_ds</i>	Capability for receiving the dataspace capability for the shared L4-VIRTIO config data space.
<i>ds_offset</i>	Offset into the dataspace where the device configuration structure starts.

References [L4_CV](#), and [L4_NOTHROW](#).

14.7.2.4.5 l4virtio_device_notification_irq()

```
int l4virtio_device_notification_irq (  
    l4_cap_idx_t cap,  
    unsigned index,  
    l4_cap_idx_t irq)
```

Parameters

<i>cap</i>	Capability to the L4-VIRTIO host
------------	----------------------------------

Get the notification interrupt corresponding to the given index.

Parameters

	<i>index</i>	Index of the interrupt.
out	<i>irq</i>	Triggerable for the given index.

Return values

<i>L4_EOK</i>	Success.
<i>L4_ENOSYS</i>	IRQ notification not supported by device.
<i><0</i>	Other error.

An index is only guaranteed to return an IRQ object when the index is set in one of the device notify index fields. The device must return the same interrupt for a given index as long as the index is in use. If an index disappears as a result of a configuration change and then is reused later, the interrupt is not guaranteed to be the same.

Interrupts must always be rerequested after a device reset.

References [L4_END_DECLS](#), and [L4_NOTHROW](#).

14.7.2.4.6 l4virtio_register_ds()

```
int l4virtio_register_ds (
    l4_cap_idx_t cap,
    l4_cap_idx_t ds_cap,
    l4_uint64_t base,
    l4_umword_t offset,
    l4_umword_t size)
```

Parameters

<i>cap</i>	Capability to the VIRTIO host
------------	-------------------------------

Register a shared data space with VIRTIO host.

Parameters

<i>ds_cap</i>	Dataspace capability to register. The lower 8 bits determine the rights mask with which the guest's rights are masked during the registration of the dataspace at the VIRTIO host.
<i>base</i>	VIRTIO guest physical start address of shared memory region
<i>offset</i>	Offset within the data space that is attached to the given <i>base</i> in the guest physical memory.
<i>size</i>	Size of the memory region in the guest

Return values

<i>L4_EOK</i>	Operation successful.
<i>-L4_EINVAL</i>	The <code>ds_cap</code> capability is invalid, does not refer to a valid dataspace, is not a trusted dataspace if trusted dataspace validation is enabled, or <code>size</code> and <code>offset</code> specify an invalid region.
<i>-L4_ENOMEM</i>	The limit of dataspaces that can be registered has been reached or no capability slot could be allocated.
<i>-L4_ERANGE</i>	<code>offset</code> is larger than the size of the dataspace.
<i><0</i>	Any error returned by the dataspace when queried for information during setup or any error returned by the region manager from attaching the dataspace.

References [L4_CV](#), and [L4_NOTHROW](#).

14.7.2.4.7 l4virtio_set_status()

```
int l4virtio_set_status (
    l4_cap_idx_t cap,
    unsigned status)
```

Parameters

<i>cap</i>	Capability to the VIRTIO host
------------	-------------------------------

Write the VIRTIO status register.

Parameters

<i>status</i>	Status word to write to the VIRTIO status.
---------------	--

Return values

<i>0</i>	on success.
----------	-------------

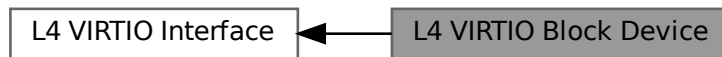
Note

All other registers are accessed via shared memory.

References [L4_CV](#), and [L4_NOTHROW](#).

14.7.3 L4 VIRTIO Block Device

Collaboration diagram for L4 VIRTIO Block Device:



Data Structures

- struct `l4virtio_block_header_t`
Header structure of a request for a block device.
- struct `l4virtio_block_discard_t`
Structure used for the write zeroes and discard commands.
- struct `l4virtio_block_config_t`
Device configuration for block devices.

Typedefs

- typedef struct `l4virtio_block_header_t` **`l4virtio_block_header_t`**
Header structure of a request for a block device.
- typedef struct `l4virtio_block_discard_t` **`l4virtio_block_discard_t`**
Structure used for the write zeroes and discard commands.
- typedef struct `l4virtio_block_config_t` **`l4virtio_block_config_t`**
Device configuration for block devices.

Enumerations

- enum `L4virtio_block_operations` {
`L4VIRTIO_BLOCK_T_IN` = 0 , `L4VIRTIO_BLOCK_T_OUT` = 1 , `L4VIRTIO_BLOCK_T_FLUSH` = 4 ,
`L4VIRTIO_BLOCK_T_GET_ID` = 8 ,
`L4VIRTIO_BLOCK_T_DISCARD` = 11 , `L4VIRTIO_BLOCK_T_WRITE_ZEROES` = 13 }
Kinds of operation over a block device.
- enum `L4virtio_block_status` { `L4VIRTIO_BLOCK_S_OK` = 0 , `L4VIRTIO_BLOCK_S_IOERR` = 1 ,
`L4VIRTIO_BLOCK_S_UNSUPP` = 2 }
Status of a finished block request.

14.7.3.1 Detailed Description

14.7.3.2 Enumeration Type Documentation

14.7.3.2.1 L4virtio_block_operations

enum `L4virtio_block_operations`

Kinds of operation over a block device.

Enumerator

L4VIRTIO_BLOCK_T_IN	Read from device.
L4VIRTIO_BLOCK_T_OUT	Write to device.
L4VIRTIO_BLOCK_T_FLUSH	Flush data to disk.
L4VIRTIO_BLOCK_T_GET_ID	Get device ID.
L4VIRTIO_BLOCK_T_DISCARD	Discard a range of sectors.
L4VIRTIO_BLOCK_T_WRITE_ZEROES	Write zeroes to a range of sectors.

Definition at line 19 of file [virtio_block.h](#).

14.7.3.2.2 L4virtio_block_status

enum [L4virtio_block_status](#)

Status of a finished block request.

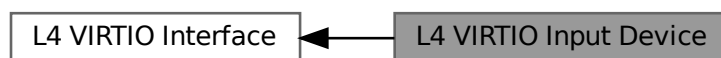
Enumerator

L4VIRTIO_BLOCK_S_OK	Request finished successfully.
L4VIRTIO_BLOCK_S_IOERR	IO error on device.
L4VIRTIO_BLOCK_S_UNSUPP	Operation is not supported.

Definition at line 32 of file [virtio_block.h](#).

14.7.4 L4 VIRTIO Input Device

Collaboration diagram for L4 VIRTIO Input Device:



Data Structures

- struct [l4virtio_input_absinfo_t](#)
Information about the absolute axis in the underlying evdev implementation.
- struct [l4virtio_input_devids_t](#)
Device ID information for the device.
- struct [l4virtio_input_config_t](#)
Device configuration for input devices.
- struct [l4virtio_input_event_t](#)
Single event in event or status queue.

Typedefs

- typedef struct [l4virtio_input_absinfo_t](#) **l4virtio_absinfo_t**
Information about the absolute axis in the underlying evdev implementation.
- typedef struct [l4virtio_input_devids_t](#) **l4virtio_input_devids_t**
Device ID information for the device.
- typedef struct [l4virtio_input_config_t](#) **l4virtio_input_config_t**
Device configuration for input devices.
- typedef struct [l4virtio_input_event_t](#) **l4virtio_input_event_t**
Single event in event or status queue.

Enumerations

- enum [L4virtio_input_config_select](#)
Device information selectors.

14.7.4.1 Detailed Description

14.7.5 L4 VIRTIO Network Device

Collaboration diagram for L4 VIRTIO Network Device:



Data Structures

- struct [l4virtio_net_header_t](#)
Header structure of a request for a network device.
- struct [l4virtio_net_config_t](#)
Device configuration for network devices.

Typedefs

- typedef struct [l4virtio_net_header_t](#) **l4virtio_net_header_t**
Header structure of a request for a network device.
- typedef struct [l4virtio_net_config_t](#) **l4virtio_net_config_t**
Device configuration for network devices.

Enumerations

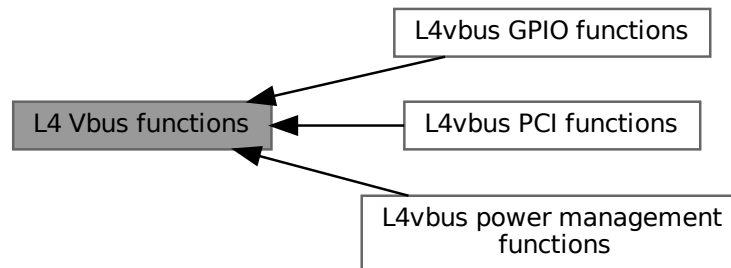
- enum [L4virtio_net_feature_bits](#)
Network device feature bits.

14.7.5.1 Detailed Description

14.8 L4 Vbus functions

C interface of the Vbus API.

Collaboration diagram for L4 Vbus functions:



Topics

- [L4vbus](#) GPIO functions ??
- [L4vbus](#) PCI functions ??
- [L4vbus](#) power management functions ??

Enumerations

- enum [L4vbus_dma_domain_assign_flags](#) { [L4VBUS_DMAD_UNBIND](#) = 0 , [L4VBUS_DMAD_BIND](#) = 1 , [L4VBUS_DMAD_L4RE_DMA_SPACE](#) = 0 , [L4VBUS_DMAD_KERNEL_DMA_SPACE](#) = 2 }
- Flags for [l4vbus_assign_dma_domain\(\)](#).*

Functions

- [L4_BEGIN_DECLS](#) int [l4vbus_get_device_by_hid](#) ([l4_cap_idx_t](#) vbus, [l4vbus_device_handle_t](#) parent, [l4vbus_device_handle_t](#) *child, char const *hid, int depth, [l4vbus_device_t](#) *devinfo)
Find a device by the hardware interface identifier (HID).
- int [l4vbus_get_next_device](#) ([l4_cap_idx_t](#) vbus, [l4vbus_device_handle_t](#) parent, [l4vbus_device_handle_t](#) *child, int depth, [l4vbus_device_t](#) *devinfo)
*Find next child following *child*.*
- int [l4vbus_get_device](#) ([l4_cap_idx_t](#) vbus, [l4vbus_device_handle_t](#) dev, [l4vbus_device_t](#) *devinfo)
Obtain detailed information about a Vbus device.
- int [l4vbus_get_resource](#) ([l4_cap_idx_t](#) vbus, [l4vbus_device_handle_t](#) dev, unsigned res_idx, [l4vbus_resource_t](#) *res)

Obtain the resource description of an individual device resource.

- int [l4vbus_is_compatible](#) ([l4_cap_idx_t](#) vbus, [l4vbus_device_handle_t](#) dev, char const *cid)

Check if the given device has a compatibility ID (CID) or HID that matches cid.

- int [l4vbus_get_hid](#) ([l4_cap_idx_t](#) vbus, [l4vbus_device_handle_t](#) dev, char *hid, unsigned long max_len)

Get the HID (hardware identifier) of a device.

- int [l4vbus_get_adr](#) ([l4_cap_idx_t](#) vbus, [l4vbus_device_handle_t](#) dev, [l4_uint32_t](#) *adr)

Get the bus-specific address of a device.

- int [l4vbus_request_ioport](#) ([l4_cap_idx_t](#) vbus, [l4vbus_resource_t](#) const *res)

Request an IO port resource.

- int [l4vbus_assign_dma_domain](#) ([l4_cap_idx_t](#) vbus, unsigned domain_id, unsigned flags, [l4_cap_idx_t](#) dma_space)

Bind or unbind a kernel DMA space or a [L4Re::Dma_space](#) to a DMA domain.

- int [l4vbus_release_ioport](#) ([l4_cap_idx_t](#) vbus, [l4vbus_resource_t](#) const *res)

Release a previously requested IO port resource.

- int [l4vbus_vicu_get_cap](#) ([l4_cap_idx_t](#) vbus, [l4vbus_device_handle_t](#) icu, [l4_cap_idx_t](#) cap)

Get capability of ICU.

14.8.1 Detailed Description

C interface of the Vbus API.

The virtual bus (Vbus) is a hierarchical (tree) structure of device nodes where each device has a set of resources attached to it. Each virtual bus provides an Icu ([Interrupt controller](#)) for interrupt handling.

The Vbus interface allows a client to find and query devices present on his virtual bus. After obtaining a device handle for a specific device the client can enumerate its resources.

Include File

```
#include <l4/vbus/vbus.h>
```

Refer to [L4vbus](#) for the C++ API.

14.8.2 Enumeration Type Documentation

14.8.2.1 L4vbus_dma_domain_assign_flags

```
enum L4vbus\_dma\_domain\_assign\_flags
```

Flags for [l4vbus_assign_dma_domain\(\)](#).

Enumerator

L4VBUS_DMAD_UNBIND	Unbind the given DMA space from the DMA domain.
L4VBUS_DMAD_BIND	Bind the given DMA space to the DMA domain.
L4VBUS_DMAD_L4RE_DMA_SPACE	The given DMA space is an L4Re::Dma_space .
L4VBUS_DMAD_KERNEL_DMA_SPACE	The given DMA space is a kernel DMA space (L4::Task).

Definition at line 174 of file [vbus.h](#).

14.8.3 Function Documentation

14.8.3.1 l4vbus_assign_dma_domain()

```
int l4vbus_assign_dma_domain (
    l4_cap_idx_t vbus,
    unsigned domain_id,
    unsigned flags,
    l4_cap_idx_t dma_space)
```

Bind or unbind a kernel [DMA space](#) or a [L4Re::Dma_space](#) to a DMA domain.

Parameters

<i>vbus</i>	Capability of the system bus
<i>domain_id</i>	DMA domain ID (resource address of DMA domain found on the vBUS). If the value is ~0U the DMA space of the whole vBUS is used.
<i>flags</i>	A combination of L4vbus_dma_domain_assign_flags .
<i>dma_space</i>	The DMA space capability to bind or unbind, this must either be an L4Re::Dma_space or a kernel DMA space (L4::Task created with L4_PROTO_DMA_SPACE) and the type must be reflected in the <i>flags</i> .

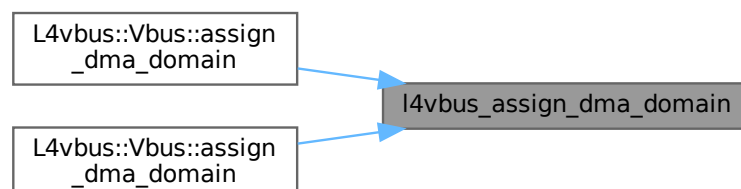
Return values

<i>0</i>	Operation completed successfully.
<i>-L4_ENOENT</i>	The vbus does not support a global DMA domain or no DMA domain could be found.
<i>-L4_EINVAL</i>	Invalid argument used.
<i>-L4_EBUSY</i>	DMA domain is already active, this means another DMA space is already assigned.

References [L4_CV](#).

Referenced by [L4vbus::Vbus::assign_dma_domain\(\)](#), and [L4vbus::Vbus::assign_dma_domain\(\)](#).

Here is the caller graph for this function:



14.8.3.2 l4vbus_get_adr()

```
int l4vbus_get_adr (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t dev,
    l4_uint32_t * adr)
```

Get the bus-specific address of a device.

Parameters

	<i>vbus</i>	Capability of the system bus
	<i>dev</i>	Handle of the device
out	<i>adr</i>	Address

Return values

<i>L4_EOK</i>	Success.
<i>-L4_ENOSYS</i>	Device has no valid address.

References [L4_CV](#).

14.8.3.3 l4vbus_get_device()

```
int l4vbus_get_device (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t dev,
    l4vbus_device_t * devinfo)
```

Obtain detailed information about a Vbus device.

Parameters

	<i>vbus</i>	Capability of the vbus to which the device is connected.
	<i>dev</i>	Device handle of the device from which to retrieve the details.
out	<i>devinfo</i>	Information structure which contains details about the device. The pointer might be NULL.

Return values

<i>0</i>	Success.
<i>-L4_ENODEV</i>	No device with the given device handle <i>dev</i> could be found.

References [L4_CV](#).

Referenced by [L4vbus::Device::device\(\)](#).

Here is the caller graph for this function:



14.8.3.4 l4vbus_get_device_by_hid()

```

L4_BEGIN_DECLS int l4vbus_get_device_by_hid (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t parent,
    l4vbus_device_handle_t * child,
    char const * hid,
    int depth,
    l4vbus_device_t * devinfo)
  
```

Find a device by the hardware interface identifier (HID).

Parameters

<i>vbus</i>	Capability of the system bus
<i>parent</i>	Handle to the parent to start the search

This function searches the vbus for a device with the given HID and returns a handle to the first matching device. The HID usually conforms to an ACPI HID or a Linux device tree compatible identifier.

It is possible to have multiple devices with the same HID on a vbus. In order to find all matching devices this function has to be called repeatedly with *child* pointing to the device found in the previous iteration. The iteration starts at *child* that might be any device node in the tree.

Parameters

in, out	<i>child</i>	Handle of the device from where in the device tree the search should start. To start searching from the beginning <i>child</i> must be initialized using the default (L4VBUS_NULL). If a matching device is found, its handle is returned through this parameter.
	<i>hid</i>	HID of the device
	<i>depth</i>	Maximum depth for the recursive lookup
out	<i>devinfo</i>	Device information structure (might be NULL)

Return values

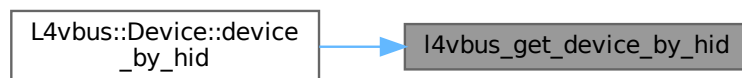
<i>>=0</i>	A device with the given HID was found.
---------------	--

<code>-L4_ENOENT</code>	No device with the given HID could be found on the vbus.
<code>-L4_EINVAL</code>	Invalid or no HID provided.
<code>-L4_ENODEV</code>	Function called on a non-existing device.

References [L4_CV](#).

Referenced by [L4vbus::Device::device_by_hid\(\)](#).

Here is the caller graph for this function:



14.8.3.5 l4vbus_get_hid()

```

int l4vbus_get_hid (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t dev,
    char * hid,
    unsigned long max_len)
  
```

Get the HID (hardware identifier) of a device.

Parameters

<i>vbus</i>	Capability of the system bus
<i>dev</i>	Handle of the device
<i>hid</i>	Pointer to a buffer for the HID string
<i>max_len</i>	The size of the buffer (<i>hid</i>)

Returns

the length of the HID string on success, else failure

References [L4_CV](#).

14.8.3.6 l4vbus_get_next_device()

```
int l4vbus_get_next_device (  
    l4_cap_idx_t vbus,  
    l4vbus_device_handle_t parent,  
    l4vbus_device_handle_t * child,  
    int depth,  
    l4vbus_device_t * devinfo)
```

Find next child following `child`.

Parameters

	<i>vbus</i>	Capability of the system bus
	<i>parent</i>	Handle to the parent device (use L4VBUS_ROOT_BUS for the system bus)
<i>in, out</i>	<i>child</i>	Handle of the device that precedes the device that shall be returned. To start from the beginning, <i>child</i> must be initialized with L4VBUS_NULL . If a device is found, its handle is returned through this parameter.
	<i>depth</i>	Depth to look for
<i>out</i>	<i>devinfo</i>	Device information (might be NULL)

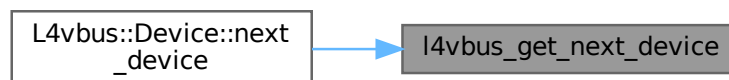
Returns

0 on success, else failure

References [L4_CV](#).

Referenced by [L4vbus::Device::next_device\(\)](#).

Here is the caller graph for this function:

**14.8.3.7 l4vbus_get_resource()**

```

int l4vbus_get_resource (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t dev,
    unsigned res_idx,
    l4vbus_resource_t * res)
  
```

Obtain the resource description of an individual device resource.

Parameters

	<i>vbus</i>	Capability of the vbus to which the device is connected.
	<i>dev</i>	Device handle of the device on the vbus. The device handle can be obtained by using the l4vbus_get_device_by_hid() and l4vbus_get_next_device() functions.
	<i>res_idx</i>	Index of the resource for which the resource description should be returned. The total number of resources for a device is available in the l4vbus_device_t structure that is returned by L4vbus::Device::device_by_hid() and L4vbus::Device::next_device() .
<i>out</i>	<i>res</i>	Descriptor of the resource.

This function returns the resource descriptor of an individual device resource selected by the *res_idx* parameter.

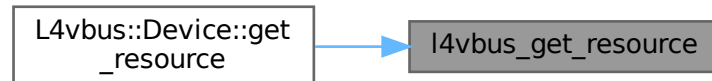
Return values

0	Success.
-L4_ENOENT	Invalid resource index <code>res_idx</code> .

References [L4_CV](#).

Referenced by [L4vbus::Device::get_resource\(\)](#).

Here is the caller graph for this function:



14.8.3.8 l4vbus_is_compatible()

```

int l4vbus_is_compatible (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t dev,
    char const * cid)
  
```

Check if the given device has a compatibility ID (CID) or HID that matches *cid*.

Parameters

<i>vbus</i>	Capability of the system bus
<i>dev</i>	device handle for which the CID shall be tested
<i>cid</i>	the compatibility ID to test

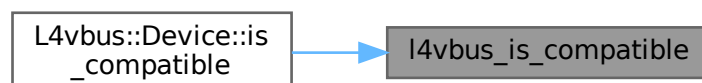
Returns

1 when the given ID (*cid*) matches this device, 0 when the given ID does not match, <0 on error.

References [L4_CV](#).

Referenced by [L4vbus::Device::is_compatible\(\)](#).

Here is the caller graph for this function:



14.8.3.9 l4vbus_release_ioport()

```
int l4vbus_release_ioport (
    l4_cap_idx_t vbus,
    l4vbus_resource_t const * res)
```

Release a previously requested IO port resource.

Parameters

	<i>vbus</i>	Capability of the system bus.
in	<i>res</i>	The IO port resource to be released from the bus.

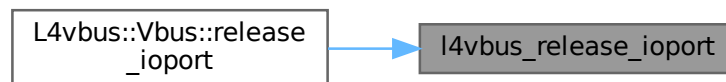
Returns

≥ 0 on success, < 0 on error.

References [L4_CV](#).

Referenced by [L4vbus::Vbus::release_ioport\(\)](#).

Here is the caller graph for this function:



14.8.3.10 l4vbus_request_ioport()

```
int l4vbus_request_ioport (
    l4_cap_idx_t vbus,
    l4vbus_resource_t const * res)
```

Request an IO port resource.

Parameters

	<i>vbus</i>	Capability of the system bus.
in	<i>res</i>	The IO port resource to be requested from the bus.

Return values

0	Success.
-L4_EINVAL	Resource is not an IO port resource.
-L4_ENOENT	No matching IO port resource found.

If any IO port resource is found that contains the requested IO port range the IO ports are obtained.

Referenced by [L4vbus::Vbus::request_ioport\(\)](#).

Here is the caller graph for this function:



14.8.3.11 l4vbus_vicu_get_cap()

```
int l4vbus_vicu_get_cap (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t icu,
    l4_cap_idx_t cap)
```

Get capability of ICU.

Parameters

<i>vbus</i>	Capability of the system bus.
<i>icu</i>	ICU device handle.
<i>cap</i>	Capability slot for the capability.

Returns

0 on success, else failure

References [L4_END_DECLS](#).

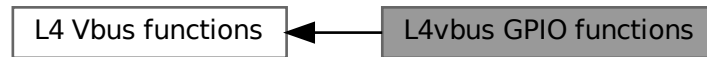
Referenced by [L4vbus::Icu::vicu\(\)](#).

Here is the caller graph for this function:



14.8.4 L4vbus GPIO functions

Collaboration diagram for L4vbus GPIO functions:



Enumerations

- enum `L4vbus_gpio_generic_func` { `L4VBUS_GPIO_SETUP_INPUT` = 0x100 , `L4VBUS_GPIO_SETUP_OUTPUT` = 0x200 , `L4VBUS_GPIO_SETUP_IRQ` = 0x300 }
Constants for generic GPIO functions.
- enum `L4vbus_gpio_pull_modes` { `L4VBUS_GPIO_PIN_PULL_NONE` = 0x100 , `L4VBUS_GPIO_PIN_PULL_UP` = 0x200 , `L4VBUS_GPIO_PIN_PULL_DOWN` = 0x300 }
Constants for generic GPIO pull up/down resistor configuration.

Functions

- int `l4vbus_gpio_setup` (`l4_cap_idx_t` vbus, `l4vbus_device_handle_t` handle, unsigned pin, unsigned mode, int value)
Configure the function of a GPIO pin.
- int `l4vbus_gpio_config_pull` (`l4_cap_idx_t` vbus, `l4vbus_device_handle_t` handle, unsigned pin, unsigned mode)
Generic function to set pull up/down mode.
- int `l4vbus_gpio_config_pad` (`l4_cap_idx_t` vbus, `l4vbus_device_handle_t` handle, unsigned pin, unsigned func, unsigned value)
Hardware specific configuration function.
- int `l4vbus_gpio_config_get` (`l4_cap_idx_t` vbus, `l4vbus_device_handle_t` handle, unsigned pin, unsigned func, unsigned *value)
Read hardware specific configuration.
- int `l4vbus_gpio_get` (`l4_cap_idx_t` vbus, `l4vbus_device_handle_t` handle, unsigned pin)
Read value of GPIO input pin.
- int `l4vbus_gpio_set` (`l4_cap_idx_t` vbus, `l4vbus_device_handle_t` handle, unsigned pin, int value)
Set GPIO output pin.
- int `l4vbus_gpio_multi_setup` (`l4_cap_idx_t` vbus, `l4vbus_device_handle_t` handle, unsigned offset, unsigned mask, unsigned mode, unsigned value)
Configure function of multiple GPIO pins at once.
- int `l4vbus_gpio_multi_config_pad` (`l4_cap_idx_t` vbus, `l4vbus_device_handle_t` handle, unsigned offset, unsigned mask, unsigned func, unsigned value)
Hardware specific configuration function for multiple GPIO pins.
- int `l4vbus_gpio_multi_get` (`l4_cap_idx_t` vbus, `l4vbus_device_handle_t` handle, unsigned offset, unsigned *data)
Read values of multiple GPIO pins at once.
- int `l4vbus_gpio_multi_set` (`l4_cap_idx_t` vbus, `l4vbus_device_handle_t` handle, unsigned offset, unsigned mask, unsigned data)
Set multiple GPIO output pins at once.
- int `l4vbus_gpio_to_irq` (`l4_cap_idx_t` vbus, `l4vbus_device_handle_t` handle, unsigned pin)
Create IRQ for GPIO pin.

14.8.4.1 Detailed Description

14.8.4.2 Enumeration Type Documentation

14.8.4.2.1 L4vbus_gpio_generic_func

enum [L4vbus_gpio_generic_func](#)

Constants for generic GPIO functions.

Enumerator

L4VBUS_GPIO_SETUP_INPUT	Set GPIO pin to input.
L4VBUS_GPIO_SETUP_OUTPUT	Set GPIO pin to output.
L4VBUS_GPIO_SETUP_IRQ	Set GPIO pin to IRQ.

Definition at line 24 of file [vbus_gpio.h](#).

14.8.4.2.2 L4vbus_gpio_pull_modes

enum [L4vbus_gpio_pull_modes](#)

Constants for generic GPIO pull up/down resistor configuration.

Enumerator

L4VBUS_GPIO_PIN_PULL_NONE	No pull up or pull down resistors.
L4VBUS_GPIO_PIN_PULL_UP	enable pull up resistor
L4VBUS_GPIO_PIN_PULL_DOWN	enable pull down resistor

Definition at line 34 of file [vbus_gpio.h](#).

14.8.4.3 Function Documentation

14.8.4.3.1 l4vbus_gpio_config_get()

```
int l4vbus_gpio_config_get (  
    l4\_cap\_idx\_t vbus,  
    l4vbus\_device\_handle\_t handle,  
    unsigned pin,  
    unsigned func,  
    unsigned * value)
```

Read hardware specific configuration.

Parameters

	<i>vbus</i>	V-BUS capability
	<i>handle</i>	Device handle for the GPIO chip
	<i>pin</i>	GPIO pin number
	<i>func</i>	Hardware specific configuration register to read from. Usually this is an offset to the GPIO chip's base address.
out	<i>value</i>	The configuration value.

Returns

0 if OK, error code otherwise

References [L4_CV](#).

Referenced by [L4vbus::Gpio_pin::config_get\(\)](#).

Here is the caller graph for this function:

**14.8.4.3.2 l4vbus_gpio_config_pad()**

```

int l4vbus_gpio_config_pad (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t handle,
    unsigned pin,
    unsigned func,
    unsigned value)
  
```

Hardware specific configuration function.

Parameters

<i>vbus</i>	V-BUS capability
<i>handle</i>	Device handle for the GPIO chip
<i>pin</i>	GPIO pin number
<i>func</i>	Hardware specific configuration register, usually offset to the GPIO chip's base address
<i>value</i>	Value which is written into the hardware specific configuration register for the specified pin

Returns

0 if OK, error code otherwise

References [L4_CV](#).

Referenced by [L4vbus::Gpio_pin::config_pad\(\)](#).

Here is the caller graph for this function:

**14.8.4.3.3 l4vbus_gpio_config_pull()**

```
int l4vbus_gpio_config_pull (  
    l4_cap_idx_t vbus,  
    l4vbus_device_handle_t handle,  
    unsigned pin,  
    unsigned mode)
```

Generic function to set pull up/down mode.

Parameters

<i>vbus</i>	V-BUS capability
<i>handle</i>	Device handle for the GPIO chip
<i>pin</i>	GPIO pin number
<i>mode</i>	mode for pull up/down resistors, see L4vbus_gpio_pull_modes

Returns

0 if OK, error code otherwise

References [L4_CV](#).

Referenced by [L4vbus::Gpio_pin::config_pull\(\)](#).

Here is the caller graph for this function:



14.8.4.3.4 l4vbus_gpio_get()

```
int l4vbus_gpio_get (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t handle,
    unsigned pin)
```

Read value of GPIO input pin.

Parameters

<i>vbus</i>	V-BUS capability
<i>handle</i>	Device handle for the GPIO chip
<i>pin</i>	GPIO pin number to read from

Returns

Value of GPIO pin (usually 0 or 1), negative error code otherwise.

References [L4_CV](#).

Referenced by [L4vbus::Gpio_pin::get\(\)](#).

Here is the caller graph for this function:



14.8.4.3.5 l4vbus_gpio_multi_config_pad()

```
int l4vbus_gpio_multi_config_pad (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t handle,
    unsigned offset,
    unsigned mask,
    unsigned func,
    unsigned value)
```

Hardware specific configuration function for multiple GPIO pins.

Parameters

<i>vbus</i>	V-BUS capability
<i>handle</i>	Device handle for the GPIO chip
<i>offset</i>	Pin corresponding to the LSB in <i>mask</i> . Note: allowed may be hardware specific.
<i>mask</i>	Mask of GPIO pins to configure. A bit set to 1 configures this pin. A maximum of 32 pins can be configured at once. The real number depends on the hardware and the driver implementation.
<i>func</i>	Hardware specific configuration register, usually offset to the GPIO chip's base address.
<i>value</i>	Value which is written into the hardware specific configuration register for the specified pins

Returns

0 if OK, error code otherwise

References [L4_CV](#).

Referenced by [L4vbus::Gpio_module::config_pad\(\)](#).

Here is the caller graph for this function:



14.8.4.3.6 l4vbus_gpio_multi_get()

```
int l4vbus_gpio_multi_get (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t handle,
    unsigned offset,
    unsigned * data)
```

Read values of multiple GPIO pins at once.

Parameters

	<i>vbus</i>	V-BUS capability
	<i>handle</i>	Device handle for the GPIO chip
	<i>offset</i>	Pin corresponding to the LSB in <i>data</i> . Note: allowed may be hardware specific.
out	<i>data</i>	Each bit returns the value (0 or 1) for the corresponding GPIO pin. The value of pins that are not accessible is undefined.

Returns

0 if OK, error code otherwise

References [L4_CV](#).

Referenced by [L4vbus::Gpio_module::get\(\)](#).

Here is the caller graph for this function:

**14.8.4.3.7 l4vbus_gpio_multi_set()**

```

int l4vbus_gpio_multi_set (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t handle,
    unsigned offset,
    unsigned mask,
    unsigned data)
  
```

Set multiple GPIO output pins at once.

Parameters

<i>vbus</i>	V-BUS capability
<i>handle</i>	Device handle for the GPIO chip
<i>offset</i>	Pin corresponding to the LSB in <i>data</i> . Note: allowed may be hardware specific.
<i>mask</i>	Mask of GPIO pins to set. A bit set to 1 selects this pin. A maximum of 32 pins can be set at once. The real number depends on the hardware and the driver implementation.
<i>data</i>	Each bit corresponds to the GPIO pin in <i>mask</i> . The value of each bit is written to the GPIO pin if its bit in <i>mask</i> is set.

Returns

0 if OK, error code otherwise

References [L4_CV](#).

Referenced by [L4vbus::Gpio_module::set\(\)](#).

Here is the caller graph for this function:



14.8.4.3.8 l4vbus_gpio_multi_setup()

```

int l4vbus_gpio_multi_setup (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t handle,
    unsigned offset,
    unsigned mask,
    unsigned mode,
    unsigned value)
  
```

Configure function of multiple GPIO pins at once.

Parameters

<i>vbus</i>	V-BUS capability
<i>handle</i>	Device handle for the GPIO chip
<i>offset</i>	Pin corresponding to the LSB in <i>mask</i> . Note: allowed may be hardware specific.
<i>mask</i>	Mask of GPIO pins to configure. A bit set to 1 configures this pin. A maximum of 32 pins can be configured at once. The real number depends on the hardware and the driver implementation.
<i>mode</i>	GPIO function, see L4vbus_gpio_generic_func for generic functions. Hardware specific functions must be provided in the lower 8 bits.
<i>value</i>	Optional value to set the GPIO pins to if they are configured as output pins

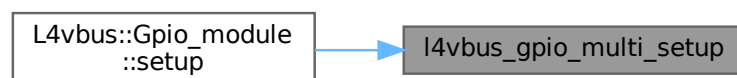
Returns

0 if OK, error code otherwise

References [L4_CV](#).

Referenced by [L4vbus::Gpio_module::setup\(\)](#).

Here is the caller graph for this function:



14.8.4.3.9 l4vbus_gpio_set()

```
int l4vbus_gpio_set (  
    l4_cap_idx_t vbus,  
    l4vbus_device_handle_t handle,  
    unsigned pin,  
    int value)
```

Set GPIO output pin.

Parameters

<i>vbus</i>	V-BUS capability
<i>handle</i>	Device handle for the GPIO chip
<i>pin</i>	GPIO pin number to write to
<i>value</i>	Value to write to the GPIO pin (usually 0 or 1)

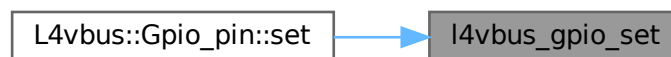
Returns

0 if OK, error code otherwise

References [L4_CV](#).

Referenced by [L4vbus::Gpio_pin::set\(\)](#).

Here is the caller graph for this function:



14.8.4.3.10 l4vbus_gpio_setup()

```
int l4vbus_gpio_setup (  
    l4_cap_idx_t vbus,  
    l4vbus_device_handle_t handle,  
    unsigned pin,  
    unsigned mode,  
    int value)
```

Configure the function of a GPIO pin.

Parameters

<i>vbus</i>	V-BUS capability
<i>handle</i>	Device handle for the GPIO chip
<i>pin</i>	GPIO pin number
<i>mode</i>	GPIO function, see L4vbus_gpio_generic_func for generic functions. Hardware specific functions must be provided in the lower 8 bits.
<i>value</i>	Optional value to set the GPIO pin to if it is configured as an output pin

Returns

0 if OK, error code otherwise

References [L4_CV](#).

Referenced by [L4vbus::Gpio_pin::setup\(\)](#).

Here is the caller graph for this function:



14.8.4.3.11 l4vbus_gpio_to_irq()

```

int l4vbus_gpio_to_irq (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t handle,
    unsigned pin)

```

Create IRQ for GPIO pin.

Parameters

<i>vbus</i>	V-BUS capability
<i>handle</i>	Device handle for the GPIO chip
<i>pin</i>	GPIO pin to create an IRQ for.

Returns

IRQ number if OK, negative error code otherwise

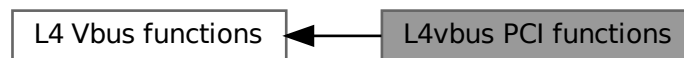
References [L4_END_DECLS](#).

Referenced by [L4vbus::Gpio_pin::to_irq\(\)](#).

Here is the caller graph for this function:

**14.8.5 L4vbus PCI functions**

Collaboration diagram for L4vbus PCI functions:

**Functions**

- [L4_BEGIN_DECLS](#) `int l4vbus_pci_cfg_read (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, l4_uint32_t bus, l4_uint32_t devfn, l4_uint32_t reg, l4_uint32_t *value, l4_uint32_t width)`
Read from the vPCI configuration space using the PCI root bridge.
- `int l4vbus_pci_cfg_write (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, l4_uint32_t bus, l4_uint32_t devfn, l4_uint32_t reg, l4_uint32_t value, l4_uint32_t width)`
Write to the vPCI configuration space using the PCI root bridge.
- `int l4vbus_pci_irq_enable (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, l4_uint32_t bus, l4_uint32_t devfn, int pin, unsigned char *trigger, unsigned char *polarity)`
Enable PCI interrupt for a specific device using the PCI root bridge.
- `int l4vbus_pcidv_cfg_read (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, l4_uint32_t reg, l4_uint32_t *value, l4_uint32_t width)`
Read from the device's vPCI configuration space.
- `int l4vbus_pcidv_cfg_write (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, l4_uint32_t reg, l4_uint32_t value, l4_uint32_t width)`
Write to the device's vPCI configuration space.
- `int l4vbus_pcidv_irq_enable (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, unsigned char *trigger, unsigned char *polarity)`
Enable the device's PCI interrupt.

14.8.5.1 Detailed Description

14.8.5.2 Function Documentation

14.8.5.2.1 l4vbus_pci_cfg_read()

```
L4_BEGIN_DECLS int l4vbus_pci_cfg_read (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t handle,
    l4_uint32_t bus,
    l4_uint32_t devfn,
    l4_uint32_t reg,
    l4_uint32_t * value,
    l4_uint32_t width)
```

Read from the vPCI configuration space using the PCI root bridge.

Parameters

	<i>vbus</i>	Capability of the system bus
	<i>handle</i>	Device handle of the PCI root bridge
	<i>bus</i>	Bus number
	<i>devfn</i>	Device id (upper 16bit) and function (lower 16bit)
	<i>reg</i>	Register in configuration space to read
out	<i>value</i>	Value that has been read
	<i>width</i>	Width to read in bits (e.g. 8, 16, 32)

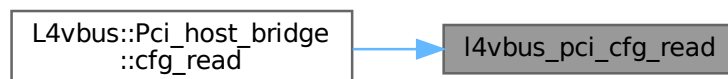
Returns

0 on success, else failure

References [L4_CV](#).

Referenced by [L4vbus::Pci_host_bridge::cfg_read\(\)](#).

Here is the caller graph for this function:



14.8.5.2.2 l4vbus_pci_cfg_write()

```
int l4vbus_pci_cfg_write (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t handle,
    l4_uint32_t bus,
    l4_uint32_t devfn,
    l4_uint32_t reg,
    l4_uint32_t value,
    l4_uint32_t width)
```

Write to the vPCI configuration space using the PCI root bridge.

Parameters

<i>vbus</i>	Capability of the system bus
<i>handle</i>	Device handle of the PCI root bridge
<i>bus</i>	Bus number
<i>devfn</i>	Device id (upper 16bit) and function (lower 16bit)
<i>reg</i>	Register in configuration space to write
<i>value</i>	Value to write
<i>width</i>	Width to write in bits (e.g. 8, 16, 32)

Returns

0 on success, else failure

References [L4_CV](#).

Referenced by [L4vbus::Pci_host_bridge::cfg_write\(\)](#).

Here is the caller graph for this function:



14.8.5.2.3 l4vbus_pci_irq_enable()

```
int l4vbus_pci_irq_enable (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t handle,
    l4_uint32_t bus,
```

```
l4_uint32_t devfn,  
int pin,  
unsigned char * trigger,  
unsigned char * polarity)
```

Enable PCI interrupt for a specific device using the PCI root bridge.

Parameters

	<i>vbus</i>	Capability of the system bus
	<i>handle</i>	Device handle of the PCI root bridge
	<i>bus</i>	Bus number
	<i>devfn</i>	Device id (upper 16bit) and function (lower 16bit)
	<i>pin</i>	Interrupt pin (normally as reported in configuration register INTR)
out	<i>trigger</i>	False if interrupt is level-triggered
out	<i>polarity</i>	True if interrupt is of low polarity

Returns

On success: Interrupt line to be used, else failure

References [L4_CV](#).

Referenced by [L4vbus::Pci_host_bridge::irq_enable\(\)](#).

Here is the caller graph for this function:



14.8.5.2.4 l4vbus_pciddev_cfg_read()

```

int l4vbus_pciddev_cfg_read (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t handle,
    l4_uint32_t reg,
    l4_uint32_t * value,
    l4_uint32_t width)
  
```

Read from the device's vPCI configuration space.

Parameters

	<i>vbus</i>	Capability of the system bus
	<i>handle</i>	Device handle of the PCI device
	<i>reg</i>	Register in configuration space to read
out	<i>value</i>	Value that has been read
	<i>width</i>	Width to read in bits (e.g. 8, 16, 32)

Returns

0 on success, else failure

References [L4_CV](#).

Referenced by [L4vbus::Pci_dev::cfg_read\(\)](#).

Here is the caller graph for this function:

**14.8.5.2.5 l4vbus_pciddev_cfg_write()**

```

int l4vbus_pciddev_cfg_write (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t handle,
    l4_uint32_t reg,
    l4_uint32_t value,
    l4_uint32_t width)
  
```

Write to the device's vPCI configuration space.

Parameters

<i>vbus</i>	Capability of the system bus
<i>handle</i>	Device handle of the PCI device
<i>reg</i>	Register in configuration space to write
<i>value</i>	Value to write
<i>width</i>	Width to write in bits (e.g. 8, 16, 32)

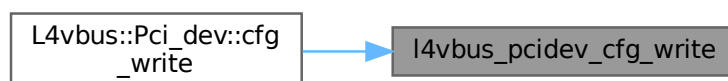
Returns

0 on success, else failure

References [L4_CV](#).

Referenced by [L4vbus::Pci_dev::cfg_write\(\)](#).

Here is the caller graph for this function:



14.8.5.2.6 l4vbus_pcidev_irq_enable()

```
int l4vbus_pcidev_irq_enable (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t handle,
    unsigned char * trigger,
    unsigned char * polarity)
```

Enable the device's PCI interrupt.

Parameters

	<i>vbus</i>	Capability of the system bus
	<i>handle</i>	Device handle of the PCI device
out	<i>trigger</i>	False if interrupt is level-triggered
out	<i>polarity</i>	True if interrupt is of low polarity

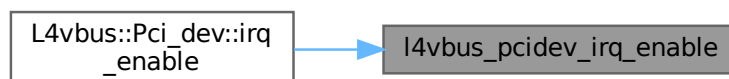
Returns

On success: Interrupt line to be used, else failure

References [L4_END_DECLS](#).

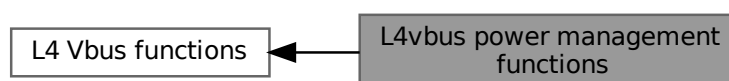
Referenced by [L4vbus::Pci_dev::irq_enable\(\)](#).

Here is the caller graph for this function:



14.8.6 L4vbus power management functions

Collaboration diagram for L4vbus power management functions:



Functions

- [L4_BEGIN_DECLS](#) `int l4vbus_pm_suspend (l4_cap_idx_t vbus, l4vbus_device_handle_t handle)`
Suspend the device.
- `int l4vbus_pm_resume (l4_cap_idx_t vbus, l4vbus_device_handle_t handle)`
Resume the device.

14.8.6.1 Detailed Description

14.8.6.2 Function Documentation

14.8.6.2.1 l4vbus_pm_resume()

```
int l4vbus_pm_resume (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t handle)
```

Resume the device.

Parameters

<i>vbus</i>	V-BUS capability
<i>handle</i>	Handle for the device to be resumed Switches the device from low-power mode to normal operation and restores the saved state.

Return values

0	Success.
---	----------

References [L4_END_DECLS](#).

Referenced by [L4vbus::Pm< DEC >::pm_resume\(\)](#).

Here is the caller graph for this function:



14.8.6.2.2 l4vbus_pm_suspend()

```
L4_BEGIN_DECLS int l4vbus_pm_suspend (
    l4_cap_idx_t vbus,
    l4vbus_device_handle_t handle)
```

Suspend the device.

Parameters

<i>vbus</i>	V-BUS capability
<i>handle</i>	Handle for the device to be suspended Saves the state of the device and puts it into a low-power mode.

Return values

0	Success.
---	----------

References [L4_CV](#).

Referenced by [L4vbus::Pm< DEC >::pm_suspend\(\)](#).

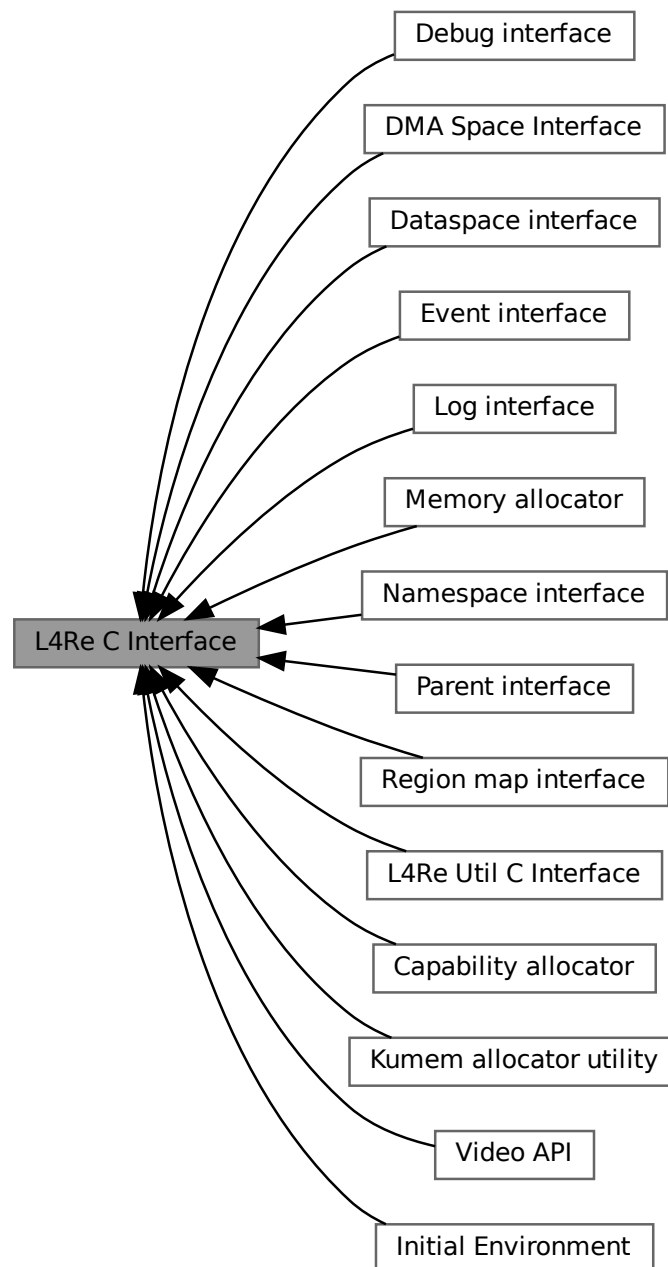
Here is the caller graph for this function:



14.9 L4Re C Interface

Documentation for the [L4Re C Interface](#).

Collaboration diagram for L4Re C Interface:



Topics

- [L4Re Util C Interface](#) ??
Documentation of the [L4](#) Runtime Environment utility functionality in C.
- [Dataspace interface](#) ??

<i>Dataspace C interface.</i>	
•	
Debug interface	??
•	
DMA Space Interface	??
<i>DMA Space C interface.</i>	
•	
Event interface	??
<i>Event C interface.</i>	
•	
Log interface	??
<i>Log C interface.</i>	
•	
Memory allocator	??
<i>Memory allocator C interface.</i>	
•	
Namespace interface	??
<i>Namespace C interface.</i>	
•	
Parent interface	??
•	
Region map interface	??
<i>Region map C interface.</i>	
•	
Capability allocator	??
<i>Capability allocator C interface.</i>	
•	
Kumem allocator utility	??
<i>Kumem allocator utility C interface.</i>	
•	
Video API	??
•	
Initial Environment	??
<i>C interface of the initial environment that is provided to an L4 task.</i>	

Files

- file [inhibitor.h](#)
Inhibitor C interface.

14.9.1 Detailed Description

Documentation for the [L4Re C](#) Interface.

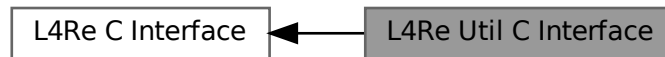
The interface functions closely align with the C++ functions and add no further functionalities.

For new programs it is advised to use the C++ interface.

14.9.2 L4Re Util C Interface

Documentation of the [L4](#) Runtime Environment utility functionality in C.

Collaboration diagram for L4Re Util C Interface:



Documentation of the [L4](#) Runtime Environment utility functionality in C.

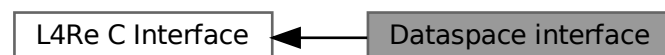
The interface functions closely align with the C++ functions and add no further functionalities.

For new programs it is advised to use the C++ interface.

14.9.3 Dataspace interface

Dataspace C interface.

Collaboration diagram for Dataspace interface:



Data Structures

- struct [l4re_ds_stats_t](#)
Information about the data space.

Enumerations

- enum [l4re_ds_map_flags](#) { }
Flags to specify the memory mapping type of a request.

Functions

- `long l4re_ds_clear (l4re_ds_t ds, l4re_ds_offset_t offset, l4re_ds_size_t size) L4_NOTHROW`
Clear parts of a dataspace.
- `long l4re_ds_allocate (l4re_ds_t ds, l4re_ds_offset_t offset, l4re_ds_size_t size) L4_NOTHROW`
Allocate a range in the dataspace.
- `int l4re_ds_copy_in (l4re_ds_t ds, l4re_ds_offset_t dst_offs, l4re_ds_t src, l4re_ds_offset_t src_offs, l4re_ds_size_t size) L4_NOTHROW`
Copy contents from another dataspace.
- `l4re_ds_size_t l4re_ds_size (l4re_ds_t ds) L4_NOTHROW`
Get size of a dataspace.
- `l4re_ds_flags_t l4re_ds_flags (l4re_ds_t ds) L4_NOTHROW`
Get flags of the dataspace.
- `int l4re_ds_info (l4re_ds_t ds, l4re_ds_stats_t *stats) L4_NOTHROW`
Get information on the dataspace.
- `int l4re_ds_map_info (l4re_ds_t ds, l4_addr_t *start_addr, l4_addr_t *end_addr) L4_NOTHROW`
Get mapping range of dataspace.

Variables

- `L4_BEGIN_DECLS typedef l4_cap_idx_t l4re_ds_t`
Dataspace type.

14.9.3.1 Detailed Description

Dataspace C interface.

14.9.3.2 Enumeration Type Documentation

14.9.3.2.1 l4re_ds_map_flags

```
enum l4re_ds_map_flags
```

Flags to specify the memory mapping type of a request.

Enumerator

L4RE_DS_F_NORMAL	request normal memory mapping
L4RE_DS_F_CACHEABLE	request normal memory mapping
L4RE_DS_F_BUFFERABLE	request bufferable (write buffered) mappings
L4RE_DS_F_UNCACHEABLE	request uncacheable memory mappings
L4RE_DS_F_CACHING_MASK	mask for caching flags
L4RE_DS_F_CACHING_SHIFT	shift value for caching flags

Definition at line 48 of file [dataspace.h](#).

14.9.3.3 Function Documentation

14.9.3.3.1 `l4re_ds_allocate()`

```
long l4re_ds_allocate (
    l4re_ds_t ds,
    l4re_ds_offset_t offset,
    l4re_ds_size_t size)
```

Allocate a range in the dataspace.

Parameters

<i>ds</i>	Dataspace capability.
<i>offset</i>	Offset in the dataspace, in bytes.
<i>size</i>	Size of the range, in bytes.

Return values

<i>L4_EOK</i>	Success
<i>-L4_ERANGE</i>	Given range is outside the dataspace. (A dataspace provider may also silently ignore areas outside the dataspace.)
<i>-L4_ENOMEM</i>	Not enough memory available.
<i><0</i>	IPC errors

On success, at least the given range is guaranteed to be allocated. The dataspace manager may also allocate more memory due to page granularity.

The memory is allocated with the same rights as the dataspace capability.

References [L4_CV](#), [L4_NOTHROW](#), and [l4re_ds_t](#).

14.9.3.3.2 `l4re_ds_clear()`

```
long l4re_ds_clear (
    l4re_ds_t ds,
    l4re_ds_offset_t offset,
    l4re_ds_size_t size)
```

Clear parts of a dataspace.

Parameters

<i>ds</i>	Dataspace capability.
<i>offset</i>	Offset within dataspace (in bytes).
<i>size</i>	Size of region to clear (in bytes).

Return values

≥ 0	Success.
<code>-L4_ERANGE</code>	Given range is outside the dataspace. (A dataspace provider may also silently ignore areas outside the dataspace.)
<code>-L4_EACCESS</code>	No L4_CAP_FPAGE_W right on dataspace capability.
< 0	IPC errors

Zeroes out the memory. Depending on the type of memory the memory could also be deallocated and replaced by a shared zero-page.

References [L4_CV](#), [L4_NOTHROW](#), and [l4re_ds_t](#).

14.9.3.3.3 l4re_ds_copy_in()

```
int l4re_ds_copy_in (
    l4re_ds_t ds,
    l4re_ds_offset_t dst_offs,
    l4re_ds_t src,
    l4re_ds_offset_t src_offs,
    l4re_ds_size_t size)
```

Copy contents from another dataspace.

Parameters

<i>ds</i>	Destination dataspace.
<i>dst_offs</i>	Offset in destination dataspace.
<i>src</i>	Source dataspace to copy from.
<i>src_offs</i>	Offset in the source dataspace.
<i>size</i>	Size to copy (in bytes).

Return values

<code>L4_EOK</code>	Success
<code>-L4_EACCESS</code>	No L4_CAP_FPAGE_W right on the destination dataspace.
<code>-L4_EINVAL</code>	Invalid parameter supplied.
< 0	IPC errors

The copy operation may use copy-on-write mechanisms. The operation may also fail if both dataspaces are not from the same dataspace manager or the dataspace managers do not cooperate.

References [L4_CV](#), [L4_NOTHROW](#), and [l4re_ds_t](#).

14.9.3.3.4 l4re_ds_flags()

```
l4re_ds_flags_t l4re_ds_flags (
    l4re_ds_t ds)
```

Get flags of the dataspace.

Parameters

<i>ds</i>	Dataspace capability.
-----------	-----------------------

Return values

≥ 0	Flags of the dataspace
< 0	IPC errors

See also

[L4Re::Dataspace::F::Flags](#)

References [L4_CV](#), [L4_NOTHROW](#), and [l4re_ds_t](#).

14.9.3.3.5 l4re_ds_info()

```
int l4re_ds_info (
    l4re_ds_t ds,
    l4re_ds_stats_t * stats)
```

Get information on the dataspace.

Parameters

	<i>ds</i>	Dataspace capability.
out	<i>stats</i>	Dataspace information

Return values

0	Success
< 0	IPC errors

References [L4_CV](#), [L4_NOTHROW](#), and [l4re_ds_t](#).

14.9.3.3.6 l4re_ds_map_info()

```
int l4re_ds_map_info (
    l4re_ds_t ds,
    l4_addr_t * start_addr,
    l4_addr_t * end_addr)
```

Get mapping range of dataspace.

Parameters

<i>ds</i>	Dataspace capability. In case of a MMU-less system, the dataspace must be mapped at the correct address in the task because virtual and physical address must match. This method returns the start and end address of the physically contiguous buffer backing the dataspace.
-----------	---

On MMU-enabled system any page aligned address is permissible. On such systems the method is just a stub.

Parameters

out	<i>start_addr</i>	Start address of dataspace.
out	<i>end_addr</i>	End address (inclusive) of dataspace.

Return values

>0	Start/end address have been set and need to be obeyed.
0	No constraint of mapping address.
$-L4_EPMR$	Cannot infer mapping address. Dataspace not mappable.
<0	IPC errors.

References [L4_END_DECLS](#), [L4_NOTHROW](#), and [l4re_ds_t](#).

14.9.3.3.7 l4re_ds_size()

```
l4re_ds_size_t l4re_ds_size (
    l4re_ds_t ds)
```

Get size of a dataspace.

Parameters

<i>ds</i>	Dataspace capability.
-----------	-----------------------

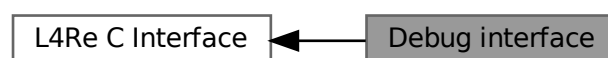
Returns

Size of the dataspace in bytes.

References [L4_CV](#), [L4_NOTHROW](#), and [l4re_ds_t](#).

14.9.4 Debug interface

Collaboration diagram for Debug interface:



Functions

- [L4_BEGIN_DECLS](#) long [l4re_debug_obj_debug](#) ([l4_cap_idx_t](#) *srv*, unsigned long *function*) [L4_NOTHROW](#)
Call debug function of [L4Re](#) service.

14.9.4.1 Detailed Description

14.9.4.2 Function Documentation

14.9.4.2.1 [l4re_debug_obj_debug\(\)](#)

```
L4\_BEGIN\_DECLS long l4re\_debug\_obj\_debug (  
    l4\_cap\_idx\_t srv,  
    unsigned long function)
```

Call debug function of [L4Re](#) service.

Parameters

<i>srv</i>	Object to call.
<i>function</i>	Function to call.

See also

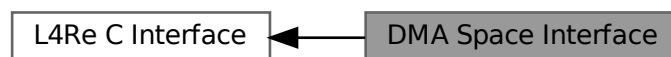
[L4Re::Debug_obj::debug](#)

References [L4_END_DECLS](#), and [L4_NOTHROW](#).

14.9.5 DMA Space Interface

DMA Space C interface.

Collaboration diagram for DMA Space Interface:



Typedefs

- typedef [l4_cap_idx_t](#) [l4re_dma_space_t](#)
DMA space capability type.

Functions

- long [l4re_dma_space_map](#) ([l4re_dma_space_t](#) dma, [l4re_ds_t](#) src, [l4re_ds_offset_t](#) offset, [l4_size_t](#) *size, unsigned long attrs, enum [l4re_dma_space_direction](#) dir, [l4re_dma_space_dma_addr_t](#) *dma_addr) [L4_NOTHROW](#)
Map the given part of this data space into the DMA address space.
- long [l4re_dma_space_unmap](#) ([l4re_dma_space_t](#) dma, [l4re_dma_space_dma_addr_t](#) dma_addr, [l4_size_t](#) size, unsigned long attrs, enum [l4re_dma_space_direction](#) dir) [L4_NOTHROW](#)
Unmap the given part of this data space from the DMA address space.
- long [l4re_dma_space_associate](#) ([l4re_dma_space_t](#) dma, [l4_cap_idx_t](#) dma_task, unsigned long attr) [L4_NOTHROW](#)
Associate a (kernel) DMA space for a device to this Dma_space.
- long [l4re_dma_space_disassociate](#) ([l4re_dma_space_t](#) dma)
Disassociate the (kernel) DMA space from this Dma_space.

14.9.5.1 Detailed Description

DMA Space C interface.

14.9.5.2 Typedef Documentation

14.9.5.2.1 l4re_dma_space_t

```
typedef l4_cap_idx_t l4re_dma_space_t
```

DMA space capability type.

Managed DMA Address Space.

A managed Dma_space represents the [L4Re](#) abstraction of an DMA address space of one or several devices. Devices are assigned to a managed Dma_space by binding the Dma_space to the respective DMA domain (see [L4vbus::Vbus::assign_dma_domain\(\)](#)), which might link the Dma_space with a kernel [DMA space](#). Note that several DMA domains can be bound to the same Dma_space. Whenever a device needs direct access to parts of an [L4Re::Dataspace](#), that part of the data space must be mapped to the managed Dma_space that is assigned to that device. Binding to DMA domains must happen before mapping. After the DMA accesses to the memory are finished the memory must be unmapped from the device's DMA address space.

Mapping to a managed DMA address space, using [map\(\)](#), makes the given parts of the data space visible to the associated device at the returned DMA address. As long as the memory is mapped into a DMA space it is 'pinned' and cannot be subject to dynamic memory management such as swapping. Additionally, [map\(\)](#) is responsible for the necessary syncing operations before the DMA.

[unmap\(\)](#) is the reverse operation to [map\(\)](#) and unmaps the given data-space part for the DMA address space. [unmap\(\)](#) is responsible for the necessary sync operations after the DMA.

Definition at line 49 of file [dma_space.h](#).

14.9.5.3 Function Documentation

14.9.5.3.1 l4re_dma_space_associate()

```
long l4re_dma_space_associate (
    l4re_dma_space_t dma,
    l4_cap_idx_t dma_task,
    unsigned long attr)
```

Associate a (kernel) [DMA space](#) for a device to this Dma_space.

Parameters

	<i>dma</i>	DMA space capability
in	<i>dma_task</i>	The (kernel) DMA space used for the device that shall be associated with this DMA space. In case no IOMMU is present or configured, the <i>dma_task</i> might be an invalid capability when L4Re::Dma_space::Phys_space is set in <i>attr</i> , in this case the CPUs physical memory is used as DMA address space.
in	<i>attr</i>	Attributes for this DMA space. See L4Re::Dma_space::Space_attr .

Return values

<i>L4_EOK</i>	Operation successful.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i>-L4_EINVAL</i>	
<i>-L4_ENOENT</i>	

Precondition

The invoked *Dma_space* capability must have the permission [L4_CAP_FPAGE_W](#).

References [L4_CV](#), and [L4_NOTHROW](#).

14.9.5.3.2 *l4re_dma_space_disassociate()*

```
long l4re_dma_space_disassociate (
    l4re_dma_space_t dma)
```

Disassociate the (kernel) [DMA space](#) from this *Dma_space*.

Parameters

<i>dma</i>	DMA space capability
------------	----------------------

Return values

<i>L4_EOK</i>	Operation successful.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i>-L4_ENOENT</i>	

Precondition

The invoked *Dma_space* capability must have the permission [L4_CAP_FPAGE_W](#).

References [L4_END_DECLS](#).

14.9.5.3.3 l4re_dma_space_map()

```
long l4re_dma_space_map (
    l4re_dma_space_t dma,
    l4re_ds_t src,
    l4re_ds_offset_t offset,
    l4_size_t * size,
    unsigned long attrs,
    enum l4re_dma_space_direction dir,
    l4re_dma_space_dma_addr_t * dma_addr)
```

Map the given part of this data space into the DMA address space.

Parameters

	<i>dma</i>	DMA space capability
in	<i>src</i>	Source data space (that describes the memory). Caller needs write right to the data space.
in	<i>offset</i>	The offset (bytes) within <i>src</i> .
in, out	<i>size</i>	The size (bytes) of the region to be mapped for DMA, after successful mapping the size returned is the size mapped for DMA as a single block. This size might be smaller than the original input size, in this case the caller might call <code>map()</code> again with a new offset and the remaining size.
in	<i>attrs</i>	The attributes used for this DMA mapping (a combination of <code>Dma_space::Attribute</code> values).
in	<i>dir</i>	The direction of the DMA transfer issued with this mapping. The same value must later be passed to <code>unmap()</code> .
out	<i>dma_addr</i>	The DMA address to use for DMA with the associated device.

Return values

<i>L4_EOK</i>	Operation successful.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i>-L4_EINVAL</i>	The capability <i>src</i> is invalid or does not refer to a valid dataspace.
<i>-L4_EEXIST</i>	The specified region overlaps an existing mapping.
<i>-L4_ENOMEM</i>	Not enough memory to allocate internal datastructures.
<i>-L4_ERANGE</i>	<i>offset</i> is larger than the size of the dataspace.

Precondition

The capability *src* must have the permission [L4_CAP_FPAGE_W](#).

Note

`associate()` must be called prior to mapping memory. Usually this is done implicitly when binding the managed `Dma_space` to a DMA domain (see [L4vbus::Vbus::assign_dma_domain\(\)](#)).

References [L4_CV](#), [L4_NOTHROW](#), and [l4re_ds_t](#).

14.9.5.3.4 l4re_dma_space_unmap()

```
long l4re_dma_space_unmap (
    l4re_dma_space_t dma,
    l4re_dma_space_dma_addr_t dma_addr,
    l4_size_t size,
    unsigned long attrs,
    enum l4re_dma_space_direction dir)
```

Unmap the given part of this data space from the DMA address space.

Parameters

<i>dma</i>	DMA space capability
<i>dma_addr</i>	The DMA address (returned by <code>Dma_space::map()</code>).
<i>size</i>	The size (bytes) of the memory region to unmap.
<i>attrs</i>	The attributes for the unmap (currently none).
<i>dir</i>	The direction of the finished DMA operation.

Returns

0 in the case of success, a negative error code otherwise.

References [L4_CV](#), and [L4_NOTHROW](#).

14.9.6 Event interface

Event C interface.

Collaboration diagram for Event interface:



Functions

- long [l4re_event_get_buffer](#) (const [l4_cap_idx_t](#) server, const [l4re_ds_t](#) ds) [L4_NOTHROW](#)
Get an event signal buffer.
- long [l4re_event_get_num_streams](#) (const [l4_cap_idx_t](#) server) [L4_NOTHROW](#)
Get number of streams.
- long [l4re_event_get_stream_info](#) (const [l4_cap_idx_t](#) server, int idx, [l4re_event_stream_info_t](#) *info) [L4_NOTHROW](#)
Get information on a stream.
- long [l4re_event_get_stream_info_for_id](#) (const [l4_cap_idx_t](#) server, [l4_umword_t](#) stream_id, [l4re_event_stream_info_t](#) *info) [L4_NOTHROW](#)
Get info for a stream given a stream id.
- long [l4re_event_get_axis_info](#) (const [l4_cap_idx_t](#) server, [l4_umword_t](#) id, unsigned naxes, unsigned const *axis, [l4re_event_absinfo_t](#) *info) [L4_NOTHROW](#)
Get Axis information for a stream.

14.9.6.1 Detailed Description

Event C interface.

14.9.6.2 Function Documentation

14.9.6.2.1 l4re_event_get_axis_info()

```
long l4re_event_get_axis_info (
    const l4_cap_idx_t server,
    l4_umword_t id,
    unsigned naxes,
    unsigned const * axis,
    l4re_event_absinfo_t * info)
```

Get Axis information for a stream.

Parameters

	<i>server</i>	Server to talk to.
	<i>id</i>	Id of the stream to get information from.
	<i>naxes</i>	Number of axes in <i>axis</i> array.
in	<i>axis</i>	Array of axis IDs whose information should be retrieved.
out	<i>info</i>	Information buffer to store the retrieved axis infos.

Return values

0	Success
<0	Error

See also

[L4Re::Event::get_axis_info](#)

References [L4_END_DECLS](#), and [L4_NOTHROW](#).

14.9.6.2.2 l4re_event_get_buffer()

```
long l4re_event_get_buffer (
    const l4_cap_idx_t server,
    const l4re_ds_t ds)
```

Get an event signal buffer.

Parameters

<i>server</i>	Server to talk to.
<i>ds</i>	Buffer to event data.

Returns

0 for success, <0 on error

See also

[L4Re::Event::get_buffer](#)

References [L4_CV](#), [L4_NOTHROW](#), and [l4re_ds_t](#).

14.9.6.2.3 l4re_event_get_num_streams()

```
long l4re_event_get_num_streams (  
    const l4\_cap\_idx\_t server)
```

Get number of streams.

Parameters

<i>server</i>	Server to talk to.
---------------	--------------------

Returns

0 for success, <0 on error

See also

[L4Re::Event::get_num_streams](#)

References [L4_CV](#), and [L4_NOTHROW](#).

14.9.6.2.4 l4re_event_get_stream_info()

```
long l4re_event_get_stream_info (  
    const l4\_cap\_idx\_t server,  
    int idx,  
    l4re\_event\_stream\_info\_t * info)
```

Get information on a stream.

Parameters

	<i>server</i>	Server to talk to.
	<i>idx</i>	Index value.
out	<i>info</i>	Information buffer.

Returns

0 for success, <0 on error

See also

[L4Re::Event::get_stream_info](#)

References [L4_CV](#), and [L4_NOTHROW](#).

14.9.6.2.5 l4re_event_get_stream_info_for_id()

```
long l4re_event_get_stream_info_for_id (
    const l4_cap_idx_t server,
    l4_umword_t stream_id,
    l4re_event_stream_info_t * info)
```

Get info for a stream given a stream id.

Parameters

	<i>server</i>	Server to talk to.
	<i>stream↔ _id</i>	Stream ID.
out	<i>info</i>	Information buffer.

Returns

0 for success, <0 on error

See also

[L4Re::Event::get_stream_info_for_id](#)

References [L4_CV](#), and [L4_NOTHROW](#).

14.9.7 Log interface

Log C interface.

Collaboration diagram for Log interface:



Functions

- [L4_BEGIN_DECLS](#) void [l4re_log_print](#) (char const *string) [L4_NOTHROW](#)
Write a null terminated string to the default log.
- void [l4re_log_printn](#) (char const *string, int len) [L4_NOTHROW](#)
Write a string of a given length to the default log.
- void [l4re_log_print_srv](#) (const [l4_cap_idx_t](#) logcap, char const *string) [L4_NOTHROW](#)
Write a null terminated string to a log.
- void [l4re_log_printn_srv](#) (const [l4_cap_idx_t](#) logcap, char const *string, int len) [L4_NOTHROW](#)
Write a string of a given length to a log.

14.9.7.1 Detailed Description

Log C interface.

14.9.7.2 Function Documentation

14.9.7.2.1 l4re_log_print()

```
void l4re_log_print (
    char const * string) [inline]
```

Write a null terminated string to the default log.

Parameters

<i>string</i>	Text to print, null terminated.
---------------	---------------------------------

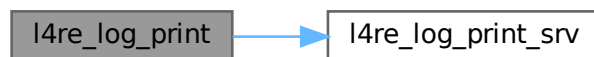
See also

[L4Re::Log::print](#)

Definition at line 81 of file [log.h](#).

References [L4_NOTHROW](#), and [l4re_log_print_srv\(\)](#).

Here is the call graph for this function:



14.9.7.2.2 l4re_log_print_srv()

```
void l4re_log_print_srv (
    const l4\_cap\_idx\_t logcap,
    char const * string)
```

Write a null terminated string to a log.

Parameters

<i>logcap</i>	Log capability (service).
<i>string</i>	Text to print, null terminated.

See also

[L4Re::Log::print](#)

References [L4_CV](#), and [L4_NOTHROW](#).

Referenced by [l4re_log_print\(\)](#).

Here is the caller graph for this function:



14.9.7.2.3 l4re_log_printn()

```
void l4re_log_printn (  
    char const * string,  
    int len) [inline]
```

Write a string of a given length to the default log.

Parameters

<i>string</i>	Text to print, null terminated.
<i>len</i>	Length of string in bytes.

See also

[L4Re::Log::printn](#)

Definition at line 87 of file [log.h](#).

References [L4_NOTHROW](#), and [l4re_log_printn_srv\(\)](#).

Here is the call graph for this function:



14.9.7.2.4 l4re_log_printn_srv()

```
void l4re_log_printn_srv (
    const l4_cap_idx_t logcap,
    char const * string,
    int len)
```

Write a string of a given length to a log.

Parameters

<i>logcap</i>	Log capability (service).
<i>string</i>	Text to print, null terminated.
<i>len</i>	Length of string in bytes.

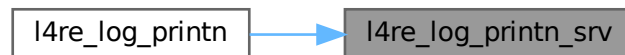
See also

[L4Re::Log::printn](#)

References [L4_CV](#), [L4_INLINE](#), and [L4_NOTHROW](#).

Referenced by [l4re_log_printn\(\)](#).

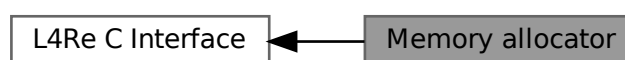
Here is the caller graph for this function:



14.9.8 Memory allocator

Memory allocator C interface.

Collaboration diagram for Memory allocator:



Enumerations

- enum [l4re_ma_flags](#)
Flags for requesting memory at the memory allocator.

Functions

- long [l4re_ma_alloc](#) (long size, [l4re_ds_t](#) const mem, unsigned long flags) [L4_NOTHROW](#)
Allocate memory.
- long [l4re_ma_alloc_align](#) (long size, [l4re_ds_t](#) const mem, unsigned long flags, unsigned long align) [L4_NOTHROW](#)
Allocate memory.
- long [l4re_ma_alloc_align_srv](#) ([l4_cap_idx_t](#) srv, long size, [l4re_ds_t](#) const mem, unsigned long flags, unsigned long align) [L4_NOTHROW](#)
Allocate memory.

14.9.8.1 Detailed Description

Memory allocator C interface.

14.9.8.2 Enumeration Type Documentation

14.9.8.2.1 l4re_ma_flags

```
enum l4re_ma_flags
```

Flags for requesting memory at the memory allocator.

See also

[L4Re::Mem_alloc::Mem_alloc_flags](#)

Definition at line 32 of file [mem_alloc.h](#).

14.9.8.3 Function Documentation

14.9.8.3.1 l4re_ma_alloc()

```
long l4re_ma_alloc (
    long size,
    l4re_ds_t const mem,
    unsigned long flags) [inline]
```

Allocate memory.

Parameters

<i>size</i>	Size in bytes to be requested. Allocation granularity is (super)pages, however, the allocator will store the byte-granular given size as the size of the dataspace and consecutively will use this byte-granular size for servicing the dataspace. Allocators may optionally also implement a maximum allocation strategy: if <i>size</i> is a negative value and <i>flags</i> set the <code>Mem_alloc_flags::Continuous</code> bit, the allocator tries to allocate as much memory as possible leaving an amount of at least <code>-size</code> bytes within the associated quota.
<i>mem</i>	Capability slot where the capability to the dataspace is received.
<i>flags</i>	Special dataspace properties, see l4re_ma_flags

Return values

<i>0</i>	Success
<i>-L4_ERANGE</i>	Given size not supported.
<i>-L4_ENOMEM</i>	Not enough memory available.
<i><0</i>	IPC error

See also

[L4Re::Mem_alloc::alloc](#)

The memory allocator returns a dataspace.

Note

This function is using the [L4Re::Env::env\(\)](#)->`mem_alloc()` service.

Examples

[examples/libs/l4re/c/ma+rm.c](#).

Definition at line 136 of file [mem_alloc.h](#).

References [L4_NOTHROW](#), [l4re_ds_t](#), and [l4re_ma_alloc_align_srv\(\)](#).

Here is the call graph for this function:



14.9.8.3.2 l4re_ma_alloc_align()

```
long l4re_ma_alloc_align (  
    long size,  
    l4re_ds_t const mem,  
    unsigned long flags,  
    unsigned long align) [inline]
```

Allocate memory.

Parameters

<i>size</i>	Size in bytes to be requested. Allocation granularity is (super)pages, however, the allocator will store the byte-granular given size as the size of the dataspace and consecutively will use this byte-granular size for servicing the dataspace. Allocators may optionally also implement a maximum allocation strategy: if <i>size</i> is a negative value and <i>flags</i> set the <code>Mem_alloc_flags::Continuous</code> bit, the allocator tries to allocate as much memory as possible leaving an amount of at least <code>-size</code> bytes within the associated quota.
<i>mem</i>	Capability slot where the capability to the dataspace is received.
<i>flags</i>	Special dataspace properties, see l4re_ma_flags
<i>align</i>	Log2 alignment of dataspace if supported by allocator, will be at least <code>L4_PAGESHIFT</code> , with <code>Super_↔</code> pages flag set at least <code>L4_SUPERPAGESHIFT</code>

Return values

<code>0</code>	Success
<code>-L4_ERANGE</code>	Given size not supported.
<code>-L4_ENOMEM</code>	Not enough memory available.
<code><0</code>	IPC error

See also

[L4Re::Mem_alloc::alloc](#) and
[l4re_ma_alloc](#)

The memory allocator returns a dataspace.

Note

This function is using the [L4Re::Env::env\(\)](#)->`mem_alloc()` service.

Definition at line [144](#) of file [mem_alloc.h](#).

References [L4_NOTHROW](#), [l4re_ds_t](#), and [l4re_ma_alloc_align_srv\(\)](#).

Here is the call graph for this function:



14.9.8.3.3 l4re_ma_alloc_align_srv()

```
long l4re_ma_alloc_align_srv (  
    l4_cap_idx_t srv,  
    long size,  
    l4re_ds_t const mem,  
    unsigned long flags,  
    unsigned long align)
```

Allocate memory.

Parameters

<i>srv</i>	Memory allocator service.
<i>size</i>	Size to be requested.
<i>mem</i>	Capability slot to put the requested dataspace in
<i>flags</i>	Flags, see l4re_ma_flags
<i>align</i>	Log2 alignment of dataspace if supported by allocator, will be at least L4_PAGESHIFT, with Super_↔ pages flag set at least L4_SUPERPAGESHIFT, default 0

Returns

0 on success, <0 on error

See also

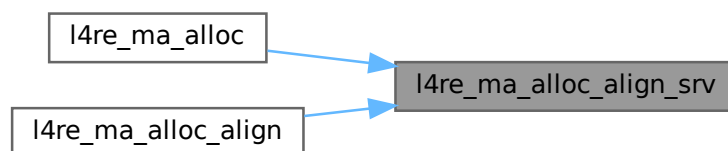
[L4Re::Mem_alloc::alloc](#)

The memory allocator returns a dataspace.

References [L4_CV](#), [L4_INLINE](#), [L4_NOTHROW](#), and [l4re_ds_t](#).

Referenced by [l4re_ma_alloc\(\)](#), and [l4re_ma_alloc_align\(\)](#).

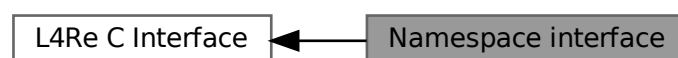
Here is the caller graph for this function:



14.9.9 Namespace interface

Namespace C interface.

Collaboration diagram for Namespace interface:



Enumerations

- enum [l4re_ns_register_flags](#)
Namespace register flags.

Functions

- long [l4re_ns_query_to_srv](#) ([l4re_namespace_t](#) srv, char const *name, [l4_cap_idx_t](#) const cap, int timeout) [L4_NOTHROW](#)
Query the name space for the object named by `name`.
- long [l4re_ns_query_srv](#) ([l4re_namespace_t](#) srv, char const *name, [l4_cap_idx_t](#) const cap) [L4_NOTHROW](#)
Query the name space for the object named by `name`.
- long [l4re_ns_register_obj_srv](#) ([l4re_namespace_t](#) srv, char const *name, [l4_cap_idx_t](#) const obj, unsigned flags) [L4_NOTHROW](#)
Register an object with a name.

Variables

- [L4_BEGIN_DECLS](#) typedef [l4_cap_idx_t](#) [l4re_namespace_t](#)
Namespace type.

14.9.9.1 Detailed Description

Namespace C interface.

14.9.9.2 Enumeration Type Documentation

14.9.9.2.1 l4re_ns_register_flags

```
enum l4re_ns_register_flags
```

Namespace register flags.

See also

[L4Re::Namespace::Register_flags](#)

Definition at line 29 of file [namespace.h](#).

14.9.9.3 Function Documentation

14.9.9.3.1 l4re_ns_query_srv()

```
long l4re_ns_query_srv (  
    l4re\_namespace\_t srv,  
    char const * name,  
    l4\_cap\_idx\_t const cap) [inline]
```

Query the name space for the object named by `name`.

Parameters

<i>srv</i>	Name space server to use for the query.
<i>name</i>	String to query.
<i>cap</i>	Capability slot where the received capability will be stored.

Return values

<i>0</i>	Name could be fully resolved.
<i>>0</i>	Name could only be partly resolved. The number of remaining characters is returned.
<i>-L4_ENOENT</i>	Entry could not be found.
<i>-L4_EAGAIN</i>	Entry exists but no object is yet attached. Try again later.
<i><0</i>	IPC errors, see l4_error_code_t .

Definition at line 95 of file [namespace.h](#).

References [L4_NOTHROW](#), [l4re_namespace_t](#), and [l4re_ns_query_to_srv\(\)](#).

Here is the call graph for this function:



14.9.9.3.2 l4re_ns_query_to_srv()

```

long l4re_ns_query_to_srv (
    l4re_namespace_t srv,
    char const * name,
    l4_cap_idx_t const cap,
    int timeout)
  
```

Query the name space for the object named by *name*.

Parameters

<i>timeout</i>	Timeout of query in milliseconds. The client will only wait if a name already has been registered with the server but no object has been attached yet.
<i>srv</i>	Name space server to use for the query.
<i>name</i>	String to query.
<i>cap</i>	Capability slot where the received capability will be stored.

Return values

<i>0</i>	Name could be fully resolved.
<i>>0</i>	Name could only be partly resolved. The number of remaining characters is returned.
<i>-L4_ENOENT</i>	Entry could not be found.
<i>-L4_EAGAIN</i>	Entry exists but no object is yet attached. Try again later.
<i><0</i>	IPC errors, see l4_error_code_t .

References [L4_CV](#), [L4_INLINE](#), [L4_NOTHROW](#), and [l4re_namespace_t](#).

Referenced by [l4re_ns_query_srv\(\)](#).

Here is the caller graph for this function:



14.9.9.3.3 l4re_ns_register_obj_srv()

```

long l4re_ns_register_obj_srv (
    l4re_namespace_t srv,
    char const * name,
    l4_cap_idx_t const obj,
    unsigned flags)
  
```

Register an object with a name.

Parameters

<i>srv</i>	Name space server to use for the query.
<i>name</i>	Name under which the object should be registered.
<i>obj</i>	Capability to object to register. An invalid capability may be given to only reserve the name for later use.
<i>flags</i>	Flags to assign to the entry, see L4Re::Namespace::Register_flags . Note that the rights that are assigned to a capability are not only determined by the rights given in these flags but also by the rights with which the <code>obj</code> capability was mapped to the name space.

Return values

<i>0</i>	Object was successfully registered with <i>name</i> .
<i>-L4_EEXIST</i>	Name already registered.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.

<code>-L4_ENOMEM</code>	Server has insufficient resources.
<code>-L4_EINVAL</code>	Invalid parameter.
<code>< 0</code>	IPC errors, see l4_error_code_t .

Precondition

The invoked Namespace capability must have the permission [L4_CAP_FPAGE_W](#).

References [L4_CV](#), [L4_INLINE](#), [L4_NOTHROW](#), and [l4re_namespace_t](#).

14.9.10 Parent interface

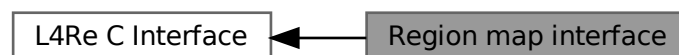
Collaboration diagram for Parent interface:



14.9.11 Region map interface

Region map C interface.

Collaboration diagram for Region map interface:



Enumerations

- enum [l4re_rm_flags_values](#) {
[L4RE_RM_F_R](#) = [L4RE_DS_F_R](#) , [L4RE_RM_F_W](#) = [L4RE_DS_F_W](#) , [L4RE_RM_F_X](#) = [L4RE_DS_F_X](#)
, [L4RE_RM_F_RX](#) = [L4RE_DS_F_RX](#) ,
[L4RE_RM_F_RW](#) = [L4RE_DS_F_RW](#) , [L4RE_RM_F_RWX](#) = [L4RE_DS_F_RWX](#) , [L4RE_RM_F_KERNEL](#)
= 0x100 , [L4RE_RM_F_DETACH_FREE](#) = 0x200 ,
[L4RE_RM_F_PAGER](#) = 0x400 , [L4RE_RM_F_RESERVED](#) = 0x800 , [L4RE_RM_CACHING_SHIFT](#) = 4 ,
[L4RE_RM_F_CACHING](#) = [L4RE_DS_F_CACHING_MASK](#) ,
[L4RE_RM_REGION_FLAGS](#) = 0xffff , [L4RE_RM_F_CACHE_NORMAL](#) = [L4RE_DS_F_NORMAL](#) ,
[L4RE_RM_F_CACHE_BUFFERED](#) = [L4RE_DS_F_BUFFERABLE](#) , [L4RE_RM_F_CACHE_UNCACHED](#)
= [L4RE_DS_F_UNCACHEABLE](#) ,
[L4RE_RM_F_SEARCH_ADDR](#) = 0x020000 , [L4RE_RM_F_IN_AREA](#) = 0x040000 , [L4RE_RM_F_EAGER_MAP](#)
= 0x080000 , [L4RE_RM_F_NO_EAGER_MAP](#) = 0x100000 ,
[L4RE_RM_F_ATTACH_FLAGS](#) = 0x1f0000 }

Flags for region operations.

Functions

- int `l4re_rm_reserve_area` (`l4_addr_t` *start, unsigned long size, `l4re_rm_flags_t` flags, unsigned char align) `L4_NOTHROW`
Reserve the given area in the region map.
- int `l4re_rm_free_area` (`l4_addr_t` addr) `L4_NOTHROW`
Free an area from the region map.
- int `l4re_rm_attach` (void **start, unsigned long size, `l4re_rm_flags_t` flags, `l4re_ds_t` mem, `l4re_rm_offset_t` offs, unsigned char align) `L4_NOTHROW`
Attach a data space to a region.
- int `l4re_rm_detach` (void *addr) `L4_NOTHROW`
Detach and unmap a region from the address space in the current task.
- int `l4re_rm_detach_ds` (void *addr, `l4re_ds_t` *ds) `L4_NOTHROW`
Detach and unmap a region and return affected dataspace in the current task.
- int `l4re_rm_detach_unmap` (`l4_addr_t` addr, `l4_cap_idx_t` task) `L4_NOTHROW`
Detach and unmap in specified task.
- int `l4re_rm_detach_ds_unmap` (void *addr, `l4re_ds_t` *ds, `l4_cap_idx_t` task) `L4_NOTHROW`
Detach and unmap in specified task.
- int `l4re_rm_find` (`l4_addr_t` *addr, unsigned long *size, `l4re_rm_offset_t` *offset, `l4re_rm_flags_t` *flags, `l4re_ds_t` *m) `L4_NOTHROW`
Find a region given an address and size.
- int `l4re_rm_get_info` (`l4_addr_t` addr, char *name, unsigned int len, `l4re_rm_offset_t` *backing_offset) `L4_NOTHROW`
Return auxiliary information of a region.
- void `l4re_rm_show_lists` (void) `L4_NOTHROW`
Dump region map internal data structures.
- int `l4re_rm_reserve_area_srv` (`l4_cap_idx_t` rm, `l4_addr_t` *start, unsigned long size, `l4re_rm_flags_t` flags, unsigned char align) `L4_NOTHROW`
- int `l4re_rm_free_area_srv` (`l4_cap_idx_t` rm, `l4_addr_t` addr) `L4_NOTHROW`
- int `l4re_rm_attach_srv` (`l4_cap_idx_t` rm, void **start, unsigned long size, `l4re_rm_flags_t` flags, `l4re_ds_t` mem, `l4re_rm_offset_t` offs, unsigned char align) `L4_NOTHROW`
- int `l4re_rm_detach_srv` (`l4_cap_idx_t` rm, `l4_addr_t` addr, `l4re_ds_t` *ds, `l4_cap_idx_t` task) `L4_NOTHROW`
- int `l4re_rm_find_srv` (`l4_cap_idx_t` rm, `l4_addr_t` *addr, unsigned long *size, `l4re_rm_offset_t` *offset, `l4re_rm_flags_t` *flags, `l4re_ds_t` *m) `L4_NOTHROW`
- int `l4re_rm_get_info_srv` (`l4_cap_idx_t` rm, `l4_addr_t` addr, char *name, unsigned int len, `l4re_rm_offset_t` *backing_offset) `L4_NOTHROW`
- void `l4re_rm_show_lists_srv` (`l4_cap_idx_t` rm) `L4_NOTHROW`
Dump region map internal data structures.

14.9.11.1 Detailed Description

Region map C interface.

14.9.11.2 Enumeration Type Documentation

14.9.11.2.1 l4re_rm_flags_values

```
enum l4re_rm_flags_values
```

Flags for region operations.

Enumerator

L4RE_RM_F_R	Region is read-only.
L4RE_RM_F_KERNEL	Kernel-provided memory (KUMEM).
L4RE_RM_F_DETACH_FREE	Free the portion of the data space after detach.
L4RE_RM_F_PAGER	Region has a pager.
L4RE_RM_F_RESERVED	Region is reserved (blocked).
L4RE_RM_CACHING_SHIFT	Start of region mapper cache bits.
L4RE_RM_F_CACHING	Mask of all region manager cache bits.
L4RE_RM_REGION_FLAGS	Mask of all region flags.
L4RE_RM_F_CACHE_NORMAL	Cache bits for normal cacheable memory.
L4RE_RM_F_CACHE_BUFFERED	Cache bits for buffered (write combining) memory.
L4RE_RM_F_CACHE_UNCACHED	Cache bits for uncached memory.
L4RE_RM_F_SEARCH_ADDR	Search for a suitable address range.
L4RE_RM_F_IN_AREA	Search only in area, or map into area.
L4RE_RM_F_EAGER_MAP	Eagerly map the attached data space in.
L4RE_RM_F_NO_EAGER_MAP	Prevent eager mapping of the attached data space.
L4RE_RM_F_ATTACH_FLAGS	Mask of all attach flags.

Definition at line 30 of file [rm.h](#).

14.9.11.3 Function Documentation

14.9.11.3.1 l4re_rm_attach()

```
int l4re_rm_attach (
    void ** start,
    unsigned long size,
    l4re_rm_flags_t flags,
    l4re_ds_t mem,
    l4re_rm_offset_t offs,
    unsigned char align) [inline]
```

Attach a data space to a region.

Parameters

<code>in, out</code>	<code>start</code>	Virtual start address where the region manager shall attach the data space. Will be rounded down to the nearest start of a page. If L4Re::Rm::F::Search_addr is given this value is used as the start address to search for a free virtual memory region and the resulting address is returned here. If L4Re::Rm::F::In_area is given the value is used as a selector for the area (see L4Re::Rm::reserve_area) to attach the data space to.
	<code>size</code>	Size of the data space to attach (in bytes). Will be rounded up to the nearest multiple of the page size.

	<i>flags</i>	The flags control how and with which rights the dataspace is attached to the region. See L4Re::Rm::F::Attach_flags and L4Re::Rm::F::Region_flags . The caller must specify the desired rights of the attached region explicitly. The default set of rights is empty. If the <code>F : ↔ Eager_map</code> flag is set this function may also return L4Re::Dataspace::map error codes if the mapping fails.
	<i>mem</i>	Data space.
	<i>offs</i>	Offset into the data space to use.
	<i>align</i>	Alignment of the virtual region, log2-size, default: a page (L4_PAGESHIFT). This is only meaningful if the L4Re::Rm::F::Search_addr flag is used.

Return values

0	Success
-L4_ENOENT	No area could be found (see L4Re::Rm::F::In_area)
-L4_EPERM	Operation not allowed.
-L4_EINVAL	
-L4_EADDRNOTAVAIL	The given address is not available.
<0	IPC errors

Makes the whole or parts of a data space visible in the virtual memory of the corresponding task. The corresponding region in the virtual address space is backed with the contents of the dataspace.

Note

When searching for a free place in the virtual address space, the space between *start* and the end of the virtual address space is

See also

[L4Re::Rm::attach](#)

This function is using the `L4::Env::env()->rm()` service.

Examples

[examples/libs/l4re/c/ma+rm.c](#).

Definition at line 342 of file [rm.h](#).

References [L4_NOTHROW](#), [l4re_ds_t](#), and [l4re_rm_attach_srv\(\)](#).

Here is the call graph for this function:



14.9.11.3.2 l4re_rm_attach_srv()

```
int l4re_rm_attach_srv (
    l4_cap_idx_t rm,
    void ** start,
    unsigned long size,
    l4re_rm_flags_t flags,
    l4re_ds_t mem,
    l4re_rm_offset_t offs,
    unsigned char align)
```

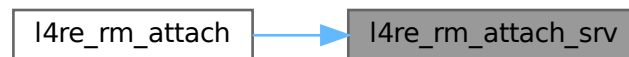
See also

[L4Re::Rm::attach](#)

References [L4_CV](#), [L4_NOTHROW](#), and [l4re_ds_t](#).

Referenced by [l4re_rm_attach\(\)](#).

Here is the caller graph for this function:



14.9.11.3.3 l4re_rm_detach()

```
int l4re_rm_detach (
    void * addr) [inline]
```

Detach and unmap a region from the address space in the current task.

Parameters

<i>addr</i>	Address of the region to detach.
-------------	----------------------------------

Return values

L4Re::Rm::Detach_result	On success.
<code>-L4_ENOENT</code>	No region found.
<code><0</code>	IPC errors

Frees a region in the virtual address space given by *addr*. The corresponding part of the address space is now available again.

Also

See also

[L4Re::Rm::detach](#)

This function is using the `L4::Env::env()->rm()` service.

Definition at line 352 of file [rm.h](#).

References [L4_BASE_TASK_CAP](#), [L4_NOTHROW](#), and [l4re_rm_detach_srv\(\)](#).

Here is the call graph for this function:



14.9.11.3.4 l4re_rm_detach_ds()

```
int l4re_rm_detach_ds (
    void * addr,
    l4re_ds_t * ds) [inline]
```

Detach and unmap a region and return affected dataspace in the current task.

Parameters

	<i>addr</i>	Address of the region to detach.
out	<i>ds</i>	Returns dataspace that is affected.

Return values

L4Re::Rm::Detach_result	On success.
<code>-L4_ENOENT</code>	No region found.
<code><0</code>	IPC errors

Frees a region in the virtual address space given by `addr`. The corresponding part of the address space is now available again.

Also

See also

[L4Re::Rm::detach](#)

This function is using the `L4::Env::env()->rm()` service.

Examples

[examples/libs/l4re/c/ma+rm.c](#).

Definition at line 365 of file [rm.h](#).

References [L4_BASE_TASK_CAP](#), [L4_NOTHROW](#), [l4re_ds_t](#), and [l4re_rm_detach_srv\(\)](#).

Here is the call graph for this function:



14.9.11.3.5 l4re_rm_detach_ds_unmap()

```
int l4re_rm_detach_ds_unmap (
    void * addr,
    l4re_ds_t * ds,
    l4_cap_idx_t task) [inline]
```

Detach and unmap in specified task.

Parameters

	<i>addr</i>	Address of the region to detach.
out	<i>ds</i>	Returns dataspace that is affected.
	<i>task</i>	Task to unmap pages from, specify <code>L4_INVALID_CAP</code> to not unmap

Returns

0 on success, <0 on error

Also

See also

[L4Re::Rm::detach](#)

This function is using the `L4::Env::env()->rm()` service.

Definition at line 372 of file [rm.h](#).

References [L4_NOTHROW](#), [l4re_ds_t](#), and [l4re_rm_detach_srv\(\)](#).

Here is the call graph for this function:



14.9.11.3.6 l4re_rm_detach_srv()

```

int l4re_rm_detach_srv (
    l4_cap_idx_t rm,
    l4_addr_t addr,
    l4re_ds_t * ds,
    l4_cap_idx_t task)
  
```

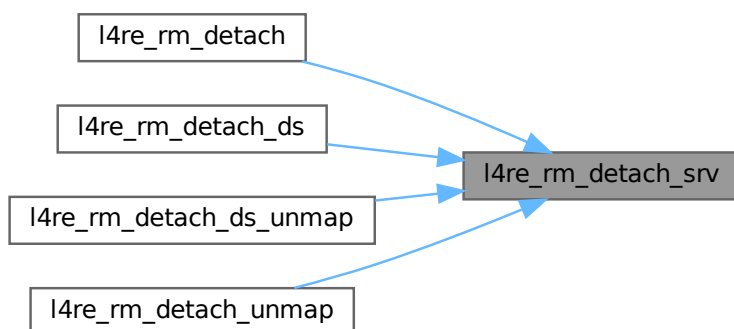
See also

[L4Re::Rm::detach](#)

References [L4_CV](#), [L4_NOTHROW](#), and [l4re_ds_t](#).

Referenced by [l4re_rm_detach\(\)](#), [l4re_rm_detach_ds\(\)](#), [l4re_rm_detach_ds_unmap\(\)](#), and [l4re_rm_detach_unmap\(\)](#).

Here is the caller graph for this function:



14.9.11.3.7 l4re_rm_detach_unmap()

```
int l4re_rm_detach_unmap (
    l4_addr_t addr,
    l4_cap_idx_t task) [inline]
```

Detach and unmap in specified task.

Parameters

<i>addr</i>	Address of the region to detach.
<i>task</i>	Task to unmap pages from, specify L4_INVALID_CAP to not unmap

Returns

0 on success, <0 on error

Also

See also

[L4Re::Rm::detach](#)

This function is using the `L4::Env::env()->rm()` service.

Definition at line 359 of file [rm.h](#).

References [L4_NOTHROW](#), and [l4re_rm_detach_srv\(\)](#).

Here is the call graph for this function:



14.9.11.3.8 l4re_rm_find()

```
int l4re_rm_find (
    l4_addr_t * addr,
    unsigned long * size,
    l4re_rm_offset_t * offset,
    l4re_rm_flags_t * flags,
    l4re_ds_t * m) [inline]
```

Find a region given an address and size.

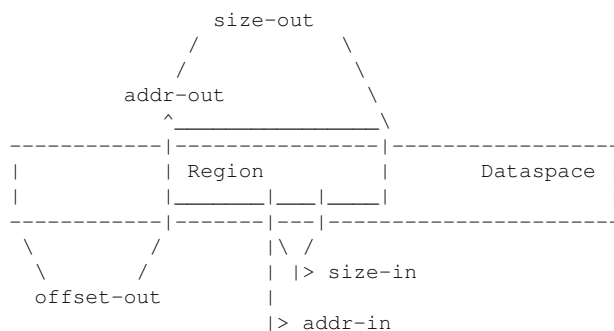
Parameters

<i>in, out</i>	<i>addr</i>	Address to look for. Returns the start address of the found region.
<i>in, out</i>	<i>size</i>	Size of the area to look for (in bytes). Returns the size of the found region (in bytes).
<i>out</i>	<i>offset</i>	Offset at the beginning of the region within the associated dataspace.
<i>out</i>	<i>flags</i>	Region flags, see <code>F::Region_flags</code> (and <code>F::In_area</code>).
<i>out</i>	<i>m</i>	Associated dataspace or paging service.

Return values

<i>0</i>	Success
<code>-L4_EPERM</code>	Operation not allowed.
<code>-L4_ENOENT</code>	No region found.
<i><0</i>	IPC errors

This function returns the properties of the region that contains the area described by the `addr` and `size` parameter. If no such region is found but a reserved area, the area is returned and `F::In_area` is set in `flags`. Note, in the case of an area the `offset` and `m` return values are invalid.



Note

The value of the `size` input parameter should be 1 to assure that a region can be determined unambiguously.

See also

[L4Re::Rm::find](#)

Definition at line 379 of file [rm.h](#).

References [L4_NOTHROW](#), [l4re_ds_t](#), and [l4re_rm_find_srv\(\)](#).

Here is the call graph for this function:



14.9.11.3.9 l4re_rm_find_srv()

```
int l4re_rm_find_srv (
    l4_cap_idx_t rm,
    l4_addr_t * addr,
    unsigned long * size,
    l4re_rm_offset_t * offset,
    l4re_rm_flags_t * flags,
    l4re_ds_t * m)
```

See also

[L4Re::Rm::find](#)

References [L4_CV](#), [L4_NOTHROW](#), and [l4re_ds_t](#).

Referenced by [l4re_rm_find\(\)](#).

Here is the caller graph for this function:



14.9.11.3.10 l4re_rm_free_area()

```
int l4re_rm_free_area (
    l4_addr_t addr) [inline]
```

Free an area from the region map.

Parameters

<i>addr</i>	An address within the area to free.
-------------	-------------------------------------

Return values

0	Success
-L4_ENOENT	No area found.
<0	IPC errors

Note

The data spaces that are attached to that area are not detached by this operation.

See also

`reserve_area()` for more information about areas.

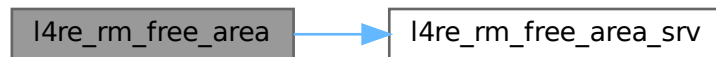
[L4Re::Rm::free_area](#)

This function is using the `L4::Env::env()->rm()` service.

Definition at line 336 of file `rm.h`.

References [L4_NOTHROW](#), and [l4re_rm_free_area_srv\(\)](#).

Here is the call graph for this function:

**14.9.11.3.11 l4re_rm_free_area_srv()**

```
int l4re_rm_free_area_srv (  
    l4_cap_idx_t rm,  
    l4_addr_t addr)
```

See also

[L4Re::Rm::free_area](#)

References [L4_CV](#), and [L4_NOTHROW](#).

Referenced by [l4re_rm_free_area\(\)](#).

Here is the caller graph for this function:



14.9.11.3.12 l4re_rm_get_info()

```
int l4re_rm_get_info (
    l4_addr_t addr,
    char * name,
    unsigned int len,
    l4re_rm_offset_t * backing_offset) [inline]
```

Return auxiliary information of a region.

This is a debugging feature and might not be available.

Parameters

	<i>addr</i>	Virtual address of the region.
out	<i>name</i>	Name of the region.
out	<i>backing_offset</i>	Backing offset information.

Return values

0	Success
-L4_ENOENT	Region not found.
-L4_ENOSYS	Function not available.
<0	IPC errors

Parameters

<i>len</i>	Length of the name given in name argument, in bytes.
------------	--

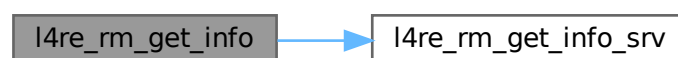
See also

[L4Re::Rm::get_info](#)

Definition at line 395 of file [rm.h](#).

References [L4_NOTHROW](#), and [l4re_rm_get_info_srv\(\)](#).

Here is the call graph for this function:



14.9.11.3.13 l4re_rm_get_info_srv()

```
int l4re_rm_get_info_srv (
    l4_cap_idx_t rm,
    l4_addr_t addr,
    char * name,
    unsigned int len,
    l4re_rm_offset_t * backing_offset)
```

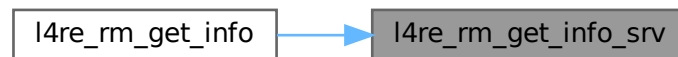
See also

[L4Re::Rm::get_info](#)

References [L4_CV](#), and [L4_NOTHROW](#).

Referenced by [l4re_rm_get_info\(\)](#).

Here is the caller graph for this function:



14.9.11.3.14 l4re_rm_reserve_area()

```
int l4re_rm_reserve_area (
    l4_addr_t * start,
    unsigned long size,
    l4re_rm_flags_t flags,
    unsigned char align) [inline]
```

Reserve the given area in the region map.

Parameters

<i>in, out</i>	<i>start</i>	The virtual start address of the area to reserve. Returns the start address of the area.
	<i>size</i>	The size of the area to reserve (in bytes).
	<i>flags</i>	Flags for the reserved area (see L4Re::Rm::F::Region_flags and L4Re::Rm::F::Attach_flags).
	<i>align</i>	Alignment of area if searched as bits (log2 value).

Return values

0	Success
-L4_EADDRNOTAVAIL	The given area cannot be reserved.
<0	IPC errors

This function reserves an area within the virtual address space managed by the region map. There are two kinds of areas available:

- Reserved areas (*flags* = [L4Re::Rm::F::Reserved](#)), where no data spaces can be attached
- Special purpose areas (*flags* = 0), where data spaces can be attached to the area via the [L4Re::Rm::F::In_area](#) flag and a start address within the area itself.

Note

When searching for a free place in the virtual address space (with *flags* = [L4Re::Rm::F::Search_addr](#)), the space between *start* and the end of the virtual address space is searched.

See also

[L4Re::Rm::reserve_area](#)

This function is using the `L4::Env::env()->rm()` service.

Definition at line 328 of file [rm.h](#).

References [L4_NOTHROW](#), and [l4re_rm_reserve_area_srv\(\)](#).

Here is the call graph for this function:



14.9.11.3.15 l4re_rm_reserve_area_srv()

```

int l4re_rm_reserve_area_srv (
    l4_cap_idx_t rm,
    l4_addr_t * start,
    unsigned long size,
    l4re_rm_flags_t flags,
    unsigned char align)
  
```

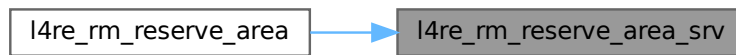
See also

[L4Re::Rm::reserve_area](#)

References [L4_CV](#), and [L4_NOTHROW](#).

Referenced by [l4re_rm_reserve_area\(\)](#).

Here is the caller graph for this function:



14.9.11.3.16 l4re_rm_show_lists()

```
void l4re_rm_show_lists (
    void ) [inline]
```

Dump region map internal data structures.

This function is using the `L4::Env::env()->rm()` service.

Definition at line 387 of file [rm.h](#).

References [L4_NOTHROW](#), and [l4re_rm_show_lists_srv\(\)](#).

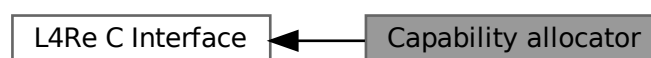
Here is the call graph for this function:



14.9.12 Capability allocator

Capability allocator C interface.

Collaboration diagram for Capability allocator:



Functions

- [L4_BEGIN_DECLS](#) [l4_cap_idx_t](#) **l4re_util_cap_alloc** (void) [L4_NOTHROW](#)
Get free capability index at capability allocator.
- void **l4re_util_cap_free** ([l4_cap_idx_t](#) cap) [L4_NOTHROW](#)
Return capability index to capability allocator.
- void **l4re_util_cap_free_um** ([l4_cap_idx_t](#) cap) [L4_NOTHROW](#)
Return capability index to capability allocator, and unmaps the object.
- long **l4re_util_cap_last** (void) [L4_NOTHROW](#)
Return last capability index the allocator can return.

14.9.12.1 Detailed Description

Capability allocator C interface.

14.9.12.2 Function Documentation

14.9.12.2.1 l4re_util_cap_last()

```
long l4re_util_cap_last (
    void )
```

Return last capability index the allocator can return.

Returns

last/biggest capability index the allocator can return

References [L4_END_DECLS](#), and [L4_NOTHROW](#).

14.9.13 Kumem allocator utility

Kumem allocator utility C interface.

Collaboration diagram for Kumem allocator utility:



Kumem allocator utility C interface.

14.9.14 Video API

Collaboration diagram for Video API:



Data Structures

- struct [l4re_video_color_component_t](#)
Color component structure.
- struct [l4re_video_pixel_info_t](#)
Pixel_info structure.
- struct [l4re_video_goos_info_t](#)
Goos information structure.
- struct [l4re_video_view_info_t](#)
View information structure.
- struct [l4re_video_view_t](#)
C representation of a goos view.

Typedefs

- typedef struct [l4re_video_color_component_t](#) **[l4re_video_color_component_t](#)**
Color component structure.
- typedef struct [l4re_video_pixel_info_t](#) **[l4re_video_pixel_info_t](#)**
Pixel_info structure.
- typedef [l4_cap_idx_t](#) **[l4re_video_goos_t](#)**
Goos object type.
- typedef struct [l4re_video_view_info_t](#) **[l4re_video_view_info_t](#)**
View information structure.
- typedef struct [l4re_video_view_t](#) **[l4re_video_view_t](#)**
C representation of a goos view.

Enumerations

- enum [l4re_video_goos_info_flags_t](#) { [F_l4re_video_goos_auto_refresh](#) = 0x01 , [F_l4re_video_goos_pointer](#) = 0x02 , [F_l4re_video_goos_dynamic_views](#) = 0x04 , [F_l4re_video_goos_dynamic_buffers](#) = 0x08 }
Flags of information on the goos.
- enum [l4re_video_view_info_flags_t](#) {
[F_l4re_video_view_none](#) = 0x00 , [F_l4re_video_view_set_buffer](#) = 0x01 , [F_l4re_video_view_set_buffer_offset](#) = 0x02 , [F_l4re_video_view_set_bytes_per_line](#) = 0x04 ,
[F_l4re_video_view_set_pixel](#) = 0x08 , [F_l4re_video_view_set_position](#) = 0x10 , [F_l4re_video_view_dyn_allocated](#) = 0x20 , [F_l4re_video_view_set_background](#) = 0x40 ,
[F_l4re_video_view_set_flags](#) = 0x80 , **[F_l4re_video_view_fully_dynamic](#)** , [F_l4re_video_view_above](#) = 0x01000 , [F_l4re_video_view_flags_mask](#) = 0xff000 }
Flags of information on a view.

Functions

- [L4_BEGIN_DECLS](#) `int l4re_video_goos_info (l4re_video_goos_t goos, l4re_video_goos_info_t *ginfo) L4_NOTHROW`
Get information on a goos.
- `int l4re_video_goos_refresh (l4re_video_goos_t goos, int x, int y, int w, int h) L4_NOTHROW`
Flush a rectangle of pixels of the goos screen.
- `int l4re_video_goos_create_buffer (l4re_video_goos_t goos, unsigned long size, l4_cap_idx_t buffer) L4_NOTHROW`
Create a new buffer (memory buffer) for pixel data.
- `int l4re_video_goos_delete_buffer (l4re_video_goos_t goos, unsigned idx) L4_NOTHROW`
Delete a pixel buffer.
- `int l4re_video_goos_get_static_buffer (l4re_video_goos_t goos, unsigned idx, l4_cap_idx_t buffer) L4_NOTHROW`
Get the data-space capability of the static pixel buffer.
- `int l4re_video_goos_create_view (l4re_video_goos_t goos, l4re_video_view_t *view) L4_NOTHROW`
Create a new view (.
- `int l4re_video_goos_delete_view (l4re_video_goos_t goos, l4re_video_view_t *view) L4_NOTHROW`
Delete a view.
- `int l4re_video_goos_get_view (l4re_video_goos_t goos, unsigned idx, l4re_video_view_t *view) L4_NOTHROW`
Get a view for the given index.
- [L4_BEGIN_DECLS](#) `int l4re_video_view_refresh (l4re_video_view_t *view, int x, int y, int w, int h) L4_NOTHROW`
Flush the given rectangle of pixels of the given view.
- `int l4re_video_view_get_info (l4re_video_view_t *view, l4re_video_view_info_t *info) L4_NOTHROW`
Retrieve information about the given view.
- `int l4re_video_view_set_info (l4re_video_view_t *view, l4re_video_view_info_t *info) L4_NOTHROW`
Set properties of the view.
- `int l4re_video_view_set_viewport (l4re_video_view_t *view, int x, int y, int w, int h, unsigned long bofs) L4_NOTHROW`
Set the viewport parameters of a view.
- `int l4re_video_view_stack (l4re_video_view_t *view, l4re_video_view_t *pivot, int behind) L4_NOTHROW`
Change the stacking order in the stack of visible views.

14.9.14.1 Detailed Description

14.9.14.2 Typedef Documentation

14.9.14.2.1 l4re_video_view_t

```
typedef struct l4re_video_view_t l4re_video_view_t
```

C representation of a goos view.

A view is a visible rectangle that provides a view to the contents of a buffer (frame buffer) memory object and is placed on a real screen.

14.9.14.3 Enumeration Type Documentation

14.9.14.3.1 l4re_video_goos_info_flags_t

```
enum l4re_video_goos_info_flags_t
```

Flags of information on the goos.

Enumerator

F_l4re_video_goos_auto_refresh	The graphics display is automatically refreshed.
F_l4re_video_goos_pointer	We have a mouse pointer.
F_l4re_video_goos_dynamic_views	Supports dynamically allocated views.
F_l4re_video_goos_dynamic_buffers	Supports dynamically allocated buffers.

Definition at line 29 of file [goos.h](#).

14.9.14.3.2 l4re_video_view_info_flags_t

enum [l4re_video_view_info_flags_t](#)

Flags of information on a view.

Enumerator

F_l4re_video_view_none	everything for this view is static (the VESA-FB case)
F_l4re_video_view_set_buffer	buffer object for this view can be changed
F_l4re_video_view_set_buffer_offset	buffer offset can be set
F_l4re_video_view_set_bytes_per_line	bytes per line can be set
F_l4re_video_view_set_pixel	pixel type can be set
F_l4re_video_view_set_position	position on screen can be set
F_l4re_video_view_dyn_allocated	View is dynamically allocated.
F_l4re_video_view_set_background	Set view as background for session.
F_l4re_video_view_set_flags	Set view property flags.
F_l4re_video_view_above	Flag the view as stay on top.
F_l4re_video_view_flags_mask	Mask containing all possible property flags.

Definition at line 23 of file [view.h](#).

14.9.14.4 Function Documentation

14.9.14.4.1 l4re_video_goos_create_buffer()

```
int l4re_video_goos_create_buffer (
    l4re\_video\_goos\_t goos,
    unsigned long size,
    l4\_cap\_idx\_t buffer)
```

Create a new buffer (memory buffer) for pixel data.

Parameters

<i>goos</i>	the target object for the operation.
<i>size</i>	the size in bytes for the pixel buffer.

<i>buffer</i>	a capability index to receive the data-space capability for the buffer.
---------------	---

Returns

>=0: The index of the created buffer (used to assign views and for deletion). < 0: on error

References [L4_CV](#), and [L4_NOTHROW](#).

14.9.14.4.2 l4re_video_goos_create_view()

```
int l4re_video_goos_create_view (
    l4re_video_goos_t goos,
    l4re_video_view_t * view)
```

Create a new view (.

See also

[l4re_video_view_t](#)

Parameters

	<i>goos</i>	the goos session to use.
out	<i>view</i>	structure initialized to the new view.

References [L4_CV](#), and [L4_NOTHROW](#).

14.9.14.4.3 l4re_video_goos_delete_buffer()

```
int l4re_video_goos_delete_buffer (
    l4re_video_goos_t goos,
    unsigned idx)
```

Delete a pixel buffer.

Parameters

<i>goos</i>	the target goos object.
<i>idx</i>	the buffer index of the buffer to delete (the return value of l4re_video_goos_create_buffer())

References [L4_CV](#), and [L4_NOTHROW](#).

14.9.14.4.4 l4re_video_goos_delete_view()

```
int l4re_video_goos_delete_view (
    l4re_video_goos_t goos,
    l4re_video_view_t * view)
```

Delete a view.

Parameters

<i>goos</i>	the goos session to use.
<i>view</i>	the view to delete, the given data-structure is invalid afterwards.

References [L4_CV](#), and [L4_NOTHROW](#).

14.9.14.4.5 l4re_video_goos_get_static_buffer()

```
int l4re_video_goos_get_static_buffer (
    l4re_video_goos_t goos,
    unsigned idx,
    l4_cap_idx_t buffer)
```

Get the data-space capability of the static pixel buffer.

Parameters

<i>goos</i>	The target goos object.
<i>idx</i>	Index of the static buffer.
<i>buffer</i>	A capability index to receive the data-space capability.

This function allows access to static, preexisting pixel buffers. Such static buffers exist for static configurations, such as the VESA framebuffer.

References [L4_CV](#), and [L4_NOTHROW](#).

14.9.14.4.6 l4re_video_goos_get_view()

```
int l4re_video_goos_get_view (
    l4re_video_goos_t goos,
    unsigned idx,
    l4re_video_view_t * view)
```

Get a view for the given index.

Parameters

	<i>goos</i>	the target goos session.
	<i>idx</i>	the index of the view to retrieve.
out	<i>view</i>	structure initialized to the view with the given index.

This function allows to access static views as provided by the VESA framebuffer (the monitor). However, it also allows to access dynamic views created with [l4re_video_goos_create_view\(\)](#).

References [L4_END_DECLS](#), and [L4_NOTHROW](#).

14.9.14.4.7 l4re_video_goos_info()

```
L4_BEGIN_DECLS int l4re_video_goos_info (
    l4re_video_goos_t goos,
    l4re_video_goos_info_t * ginfo)
```

Get information on a goos.

Parameters

	<i>goos</i>	Goos object
out	<i>ginfo</i>	Pointer to goos information structure.

Returns

0 for success, <0 on error

- [-L4_ENODEV](#)
- IPC errors

References [L4_CV](#), and [L4_NOTHROW](#).

14.9.14.4.8 l4re_video_goos_refresh()

```
int l4re_video_goos_refresh (
    l4re_video_goos_t goos,
    int x,
    int y,
    int w,
    int h)
```

Flush a rectangle of pixels of the goos screen.

Parameters

<i>goos</i>	the target object of the operation.
<i>x</i>	the x-coordinate of the upper left corner of the rectangle
<i>y</i>	the y-coordinate of the upper left corner of the rectangle
<i>w</i>	the width of the rectangle to be flushed
<i>h</i>	the height of the rectangle

References [L4_CV](#), and [L4_NOTHROW](#).

14.9.14.4.9 l4re_video_view_get_info()

```
int l4re_video_view_get_info (
    l4re_video_view_t * view,
    l4re_video_view_info_t * info)
```

Retrieve information about the given *view*.

Parameters

	<i>view</i>	the target view for the operation.
out	<i>info</i>	a buffer receiving the information about the view.

References [L4_CV](#), and [L4_NOTHROW](#).

14.9.14.4.10 l4re_video_view_refresh()

```
L4_BEGIN_DECLS int l4re_video_view_refresh (
    l4re_video_view_t * view,
    int x,
    int y,
    int w,
    int h)
```

Flush the given rectangle of pixels of the given *view*.

Parameters

<i>view</i>	the target view of the operation.
<i>x</i>	x-coordinate of the upper left corner
<i>y</i>	y-coordinate of the upper left corner
<i>w</i>	the width of the rectangle
<i>h</i>	the height of the rectangle

References [L4_CV](#), and [L4_NOTHROW](#).

14.9.14.4.11 l4re_video_view_set_info()

```
int l4re_video_view_set_info (
    l4re_video_view_t * view,
    l4re_video_view_info_t * info)
```

Set properties of the view.

Parameters

<i>view</i>	the target view of the operation.
<i>info</i>	the parameters to be set on the view.

Which parameters can be manipulated on a given view can be figured out with [l4re_video_view_get_info\(\)](#) and this depends on the concrete instance the view object.

References [L4_CV](#), and [L4_NOTHROW](#).

14.9.14.4.12 l4re_video_view_set_viewport()

```
int l4re_video_view_set_viewport (
    l4re_video_view_t * view,
    int x,
    int y,
    int w,
    int h,
    unsigned long bofs)
```

Set the viewport parameters of a view.

Parameters

<i>view</i>	the target view of the operation.
<i>x</i>	the x-coordinate of the upper left corner on the screen.
<i>y</i>	the y-coordinate of the upper left corner on the screen.
<i>w</i>	the width of the view.
<i>h</i>	the height of the view.
<i>bofs</i>	the offset (in bytes) of the upper left pixel in the memory buffer

This function is a convenience wrapper for [l4re_video_view_set_info\(\)](#), just setting the often changed parameters of a dynamic view. With this function a view can be placed on the real screen and at the same time on its backing buffer.

References [L4_CV](#), and [L4_NOTHROW](#).

14.9.14.4.13 l4re_video_view_stack()

```
int l4re_video_view_stack (
    l4re_video_view_t * view,
    l4re_video_view_t * pivot,
    int behind)
```

Change the stacking order in the stack of visible views.

Parameters

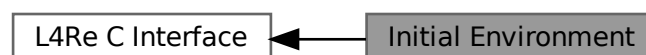
<i>view</i>	the target view for the operation.
<i>pivot</i>	the neighbor view, relative to which <i>view</i> shall be stacked. a NULL value allows top (<i>behind</i> = 1) and bottom (<i>behind</i> = 0) placement of the view.
<i>behind</i>	describes the placement of the view relative to the <i>pivot</i> view.

References [L4_END_DECLS](#), and [L4_NOTHROW](#).

14.9.15 Initial Environment

C interface of the initial environment that is provided to an [L4](#) task.

Collaboration diagram for Initial Environment:



Data Structures

- struct [l4re_env_cap_entry_t](#)
Entry in the [L4Re](#) environment array for the named initial objects.

Typedefs

- typedef struct [l4re_env_cap_entry_t](#) [l4re_env_cap_entry_t](#)
Entry in the [L4Re](#) environment array for the named initial objects.

Functions

- [l4re_env_t](#) * [l4re_env](#) (void) [L4_NOTHROW](#)
Get [L4Re](#) initial environment.
- [l4_kernel_info_t](#) const * [l4re_kip](#) (void) [L4_NOTHROW](#)
Get Kernel Info Page.
- [l4_cap_idx_t](#) [l4re_env_get_cap](#) (char const *name) [L4_NOTHROW](#)
Get the capability selector for the object named name.
- [l4_cap_idx_t](#) [l4re_env_get_cap_e](#) (char const *name, [l4re_env_t](#) const *e) [L4_NOTHROW](#)
Get the capability selector for the object named name.
- [l4re_env_cap_entry_t](#) const * [l4re_env_get_cap_l](#) (char const *name, unsigned l, [l4re_env_t](#) const *e) [L4_NOTHROW](#)
Get the full [l4re_env_cap_entry_t](#) for the object named name.

14.9.15.1 Detailed Description

C interface of the initial environment that is provided to an [L4](#) task.

Include File

```
#include <l4/re/env.h>
```

For an explanation of the default task capabilities see [l4_default_caps_t](#).

For the C++ interface refer to [L4Re::Env](#).

14.9.15.2 Function Documentation

14.9.15.2.1 [l4re_env\(\)](#)

```
l4re\_env\_t * l4re\_env (  
    void ) [inline]
```

Get [L4Re](#) initial environment.

Returns

Pointer to [L4Re](#) initial environment.

Examples

[examples/sys/aliens/main.c](#), [examples/sys/isr/main.c](#), [examples/sys/singlestep/main.c](#), [examples/sys/start-with-exc/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

Definition at line 181 of file [env.h](#).

References [L4_NOTHROW](#).

Referenced by [l4re_env_get_cap\(\)](#).

Here is the caller graph for this function:

**14.9.15.2.2 l4re_env_get_cap()**

```

l4_cap_idx_t l4re_env_get_cap (
    char const * name) [inline]
  
```

Get the capability selector for the object named *name*.

Parameters

<i>name</i>	is the name of the object to lookup in the initial objects.
-------------	---

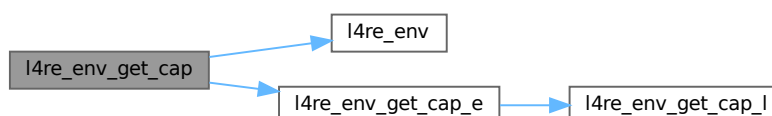
Returns

A valid capability selector if the object exists or an invalid capability selector if not ([l4_is_invalid_cap\(\)](#)).

Definition at line 220 of file [env.h](#).

References [L4_NOTHROW](#), [l4re_env\(\)](#), and [l4re_env_get_cap_e\(\)](#).

Here is the call graph for this function:



14.9.15.2.3 l4re_env_get_cap_e()

```
l4_cap_idx_t l4re_env_get_cap_e (
    char const * name,
    l4re_env_t const * e) [inline]
```

Get the capability selector for the object named *name*.

Parameters

<i>name</i>	is the name of the object to lookup in the initial objects.
<i>e</i>	is the environment structure to use for the operation.

Returns

A valid capability selector if the object exists or an invalid capability selector if not ([l4_is_invalid_cap\(\)](#)).

Definition at line 207 of file [env.h](#).

References [l4re_env_cap_entry_t::cap](#), [L4_INVALID_CAP](#), [L4_NOTHROW](#), and [l4re_env_get_cap_l\(\)](#).

Referenced by [l4re_env_get_cap\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.9.15.2.4 l4re_env_get_cap_l()

```
l4re_env_cap_entry_t const * l4re_env_get_cap_l (
    char const * name,
    unsigned l,
    l4re_env_t const * e) [inline]
```

Get the full [l4re_env_cap_entry_t](#) for the object named *name*.

Parameters

<i>name</i>	is the name of the object to lookup in the initial objects.
<i>l</i>	is the length of the name string, thus <i>name</i> might not be zero terminated.
<i>e</i>	is the environment structure to use for the operation.

Returns

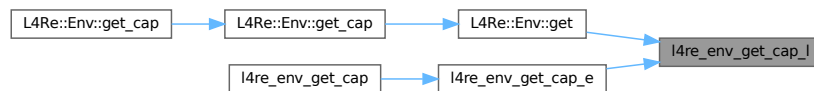
A pointer to an [l4re_env_cap_entry_t](#) if the object exists or NULL if not.

Definition at line 189 of file [env.h](#).

References [l4re_env_cap_entry_t::flags](#), [L4_NOTHROW](#), and [l4re_env_cap_entry_t::name](#).

Referenced by [L4Re::Env::get\(\)](#), and [l4re_env_get_cap_e\(\)](#).

Here is the caller graph for this function:



14.9.15.2.5 l4re_kip()

```
l4_kernel_info_t const * l4re_kip (
    void ) [inline]
```

Get Kernel Info Page.

Returns

Pointer to Kernel Info Page (KIP) structure.

Examples

[examples/libs/shmc/prodcons.c](#), and [examples/sys/aliens/main.c](#).

Definition at line 185 of file [env.h](#).

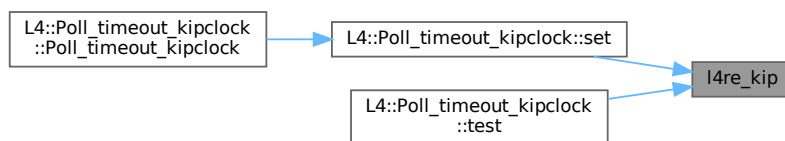
References [l4_kip\(\)](#), and [L4_NOTHROW](#).

Referenced by [L4::Poll_timeout_kipclock::set\(\)](#), and [L4::Poll_timeout_kipclock::test\(\)](#).

Here is the call graph for this function:



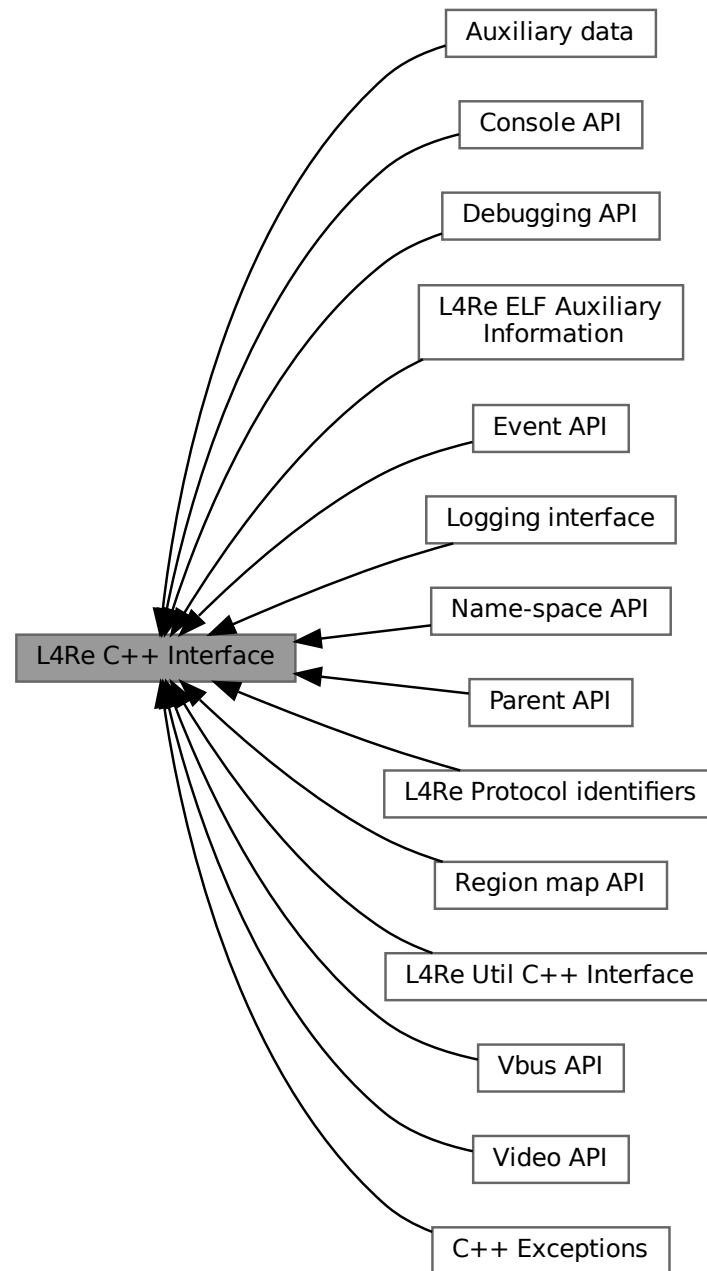
Here is the caller graph for this function:



14.10 L4Re C++ Interface

Documentation of the [L4](#) Runtime Environment C++ API.

Collaboration diagram for L4Re C++ Interface:



Topics

- [L4Re Util C++ Interface](#) ??
Documentation of the [L4](#) Runtime Environment utility functionality in C++.
- [Console API](#) ??

•	<i>Console interface.</i>	
•	Debugging API	??
	<i>Debugging Interface.</i>	
•	L4Re ELF Auxiliary Information	??
	<i>API for embedding auxiliary information into binary programs.</i>	
•	Event API	??
	<i>Event API.</i>	
•	Auxiliary data	??
•	Logging interface	??
	<i>Interface for log output.</i>	
•	Name-space API	??
	<i>API for name spaces that store capabilities.</i>	
•	Parent API	??
	<i>Parent interface.</i>	
•	L4Re Protocol identifiers	??
	<i>Fix L4Re Protocol Constants.</i>	
•	Region map API	??
	<i>Virtual address-space management.</i>	
•	Video API	??
	<i>API for framebuffer based graphics.</i>	
•	C++ Exceptions	??
•	Vbus API	??
	<i>C++ interface of the Vbus API.</i>	

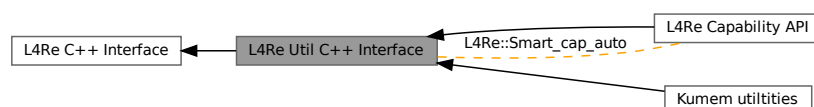
14.10.1 Detailed Description

Documentation of the [L4](#) Runtime Environment C++ API.

14.10.2 L4Re Util C++ Interface

Documentation of the [L4](#) Runtime Environment utility functionality in C++.

Collaboration diagram for L4Re Util C++ Interface:



Topics

- [L4Re Capability API](#) ??
- [Kumem•utilities](#) ??

Data Structures

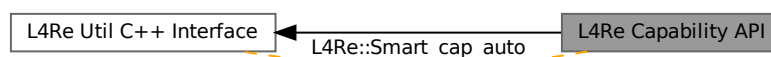
- class [L4Re::Smart_cap_auto< Unmap_flags >](#)
Helper for [Unique_cap](#) and [Unique_del_cap](#).
- class [L4Re::Util::Cap_alloc_base](#)
Capability allocator.
- class [L4Re::Util::Br_manager](#)
Buffer-register (BR) manager for [L4::Server](#).
- class [L4Re::Util::Counting_cap_alloc< COUNTERTYPE, Dbg >](#)
Internal reference-counting cap allocator.
- class [L4Re::Util::Event_buffer_t< PAYLOAD >](#)
Event_buffer utility class.
- class [L4Re::Util::Event_buffer_consumer_t< PAYLOAD >](#)
An event buffer consumer.
- class [L4Re::Util::Vcon_svr< SVR >](#)
[Console](#) server template class.
- class [L4Re::Util::Video::Goos_svr](#)
Goos server class.

14.10.2.1 Detailed Description

Documentation of the [L4](#) Runtime Environment utility functionality in C++.

14.10.2.2 L4Re Capability API

Collaboration diagram for L4Re Capability API:



Data Structures

- class [L4Re::Cap_alloc](#)
Capability allocator interface.
- class [L4Re::Smart_cap_auto< Unmap_flags >](#)
Helper for [Unique_cap](#) and [Unique_del_cap](#).
- class [L4Re::Smart_count_cap< Unmap_flags >](#)
Helper for [Ref_cap](#) and [Ref_del_cap](#).
- class [L4Re::Util::Smart_cap_auto< Unmap_flags >](#)
Helper for [Unique_cap](#) and [Unique_del_cap](#).
- class [L4Re::Util::Smart_count_cap< Unmap_flags >](#)
Helper for [Ref_cap](#) and [Ref_del_cap](#).
- struct [L4Re::Util::Ref_cap< T >](#)
Automatic capability that implements automatic free and unmap of the capability selector.
- struct [L4Re::Util::Ref_del_cap< T >](#)
Automatic capability that implements automatic free and unmap+delete of the capability selector.

Functions

- `template<typename T>
Ref_cap< T >::Cap L4Re::Util::make_ref_cap ()`
Allocate a capability slot and wrap it in a [Ref_cap](#).
- `template<typename T>
Ref_del_cap< T >::Cap L4Re::Util::make_ref_del_cap ()`
Allocate a capability slot and wrap it in a [Ref_del_cap](#).
- `virtual L4Re::Cap_alloc::~Cap_alloc ()=0`
Destructor.

Variables

- [_Cap_alloc](#) & [L4Re::Util::cap_alloc](#)
Capability allocator.

14.10.2.2.1 Detailed Description

14.10.2.2.2 Function Documentation

14.10.2.2.2.1 [make_ref_cap\(\)](#)

```
template<typename T>
Ref\_cap< T >::Cap L4Re::Util::make\_ref\_cap ()
```

Allocate a capability slot and wrap it in a [Ref_cap](#).

Template Parameters

<i>T</i>	Type of capability the slot is used for.
----------	--

Definition at line 195 of file [cap_alloc](#).

References [cap_alloc](#).

14.10.2.2.2 make_ref_del_cap()

```
template<typename T>
Ref_del_cap< T >::Cap L4Re::Util::make_ref_del_cap ()
```

Allocate a capability slot and wrap it in a [Ref_del_cap](#).

Template Parameters

<i>T</i>	Type of capability the slot is used for.
----------	--

Definition at line 204 of file [cap_alloc](#).

References [cap_alloc](#).

14.10.2.2.3 Variable Documentation

14.10.2.2.3.1 cap_alloc

```
_Cap_alloc& L4Re::Util::cap_alloc [extern]
```

Capability allocator.

This is the instance of the capability allocator that is used by usual applications.

The capability allocator uses the [Counting_cap_alloc](#), a reference-counting thread-safe capability allocator, that keeps a reference counter for each managed capability selector.

Examples

[examples/libs/l4re/c++/mem_alloc/ma+rm.cc](#), [examples/libs/l4re/c++/shared_ds/ds_clnt.cc](#), [examples/libs/l4re/c++/shared_ds/d](#)
and [examples/libs/l4re/streammap/client.cc](#).

Referenced by [L4Re::Util::Br_manager::alloc_buffer_demand\(\)](#), [L4Re::Util::Smart_count_cap< Unmap_flags >::copy\(\)](#), [L4Re::Util::Smart_cap_auto< Unmap_flags >::free\(\)](#), [L4Re::Util::Smart_count_cap< Unmap_flags >::free\(\)](#), [L4Re::Util::Event_t< Default_event_payload >::init\(\)](#), [L4Re::Util::Event_t< Default_event_payload >::init_poll\(\)](#), [make_ref_cap\(\)](#), [make_ref_del_cap\(\)](#), [make_shared_cap\(\)](#), [make_shared_del_cap\(\)](#), [make_unique_cap\(\)](#), [make_unique_del_cap\(\)](#), [L4Re::Util::Br_manager::realloc_rcv_cap\(\)](#), and [L4Re::Util::Object_registry::unregister_obj\(\)](#).

14.10.2.3 Kumem utilities

Collaboration diagram for Kumem utilities:



Functions

- `int L4Re::Util::kumem_alloc (l4_addr_t *mem, unsigned pages_order, L4::Cap< L4::Task > task=L4Re::Env::env() ->task(), L4::Cap< L4Re::Rm > rm=L4Re::Env::env() ->rm()) noexcept`
Allocate state area.

14.10.2.3.1 Detailed Description

14.10.2.3.2 Function Documentation

14.10.2.3.2.1 kumem_alloc()

```
int L4Re::Util::kumem_alloc (
    l4_addr_t * mem,
    unsigned pages_order,
    L4::Cap< L4::Task > task = L4Re::Env::env() ->task(),
    L4::Cap< L4Re::Rm > rm = L4Re::Env::env() ->rm())    [noexcept]
```

Allocate state area.

Parameters

out	<i>mem</i>	Pointer to memory that has been allocated.
	<i>pages_order</i>	Size to allocate, in log2 pages.
	<i>task</i>	Task to use for allocation.
	<i>rm</i>	Region manager to use for allocation.

Return values

0	for success
<0	error code on failure

Note

The amount of kernel-user memory that can be allocated at once is limited by the used kernel implementation. The minimum allocatable amount is one page. A portable implementation should not depend on allocations greater than 16KiB to succeed.

References [L4Re::Env::env\(\)](#).

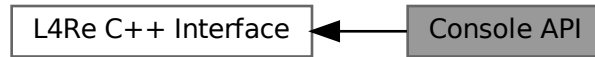
Here is the call graph for this function:



14.10.3 Console API

[Console](#) interface.

Collaboration diagram for Console API:



Data Structures

- class [L4Re::Console](#)
Console class.

14.10.3.1 Detailed Description

[Console](#) interface.

14.10.4 Debugging API

Debugging Interface.

Collaboration diagram for Debugging API:



Data Structures

- class [L4Re::Debug_obj](#)
Debug interface.

14.10.4.1 Detailed Description

Debugging Interface.

The debugging interface can be provided to retrieve, or log debugging information for an object. Each class may realize the debug interface to provide debugging functionality. For example, the region map objects provide a facility to dump the currently established memory regions.

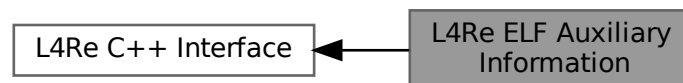
See also

L4::Debug_obj for more information.

14.10.5 L4Re ELF Auxiliary Information

API for embedding auxiliary information into binary programs.

Collaboration diagram for L4Re ELF Auxiliary Information:



Data Structures

- struct [l4re_elf_aux_t](#)
Generic header for each auxiliary vector element.
- struct [l4re_elf_aux_vma_t](#)
Auxiliary vector element for a reserved virtual memory area.
- struct [l4re_elf_aux_mword_t](#)
Auxiliary vector element for a single unsigned data word.

Macros

- #define [L4RE_ELF_AUX_ELEM](#) const __attribute__((used, section(".rol4re_elf_aux"), aligned(sizeof(l4_umword_t))))
Define an auxiliary vector element.
- #define [L4RE_ELF_AUX_ELEM_T](#)(type, id, tag, val...)
Define an auxiliary vector element.

Typedefs

- typedef struct l4re_elf_aux_t **l4re_elf_aux_t**
Generic header for each auxiliary vector element.
- typedef struct l4re_elf_aux_vma_t **l4re_elf_aux_vma_t**
Auxiliary vector element for a reserved virtual memory area.
- typedef struct l4re_elf_aux_mword_t **l4re_elf_aux_mword_t**
Auxiliary vector element for a single unsigned data word.

Enumerations

- enum {
[L4RE_ELF_AUX_T_NONE](#) = 0 , [L4RE_ELF_AUX_T_VMA](#) , [L4RE_ELF_AUX_T_STACK_SIZE](#) ,
[L4RE_ELF_AUX_T_STACK_ADDR](#) ,
[L4RE_ELF_AUX_T_KIP_ADDR](#) , [L4RE_ELF_AUX_T_EX_REGS_FLAGS](#) }

14.10.5.1 Detailed Description

API for embedding auxiliary information into binary programs.

This API allows information for the binary loader to be embedded into a binary application. This information can be reserved areas in the virtual memory of an application and things such as the stack size to be allocated for the first application thread.

14.10.5.2 Macro Definition Documentation

14.10.5.2.1 L4RE_ELF_AUX_ELEM

```
#define L4RE_ELF_AUX_ELEM const __attribute__((used, section(".ro14re_elf_aux"), aligned(sizeof(l4\_umword\_t))))
```

Define an auxiliary vector element.

This is the generic method for defining auxiliary vector elements. A more convenient way is to use [L4RE_ELF_AUX_ELEM_T](#).

Usage:

```
L4RE\_ELF\_AUX\_ELEM l4re\_elf\_aux\_vma\_t decl_name =  
{ L4RE\_ELF\_AUX\_T\_VMA, sizeof(l4re\_elf\_aux\_vma\_t), 0x2000, 0x4000 };
```

Definition at line 41 of file [elf_aux.h](#).

14.10.5.2.2 L4RE_ELF_AUX_ELEM_T

```
#define L4RE_ELF_AUX_ELEM_T(  
    type,  
    id,  
    tag,  
    val...)
```

Value:

```
static L4RE\_ELF\_AUX\_ELEM type id = {tag, sizeof(type), val}
```

Define an auxiliary vector element.

Parameters

<i>type</i>	is the data type for the element (e.g., l4re_elf_aux_vma_t)
<i>id</i>	is the identifier (variable name) for the declaration (the variable is defined with <code>static</code> storage class)
<i>tag</i>	is the tag value for the element e.g., L4RE_ELF_AUX_T_VMA

<i>val</i>	are the values to be set in the descriptor
------------	--

Usage:

```
L4RE_ELF_AUX_ELEM_T(l4re_elf_aux_vma_t, decl_name, L4RE_ELF_AUX_T_VMA, 0x2000, 0x4000 );
```

Definition at line 56 of file [elf_aux.h](#).

14.10.5.3 Enumeration Type Documentation

14.10.5.3.1 anonymous enum

anonymous enum

Enumerator

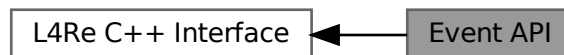
L4RE_ELF_AUX_T_NONE	Tag for an invalid element in the auxiliary vector.
L4RE_ELF_AUX_T_VMA	Tag for descriptor for a reserved virtual memory area.
L4RE_ELF_AUX_T_STACK_SIZE	Tag for descriptor that defines the stack size for the first application thread.
L4RE_ELF_AUX_T_STACK_ADDR	Tag for descriptor that defines the stack address for the first application thread.
L4RE_ELF_AUX_T_KIP_ADDR	Tag for descriptor that defines the KIP address for the binaries address space.
L4RE_ELF_AUX_T_EX_REGS_FLAGS	Tag for descriptor to override <code>ex_regs()</code> flags.

Definition at line 59 of file [elf_aux.h](#).

14.10.6 Event API

[Event API](#).

Collaboration diagram for Event API:



Data Structures

- class [L4Re::Event](#)
Event class.
- struct [L4Re::Default_event_payload](#)
Default event stream payload.
- class [L4Re::Event_buffer_t< PAYLOAD >](#)
Event buffer class.

14.10.6.1 Detailed Description

Event API.

On top of a shared [L4Re::Dataspace](#) (and optionally using [L4::Triggerable](#)), the event API implements asynchronous event transmission from an event provider (server) to an event receiver (client). Events are put into an [Event_buffer_t](#) residing on the shared [L4Re::Dataspace](#).

This interface is usually not used directly. Instead use [L4Re::Util::Event_t](#) for clients. An example server portion is implemented in [L4Re::Util::Event_svr](#).

This interface is usually used with [L4Re::Default_event_payload](#) which delivers HID events modeled on the Linux evdev API, and the interface's methods allow further querying of information about the HID event streams.

14.10.7 Auxiliary data

Collaboration diagram for Auxiliary data:



Data Structures

- struct [l4re_aux_t](#)
Auxiliary descriptor.

Typedefs

- typedef struct l4re_aux_t [l4re_aux_t](#)
Auxiliary descriptor.

Enumerations

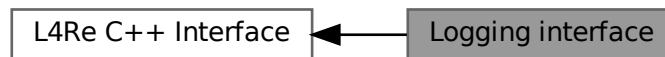
- enum [l4re_aux_ldr_flags_t](#)
Flags for program loading.

14.10.7.1 Detailed Description

14.10.8 Logging interface

Interface for log output.

Collaboration diagram for Logging interface:



Data Structures

- class [L4Re::Log](#)
Log interface class.

14.10.8.1 Detailed Description

Interface for log output.

The logging interface provides a facility sending log output. One purpose of the interface is to serialize the output and provide the possibility to tag output sent to a specific log object.

14.10.9 Name-space API

API for name spaces that store capabilities.

Collaboration diagram for Name-space API:



Data Structures

- class [L4Re::Namespace](#)
Name-space interface.

14.10.9.1 Detailed Description

API for name spaces that store capabilities.

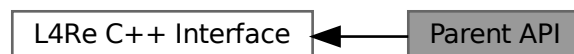
This is a basic abstraction for managing a mapping from human-readable names to capabilities. In particular, a name can also be mapped to a capability that refers to another name space object. By this means name spaces can be constructed hierarchically.

Name spaces play a central role in [L4Re](#), because the implementation of the name space objects determines the policy which capabilities (which objects) are accessible to a client of a name space.

14.10.10 Parent API

[Parent](#) interface.

Collaboration diagram for Parent API:



Data Structures

- class [L4Re::Parent](#)
[Parent](#) interface.

14.10.10.1 Detailed Description

[Parent](#) interface.

The parent interface provides means for an [L4](#) task to signal changes in its execution state. The main purpose is to signal program termination to the program that started it, so that its resources can be reclaimed. In a typical [L4Re](#) system, this program will be Moe or Ned.

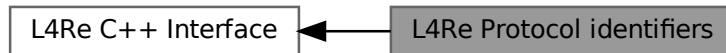
See also

[L4Re::Parent](#) for information about the concrete interface.

14.10.11 L4Re Protocol identifiers

Fix [L4Re](#) Protocol Constants.

Collaboration diagram for L4Re Protocol identifiers:



Enumerations

- enum [L4Re::Dataspace_::Opcodes](#)
Data-space communication-protocol opcodes.
- enum [L4Re::Event_::Opcodes](#)
Event communication-protocol opcodes.
- enum [L4Re::Inhibitor_::Opcodes](#)
Inhibitor communication-protocol opcodes.
- enum [L4Re::Log_::Opcodes](#)
Logging-service communication-protocol opcodes.
- enum [L4Re::Mem_alloc_::Opcodes](#)
Memory-allocator communication-protocol opcodes.
- enum [L4Re::Namespace_::Opcodes](#)
Name-space communication-protocol opcodes.
- enum [L4Re::Parent_::Opcodes](#)
Parent communication-protocol opcodes.
- enum [L4re_protocols](#) {
[L4RE_PROTO_DATASPACE](#) = 0x4000 , [L4RE_PROTO_NAMESPACE](#) , [L4RE_PROTO_PARENT](#) ,
[L4RE_PROTO_GOOS](#) ,
[L4RE_PROTO_RSVD_1](#) , [L4RE_PROTO_RM](#) , [L4RE_PROTO_EVENT](#) , [L4RE_PROTO_INHIBITOR](#) ,
[L4RE_PROTO_DMA_SPACE](#) , [L4RE_PROTO_MMIO_SPACE](#) , [L4RE_PROTO_ITAS](#) , [L4RE_PROTO_MEM_ALLOC](#)
 ,
[L4RE_PROTO_REMOTE_ACCESS](#) , [L4RE_PROTO_DEBUG](#) = ~0x7fffL }
Common [L4Re](#) Protocol Constants.
- enum [L4Re::Rm_::Opcodes](#)
Region-map communication-protocol opcodes.
- enum [L4Re::Video::Goos_::Opcodes](#)
Frame buffer communication-protocol opcodes.

14.10.11.1 Detailed Description

Fix [L4Re](#) Protocol Constants.

14.10.11.2 Enumeration Type Documentation

14.10.11.2.1 L4re_protocols

enum [L4re_protocols](#)

Common [L4Re](#) Protocol Constants.

Enumerator

L4RE_PROTO_DATASPACE	ID for L4Re::Dataspace RPCs.
L4RE_PROTO_NAMESPACE	ID for L4Re::Namespace RPCs.
L4RE_PROTO_PARENT	ID for L4Re::Parent RPCs.
L4RE_PROTO_GOOS	ID for L4Re::Video::Goos RPCs.
L4RE_PROTO_RSVD_1	Reserved ID.
L4RE_PROTO_RM	ID for L4Re::Rm RPCs.
L4RE_PROTO_EVENT	ID for L4Re::Event RPCs.
L4RE_PROTO_INHIBITOR	ID for L4Re::Inhibitor RPCs.
L4RE_PROTO_DMA_SPACE	ID for L4Re::Dma_space RPCs.
L4RE_PROTO_MMIO_SPACE	ID for L4Re::Mmio_space .
L4RE_PROTO_ITAS	ID for L4Re::Itas .
L4RE_PROTO_MEM_ALLOC	ID for L4Re::Mem_alloc .
L4RE_PROTO_REMOTE_ACCESS	ID for L4Re::Remote_access .
L4RE_PROTO_DEBUG	ID for debugging RPCs.

Definition at line 24 of file [protocols.h](#).

14.10.12 Region map API

Virtual address-space management.

Collaboration diagram for Region map API:



Data Structures

- class [L4Re::Rm](#)
Region map.

14.10.12.1 Detailed Description

Virtual address-space management.

A region map object implements two protocols. The first protocol is the kernel page-fault protocol, to resolve page faults for threads running in an [L4](#) task. The second protocol is the region map protocol itself, which allows managing the virtual memory address space of an [L4](#) task.

There are two basic concepts provided by the region map abstraction:

- **Areas** are reserved ranges in the virtual memory address space.
- **Regions** are ranges that are backed by (part of) a dataspace, i.e. accessing them results in access to the physical memory the dataspace manages.

Note that regions may live outside of areas and that an area does not necessarily contain any region.

Areas can be reserved for special use or for attaching a dataspace at a later time. When attaching a dataspace, the user can instruct the region map to search for an appropriate range to attach to. Regions are skipped in this search since they already have dataspace attached to them, and, depending on [L4Re::Rm::F::In_area](#), areas are skipped because they are reserved. Amongst others, areas can be used to attach several dataspace inside a certain range of addresses without interference from other threads.

When a region map receives a page fault IPC, the region map will check if the faulting virtual address lies in a region. If yes, it will answer the page fault IPC with a mapping from the backing dataspace. If not, an error is returned.

Depending on the system type, an attached dataspace might or might not be mapped eagerly. MMU-based systems resort to lazy mapping while systems without MMU do eager mappings by default. The [L4Re::Rm::F::Eager_map](#) and [L4Re::Rm::F::No_eager_map](#) flags can be used to force the respective behaviour, independent of the underlying system. In case both flags are given, the [L4Re::Rm::F::No_eager_map](#) flag wins.

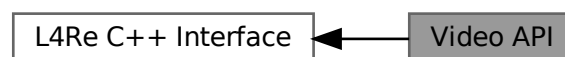
See also

[L4Re::Dataspace](#), [L4Re::Rm](#),
[Memory management - Data Spaces and the Region Map](#)

14.10.13 Video API

API for framebuffer based graphics.

Collaboration diagram for Video API:



Data Structures

- class [L4Re::Video::Color_component](#)
A color component.
- class [L4Re::Video::Pixel_info](#)
Pixel information.

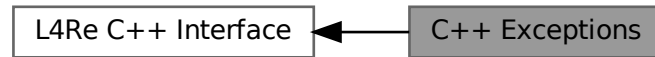
14.10.13.1 Detailed Description

API for framebuffer based graphics.

Contains the basic APIs that abstract framebuffers and views into them for [L4Re](#) applications.

14.10.14 C++ Exceptions

Collaboration diagram for C++ Exceptions:



Files

- file [exceptions](#)
Base exceptions.

Data Structures

- class [L4::Exception_tracer](#)
Back-trace support for exceptions.
- class [L4::Base_exception](#)
Base class for all exceptions, thrown by the [L4Re](#) framework.
- class [L4::Runtime_error](#)
Exception for an abstract runtime error.
- class [L4::Out_of_memory](#)
Exception signalling insufficient memory.
- class [L4::Element_already_exists](#)
Exception for duplicate element insertions.
- class [L4::Unknown_error](#)
Exception for an unknown condition.
- class [L4::Element_not_found](#)
Exception for a failed lookup (element not found).
- class [L4::Invalid_capability](#)
Indicates that an invalid object was invoked.
- class [L4::Com_error](#)
Error conditions during IPC.
- class [L4::Bounds_error](#)
Access out of bounds.

Macros

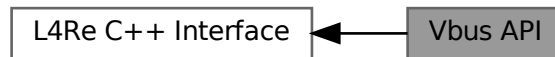
- `#define L4_CXX_EXCEPTION_BACKTRACE 20`
Number of instruction pointers in backtrace.

14.10.14.1 Detailed Description

14.10.15 Vbus API

C++ interface of the Vbus API.

Collaboration diagram for Vbus API:



Data Structures

- class [L4vbus::Pm< DEC >](#)
Power-management API mixin.
- class [L4vbus::Device](#)
Device on a L4vbus::Vbus.
- class [L4vbus::Icu](#)
Vbus Interrupt controller API.
- class [L4vbus::Vbus](#)
The virtual bus (Vbus) interface.
- class [L4vbus::Gpio_pin](#)
A GPIO pin.
- class [L4vbus::Gpio_module](#)
A Gpio_module groups multiple GPIO pins together.
- class [L4vbus::Pci_host_bridge](#)
A Pci host bridge.
- class [L4vbus::Pci_dev](#)
A PCI device.

14.10.15.1 Detailed Description

C++ interface of the Vbus API.

The virtual bus (Vbus) is a hierarchical (tree) structure of device nodes where each device has a set of resources attached to it. Each virtual bus provides an Icu ([Interrupt controller](#)) for interrupt handling.

The Vbus interface allows a client to find and query devices present on his virtual bus. After obtaining a device handle for a specific device the client can enumerate its resources.

Refer to [L4 Vbus functions](#) for the C API.

Include File

```
#include <l4/vbus/vbus>
```

Include File

```
#include <l4/vbus/vbus_gpio>
```

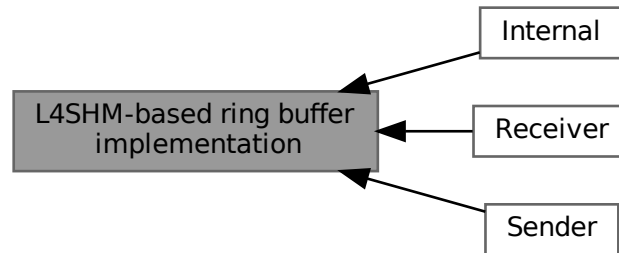
Include File

```
#include <l4/vbus/vbus_pci>
```


14.11 L4SHM-based ring buffer implementation

The library provides a non-locking (strictly 1:1) shared-memory-based ring buffer implementation based on the L4SHM library.

Collaboration diagram for L4SHM-based ring buffer implementation:



Topics

- Sender • ??
- Receiver ??
- Internal • ??

14.11.1 Detailed Description

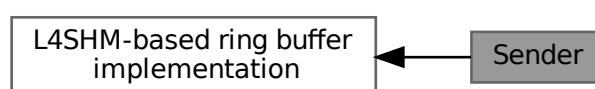
The library provides a non-locking (strictly 1:1) shared-memory-based ring buffer implementation based on the L4SHM library.

It requires an already allocated L4SHM area to be attached to sender and receiver. It will allocate an SHM chunk within this area and provides functions to produce data and consume data in FIFO order from the ring buffer.

The sender side of the buffer needs to be initialized *before* the receiver side, because allocation of the SHM chunk and the necessary signals is done on the sender side and the receiver initialization tries to attach to these objects.

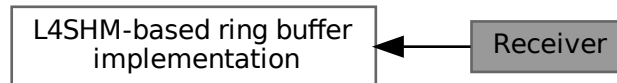
14.11.2 Sender

Collaboration diagram for Sender:



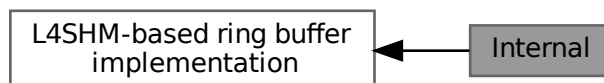
14.11.3 Receiver

Collaboration diagram for Receiver:



14.11.4 Internal

Collaboration diagram for Internal:



Data Structures

- struct [l4shmc_ringbuf_head_t](#)
Head field of a ring buffer.
- struct [l4shmc_ringbuf_t](#)
Ring buffer.

Macros

- #define [L4SHMC_RINGBUF_HEAD](#)(ringbuf)
Get ring buffer head pointer.
- #define [L4SHMC_RINGBUF_DATA](#)(ringbuf)
Get ring buffer data pointer.
- #define [L4SHMC_RINGBUF_DATA_SIZE](#)(ringbuf)
Get size of data area.

14.11.4.1 Detailed Description

14.11.4.2 Macro Definition Documentation

14.11.4.2.1 L4SHMC_RINGBUF_DATA

```
#define L4SHMC_RINGBUF_DATA(  
    ringbuf)
```

Value:

```
(L4SHMC_RINGBUF_HEAD(ringbuf)->data)
```

Get ring buffer data pointer.

Parameters

<i>ringbuf</i>	l4shmc_ringbuf_t struct
----------------	---

Definition at line 114 of file [ringbuf.h](#).

14.11.4.2.2 L4SHMC_RINGBUF_DATA_SIZE

```
#define L4SHMC_RINGBUF_DATA_SIZE(  
    ringbuf)
```

Value:

```
((ringbuf)->_size - sizeof(l4shmc_ringbuf_head_t))
```

Get size of data area.

Parameters

<i>ringbuf</i>	l4shmc_ringbuf_t struct
----------------	---

Definition at line 123 of file [ringbuf.h](#).

14.11.4.2.3 L4SHMC_RINGBUF_HEAD

```
#define L4SHMC_RINGBUF_HEAD(  
    ringbuf)
```

Value:

```
((l4shmc_ringbuf_head_t*) ((ringbuf)->_addr))
```

Get ring buffer head pointer.

Parameters

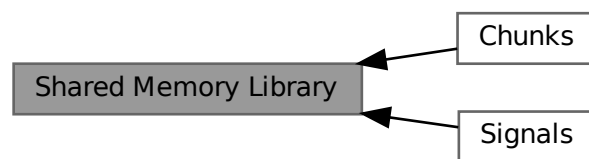
<i>ringbuf</i>	l4shmc_ringbuf_t struct
----------------	---

Definition at line 105 of file [ringbuf.h](#).

14.12 Shared Memory Library

L4SHM provides a shared memory infrastructure that establishes a shared memory area between multiple parties and uses a fast notification mechanism.

Collaboration diagram for Shared Memory Library:



Topics

- [Chunks](#) • ??
- [Signals](#) • ??

Functions

- [L4_BEGIN_DECLS](#) long [l4shmc_create](#) (char const *shmc_name)
Create a shared memory area.
- long [l4shmc_attach](#) (char const *shmc_name, l4shmc_area_t *shmarea)
Attach to a shared memory area.
- long [l4shmc_get_client_nr](#) (l4shmc_area_t const *shmarea)
Determine the client number of the shared memory region.
- long [l4shmc_mark_client_initialized](#) (l4shmc_area_t *shmarea)
Mark this shared memory client as 'initialized'.
- long [l4shmc_get_initialized_clients](#) (l4shmc_area_t *shmarea, l4_umword_t *bitmask)
Fetch the `_clients_init_done` bitmask of the shared memory area.
- long [l4shmc_connect_chunk_signal](#) (l4shmc_chunk_t *chunk, l4shmc_signal_t *signal)
Connect a signal with a chunk.
- long [l4shmc_area_size](#) (l4shmc_area_t const *shmarea)
Get size of shared memory area.
- long [l4shmc_area_size_free](#) (l4shmc_area_t const *shmarea)
Get free size of shared memory area.
- long [l4shmc_area_overhead](#) (void)
Get memory overhead per area that is not available for chunks.
- long [l4shmc_chunk_overhead](#) (void)
Get memory overhead required in addition to the chunk capacity for adding one chunk.

14.12.1 Detailed Description

L4SHM provides a shared memory infrastructure that establishes a shared memory area between multiple parties and uses a fast notification mechanism.

A shared memory area consists of chunks and signals. A chunk is a defined chunk of memory within the memory area with a maximum size. A chunk is filled (written) by a producer and read by a consumer. When a producer has finished writing to the chunk it signals a data ready notification to the consumer.

A consumer attaches to a chunk and waits for the producer to fill the chunk. After reading out the chunk it marks the chunk free again.

A shared memory area can have multiple chunks.

The interface is divided in three roles.

- The master role, responsible for setting up a shared memory area.
- A producer, generating data into a chunk
- A consumer, receiving data.

A signal can be connected with a chunk or can be used independently (e.g. for multiple chunks).

14.12.2 Function Documentation

14.12.2.1 `l4shm_area_overhead()`

```
long l4shm_area_overhead (
    void )
```

Get memory overhead per area that is not available for chunks.

Returns

Size of the overhead in bytes.

References [L4_CV](#).

14.12.2.2 `l4shm_area_size()`

```
long l4shm_area_size (
    l4shm_area_t const * shmarea)
```

Get size of shared memory area.

Parameters

<i>shmarea</i>	Shared memory area.
----------------	---------------------

Return values

>0	Size of the shared memory area.
<0	Error.

References [L4_CV](#).

14.12.2.3 l4shmc_area_size_free()

```
long l4shmc_area_size_free (
    l4shmc_area_t const * shmarea)
```

Get free size of shared memory area.

To get the max size to pass to `l4shmc_add_chunk`, subtract [l4shmc_chunk_overhead\(\)](#).

Parameters

<i>shmarea</i>	Shared memory area.
----------------	---------------------

Returns

Size of the shared memory area.

References [L4_CV](#).

14.12.2.4 l4shmc_attach()

```
long l4shmc_attach (
    char const * shm_name,
    l4shmc_area_t * shmarea)
```

Attach to a shared memory area.

Parameters

	<i>shm_name</i>	Name of the shared memory area.
out	<i>shmarea</i>	Pointer to shared memory area descriptor to be filled with information for the shared memory area.

On success, the data in 'shmarea' contains a client number which can be used to mutual agree about client initialization:

- [l4shmc_get_client_nr\(\)](#) returns the client number stored in 'shmarea'. The first attached client will get 0 and this number is increased for each attached client.
- [l4shmc_mark_client_initialized\(\)](#) tells other clients that this client has finished its initialization.
- [l4shmc_get_initialized_clients\(\)](#) returns the bitmap of initialized clients attached to this shared memory.

Return values

0	Success.
<0	Error.

Examples

[examples/libs/shmc/prodcons.c](#).

References [L4_CV](#).

14.12.2.5 l4shmc_chunk_overhead()

```
long l4shmc_chunk_overhead (
    void )
```

Get memory overhead required in addition to the chunk capacity for adding one chunk.

Returns

Size of the overhead in bytes.

References [L4_END_DECLS](#).

14.12.2.6 l4shmc_connect_chunk_signal()

```
long l4shmc_connect_chunk_signal (
    l4shmc_chunk_t * chunk,
    l4shmc_signal_t * signal)
```

Connect a signal with a chunk.

Parameters

<i>chunk</i>	Chunk to attach the signal to.
<i>signal</i>	Signal to attach.

Return values

0	Success.
<0	Error.

Examples

[examples/libs/shmc/prodcons.c](#).

References [L4_CV](#), and [L4_INLINE](#).

14.12.2.7 l4shmc_create()

```
L4\_BEGIN\_DECLS long l4shmc_create (
    char const * shmc_name)
```

Create a shared memory area.

Parameters

<i>shmc_name</i>	Name of the shared memory area.
------------------	---------------------------------

Return values

0	Success.
-L4_ENOMEM	The requested size is too big.
-L4_ENOENT	No valid capability with the name of the shared memory area found.
<0	Errors from l4re_rm_attach or l4re_ns_register_obj_srv.

Examples

[examples/libs/shmc/prodcons.c](#).

References [L4_CV](#).

14.12.2.8 l4shmc_get_client_nr()

```
long l4shmc_get_client_nr (
    l4shmc_area_t const * shmarea)
```

Determine the client number of the shared memory region.

Parameters

<i>shmarea</i>	The shared memory area.
----------------	-------------------------

Returns

client number.

References [L4_CV](#).

14.12.2.9 l4shmc_get_initialized_clients()

```
long l4shmc_get_initialized_clients (
    l4shmc_area_t * shmarea,
    l4_umword_t * bitmask)
```

Fetch the `_clients_init_done` bitmask of the shared memory area.

Parameters

	<i>shmarea</i>	The shared memory area.
out	<i>bitmask</i>	The bitmask describing which clients are initialized.

Return values

0	Success.
<0	Error.

See also

[l4shmc_mark_client_initialized\(\)](#), [l4shmc_get_client_nr\(\)](#)

Examples

[examples/libs/shmc/prodcons.c](#).

References [L4_CV](#).

14.12.2.10 l4shmc_mark_client_initialized()

```
long l4shmc_mark_client_initialized (  
    l4shmc_area_t * shmarea)
```

Mark this shared memory client as 'initialized'.

The corresponding bit is set in the `_clients_init_done` bitmask. The bitmask can be fetched with [l4shmc_get_initialized_clients\(\)](#).

Parameters

<i>shmarea</i>	The shared memory area.
----------------	-------------------------

Return values

0	Success.
<0	Error.

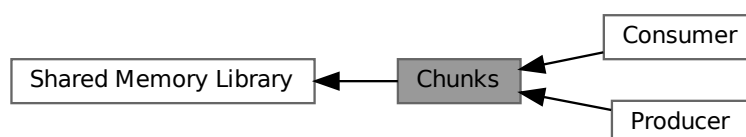
Examples

[examples/libs/shmc/prodcons.c](#).

References [L4_CV](#).

14.12.3 Chunks

Collaboration diagram for Chunks:



Topics

- Producer ??
- Consumer ??

Functions

- long [l4shmc_add_chunk](#) (l4shmc_area_t *shmarea, char const *chunk_name, [l4_umword_t](#) chunk_capacity, l4shmc_chunk_t *chunk)
Add a chunk in the shared memory area.
- long [l4shmc_get_chunk](#) (l4shmc_area_t *shmarea, char const *chunk_name, l4shmc_chunk_t *chunk)
Get chunk out of shared memory area.
- long [l4shmc_get_chunk_to](#) (l4shmc_area_t *shmarea, char const *chunk_name, [l4_umword_t](#) timeout_ms, l4shmc_chunk_t *chunk)
Get chunk out of shared memory area, with timeout.
- long [l4shmc_iterate_chunk](#) (l4shmc_area_t const *shmarea, char const **chunk_name, long offs)
Iterate over names of all existing chunks.
- void * [l4shmc_chunk_ptr](#) (l4shmc_chunk_t const *chunk)
Get data pointer to chunk.
- long [l4shmc_chunk_capacity](#) (l4shmc_chunk_t const *chunk)
Get capacity of a chunk.
- l4shmc_signal_t * [l4shmc_chunk_signal](#) (l4shmc_chunk_t const *chunk)
Get the registered signal of a chunk.

14.12.3.1 Detailed Description

14.12.3.2 Function Documentation

14.12.3.2.1 l4shmc_add_chunk()

```
long l4shmc_add_chunk (
    l4shmc_area_t * shmarea,
    char const * chunk_name,
    l4\_umword\_t chunk_capacity,
    l4shmc_chunk_t * chunk)
```

Add a chunk in the shared memory area.

Parameters

	<i>shmarea</i>	The shared memory area to put the chunk in.
	<i>chunk_name</i>	Name of the chunk.
	<i>chunk_capacity</i>	Capacity for payload of the chunk in bytes.
out	<i>chunk</i>	Chunk structure to fill in.

Return values

0	Success.
<0	Error.

Examples

[examples/libs/shmc/prodcons.c](#).

References [L4_CV](#).

14.12.3.2.2 l4shmc_chunk_capacity()

```
long l4shmc_chunk_capacity (  
    l4shmc_chunk_t const * chunk) [inline]
```

Get capacity of a chunk.

Parameters

<i>chunk</i>	Chunk.
--------------	--------

Returns

Capacity of the chunk in bytes.

References [L4_CV](#), and [L4_INLINE](#).

14.12.3.2.3 l4shmc_chunk_ptr()

```
void * l4shmc_chunk_ptr (  
    l4shmc_chunk_t const * chunk) [inline]
```

Get data pointer to chunk.

Parameters

<i>chunk</i>	Chunk.
--------------	--------

Returns

Chunk pointer.

Examples

[examples/libs/shmc/prodcons.c](#).

References [L4_CV](#), and [L4_INLINE](#).

14.12.3.2.4 l4shmc_chunk_signal()

```
l4shmc_signal_t * l4shmc_chunk_signal (  
    l4shmc_chunk_t const * chunk) [inline]
```

Get the registered signal of a chunk.

Parameters

<i>chunk</i>	Chunk.
--------------	--------

Return values

0	No signal has been registered with this chunk.
!=0	Pointer to signal otherwise.

References [L4_CV](#), and [L4_INLINE](#).

14.12.3.2.5 l4shmc_get_chunk()

```
long l4shmc_get_chunk (
    l4shmc_area_t * shmarea,
    char const * chunk_name,
    l4shmc_chunk_t * chunk) [inline]
```

Get chunk out of shared memory area.

Parameters

	<i>shmarea</i>	Shared memory area.
	<i>chunk_name</i>	Name of the chunk.
out	<i>chunk</i>	Chunk data structure to fill.

Return values

0	Success.
<0	Error.

Examples

[examples/libs/shmc/prodcons.c](#).

References [L4_CV](#).

14.12.3.2.6 l4shmc_get_chunk_to()

```
long l4shmc_get_chunk_to (
    l4shmc_area_t * shmarea,
    char const * chunk_name,
    l4_umword_t timeout_ms,
    l4shmc_chunk_t * chunk)
```

Get chunk out of shared memory area, with timeout.

Parameters

	<i>shmarea</i>	Shared memory area.
	<i>chunk_name</i>	Name of the chunk.
	<i>timeout_ms</i>	Timeout in milliseconds to wait for the chunk to appear in the shared memory area.
out	<i>chunk</i>	Chunk data structure to fill.

Return values

0	Success.
<0	Error.

References [L4_CV](#).

14.12.3.2.7 l4shmc_iterate_chunk()

```
long l4shmc_iterate_chunk (
    l4shmc_area_t const * shmarea,
    char const ** chunk_name,
    long offs)
```

Iterate over names of all existing chunks.

Parameters

<i>shmarea</i>	Shared memory area.
<i>chunk_name</i>	Where the name of the current chunk will be stored
<i>offs</i>	0 to start iteration, return value of previous call to l4shmc_iterate_chunk() to get next chunk

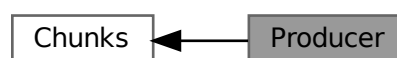
Return values

0	No more chunks available.
<0	Error.
>0	Iterator value for the next call.

References [L4_CV](#).

14.12.3.3 Producer

Collaboration diagram for Producer:



Functions

- long [l4shmc_chunk_try_to_take](#) (l4shmc_chunk_t *chunk)
Try to mark chunk busy.
- long [l4shmc_chunk_try_to_take_for_writing](#) (l4shmc_chunk_t *chunk)
Try to mark chunk busy writing.
- long [l4shmc_chunk_try_to_take_for_overwriting](#) (l4shmc_chunk_t *chunk)
Try to mark the chunk busy writing after it was ready for reading.
- long [l4shmc_chunk_ready](#) (l4shmc_chunk_t *chunk, [l4_umword_t](#) size)
Mark chunk as filled (ready).
- long [l4shmc_chunk_ready_sig](#) (l4shmc_chunk_t *chunk, [l4_umword_t](#) size)
Mark chunk as filled (ready) and signal consumer.
- long [l4shmc_is_chunk_clear](#) (l4shmc_chunk_t const *chunk)
Check whether chunk is free.

14.12.3.3.1 Detailed Description

14.12.3.3.2 Function Documentation

14.12.3.3.2.1 l4shmc_chunk_ready()

```
long l4shmc_chunk_ready (
    l4shmc_chunk_t * chunk,
    l4_umword_t size) [inline]
```

Mark chunk as filled (ready).

Parameters

<i>chunk</i>	chunk.
<i>size</i>	Size of data in the chunk, in bytes.

Return values

0	Success.
<0	Error.

References [L4_CV](#), and [L4_INLINE](#).

14.12.3.3.2.2 l4shmc_chunk_ready_sig()

```
long l4shmc_chunk_ready_sig (
    l4shmc_chunk_t * chunk,
    l4_umword_t size) [inline]
```

Mark chunk as filled (ready) and signal consumer.

Parameters

<i>chunk</i>	chunk.
<i>size</i>	Size of data in the chunk, in bytes.

Return values

0	Success.
<0	Error.

Examples

[examples/libs/shmc/prodcons.c](#).

References [L4_CV](#), and [L4_INLINE](#).

14.12.3.3.2.3 l4shmc_chunk_try_to_take()

```
long l4shmc_chunk_try_to_take (
    l4shmc_chunk_t * chunk) [inline]
```

Try to mark chunk busy.

Parameters

<i>chunk</i>	chunk to mark.
--------------	----------------

Return values

0	Chunk could be taken.
<0	Chunk could not be taken, try again.

Examples

[examples/libs/shmc/prodcons.c](#).

References [L4_CV](#), and [L4_INLINE](#).

14.12.3.3.2.4 l4shmc_chunk_try_to_take_for_overwriting()

```
long l4shmc_chunk_try_to_take_for_overwriting (
    l4shmc_chunk_t * chunk) [inline]
```

Try to mark the chunk busy writing after it was ready for reading.

Parameters

<i>chunk</i>	chunk to mark busy writing.
--------------	-----------------------------

This function is used by the producer to overwrite a message if the consumer did not read the message within an expected time. This function can only be used if the consumer uses [l4shmc_chunk_try_to_take_for_reading\(\)](#) before reading the chunk.

Return values

0	Chunk could be taken and can be written.
<0	Chunk could not be taken, try again.

References [L4_CV](#), and [L4_INLINE](#).

14.12.3.3.2.5 l4shmc_chunk_try_to_take_for_writing()

```
long l4shmc_chunk_try_to_take_for_writing (
    l4shmc_chunk_t * chunk) [inline]
```

Try to mark chunk busy writing.

This function is actually an alias for [l4shmc_chunk_try_to_take\(\)](#).

Parameters

<i>chunk</i>	chunk to mark busy writing.
--------------	-----------------------------

Return values

0	Chunk could be taken and can be written.
<0	Chunk could not be taken, try again.

References [L4_CV](#), and [L4_INLINE](#).

14.12.3.3.2.6 l4shmc_is_chunk_clear()

```
long l4shmc_is_chunk_clear (
    l4shmc_chunk_t const * chunk) [inline]
```

Check whether chunk is free.

Parameters

<i>chunk</i>	Chunk to check.
--------------	-----------------

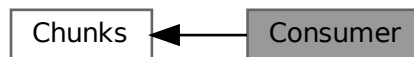
Return values

<code>!=0</code>	Chunk is clear.
<code>0</code>	Chunk is not clear.

References [L4_CV](#), and [L4_INLINE](#).

14.12.3.4 Consumer

Collaboration diagram for Consumer:



Functions

- [long l4shmc_chunk_try_to_take_for_reading](#) ([l4shmc_chunk_t](#) *chunk)
Try to mark chunk busy reading.
- [long l4shmc_enable_chunk](#) ([l4shmc_chunk_t](#) *chunk)
Enable a signal connected with a chunk.
- [long l4shmc_wait_chunk](#) ([l4shmc_chunk_t](#) *chunk)
Wait on a specific chunk.
- [long l4shmc_wait_chunk_to](#) ([l4shmc_chunk_t](#) *chunk, [l4_timeout_t](#) timeout)
Check whether a specific chunk has an event pending, with timeout.
- [long l4shmc_wait_chunk_try](#) ([l4shmc_chunk_t](#) *chunk)
Check whether a specific chunk has an event pending.
- [long l4shmc_chunk_consumed](#) ([l4shmc_chunk_t](#) *chunk)
Mark a chunk as free.
- [long l4shmc_is_chunk_ready](#) ([l4shmc_chunk_t](#) const *chunk)
Check whether data is available.
- [long l4shmc_chunk_size](#) ([l4shmc_chunk_t](#) const *chunk)
Get current size of a chunk.

14.12.3.4.1 Detailed Description

14.12.3.4.2 Function Documentation

14.12.3.4.2.1 l4shmc_chunk_consumed()

```
long l4shmc_chunk_consumed (
    l4shmc_chunk_t * chunk) [inline]
```

Mark a chunk as free.

Parameters

<i>chunk</i>	Chunk to mark as free.
--------------	------------------------

Return values

0	Success.
<0	Error.

Examples

[examples/libs/shmc/prodcons.c](#).

References [L4_CV](#).

14.12.3.4.2.2 l4shmc_chunk_size()

```
long l4shmc_chunk_size (
    l4shmc_chunk_t const * chunk) [inline]
```

Get current size of a chunk.

Parameters

<i>chunk</i>	Chunk.
--------------	--------

Returns

Current size of the chunk in bytes.

Examples

[examples/libs/shmc/prodcons.c](#).

References [L4_CV](#), and [L4_INLINE](#).

14.12.3.4.2.3 l4shmc_chunk_try_to_take_for_reading()

```
long l4shmc_chunk_try_to_take_for_reading (
    l4shmc_chunk_t * chunk) [inline]
```

Try to mark chunk busy reading.

Parameters

<i>chunk</i>	chunk to mark busy reading.
--------------	-----------------------------

Return values

0	Chunk could be taken and can be read.
<0	Chunk could not be taken, try again.

References [L4_CV](#), and [L4_INLINE](#).

14.12.3.4.2.4 l4shmc_enable_chunk()

```
long l4shmc_enable_chunk (  
    l4shmc_chunk_t * chunk)
```

Enable a signal connected with a chunk.

Parameters

<i>chunk</i>	Chunk to enable.
--------------	------------------

Return values

0	Success.
<0	Error.

A signal must be enabled before waiting when the consumer waits on any signal. Enabling is not needed if the consumer waits for a specific signal or chunk.

References [L4_CV](#), and [L4_INLINE](#).

14.12.3.4.2.5 l4shmc_is_chunk_ready()

```
long l4shmc_is_chunk_ready (  
    l4shmc_chunk_t const * chunk) [inline]
```

Check whether data is available.

Parameters

<i>chunk</i>	Chunk to check.
--------------	-----------------

Return values

<i>!=0</i>	Data is available.
0	No data available.

References [L4_CV](#), and [L4_INLINE](#).

14.12.3.4.2.6 l4shmc_wait_chunk()

```
long l4shmc_wait_chunk (  
    l4shmc_chunk_t * chunk)    [inline]
```

Wait on a specific chunk.

Parameters

<i>chunk</i>	Chunk to wait for.
--------------	--------------------

Return values

0	Success.
<0	Error.

Examples

[examples/libs/shmc/prodcons.c](#).

References [L4_CV](#).

14.12.3.4.2.7 l4shmc_wait_chunk_to()

```
long l4shmc_wait_chunk_to (  
    l4shmc_chunk_t * chunk,  
    l4_timeout_t timeout)
```

Check whether a specific chunk has an event pending, with timeout.

Parameters

<i>chunk</i>	Chunk to check.
<i>timeout</i>	Timeout.

Return values

0	Success.
<0	Error.

The return code indicates whether an event was pending or not. Success means an event was pending, if an receive timeout error is returned no event was pending.

References [L4_CV](#), and [L4_INLINE](#).

14.12.3.4.2.8 l4shmc_wait_chunk_try()

```
long l4shmc_wait_chunk_try (  
    l4shmc_chunk_t * chunk) [inline]
```

Check whether a specific chunk has an event pending.

Parameters

<i>chunk</i>	Chunk to check.
--------------	-----------------

Return values

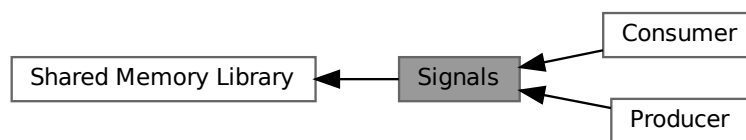
0	Success.
<0	Error.

The return code indicates whether an event was pending or not. Success means an event was pending, if an receive timeout error is returned no event was pending.

References [L4_CV](#), and [L4_INLINE](#).

14.12.4 Signals

Collaboration diagram for Signals:



Topics

- Producer ??
- Consumer ??

Functions

- long [l4shmc_add_signal](#) (l4shmc_area_t *shmarea, char const *signal_name, l4shmc_signal_t *signal)
Add a signal for the shared memory area.
- long [l4shmc_attach_signal](#) (l4shmc_area_t *shmarea, char const *signal_name, [l4_cap_idx_t](#) thread, l4shmc_signal_t *signal)
Attach to signal.
- long [l4shmc_get_signal](#) (l4shmc_area_t *shmarea, char const *signal_name, l4shmc_signal_t *signal)
Get signal object from the shared memory area.
- [l4_cap_idx_t](#) [l4shmc_signal_cap](#) (l4shmc_signal_t const *signal)
Get the signal capability of a signal.
- long [l4shmc_check_magic](#) (l4shmc_chunk_t const *chunk)
Check magic value of a chunk.

14.12.4.1 Detailed Description**14.12.4.2 Function Documentation****14.12.4.2.1 l4shmc_add_signal()**

```
long l4shmc_add_signal (
    l4shmc_area_t * shmarea,
    char const * signal_name,
    l4shmc_signal_t * signal)
```

Add a signal for the shared memory area.

Parameters

	<i>shmarea</i>	The shared memory area.
	<i>signal_name</i>	Name of the signal.
out	<i>signal</i>	Signal structure to fill in.

Return values

0	Success.
<0	Error.

Examples

[examples/libs/shmc/prodcons.c](#).

References [L4_CV](#), and [L4_INLINE](#).

14.12.4.2.2 l4shmc_attach_signal()

```
long l4shmc_attach_signal (
    l4shmc_area_t * shmarea,
    char const * signal_name,
    l4_cap_idx_t thread,
    l4shmc_signal_t * signal)
```

Attach to signal.

Parameters

	<i>shmarea</i>	Shared memory area.
	<i>signal_name</i>	Name of the signal.
	<i>thread</i>	Thread capability index to attach the signal to.
out	<i>signal</i>	Signal data structure to fill.

Return values

<i>0</i>	Success.
<i><0</i>	Error.

Examples

[examples/libs/shmc/prodcons.c](#).

References [L4_CV](#).

14.12.4.2.3 l4shmc_check_magic()

```
long l4shmc_check_magic (
    l4shmc_chunk_t const * chunk) [inline]
```

Check magic value of a chunk.

Parameters

<i>chunk</i>	Chunk.
--------------	--------

Return values

<i>0</i>	Magic value is not valid.
<i>>0</i>	Chunk is OK, the magic value is valid.

References [L4_CV](#).

14.12.4.2.4 l4shmc_get_signal()

```
long l4shmc_get_signal (
    l4shmc_area_t * shmarea,
    char const * signal_name,
    l4shmc_signal_t * signal)
```

Get signal object from the shared memory area.

Parameters

	<i>shmarea</i>	Shared memory area.
	<i>signal_name</i>	Name of the signal.
out	<i>signal</i>	Signal data structure to fill.

Return values

0	Success.
<0	Error.

References [L4_CV](#).

14.12.4.2.5 l4shmc_signal_cap()

```
l4_cap_idx_t l4shmc_signal_cap (
    l4shmc_signal_t const * signal) [inline]
```

Get the signal capability of a signal.

Parameters

<i>signal</i>	Signal.
---------------	---------

Returns

Capability of the signal object.

References [L4_CV](#), and [L4_INLINE](#).

14.12.4.3 Producer

Collaboration diagram for Producer:



Functions

- long [l4shmc_trigger](#) (l4shmc_signal_t *signal)
Trigger a signal.

14.12.4.3.1 Detailed Description

14.12.4.3.2 Function Documentation

14.12.4.3.2.1 l4shmc_trigger()

```
long l4shmc_trigger (
    l4shmc_signal_t * signal) [inline]
```

Trigger a signal.

Parameters

<i>signal</i>	Signal to trigger.
---------------	--------------------

Return values

0	Success.
<0	Error.

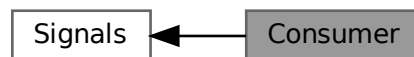
Examples

[examples/libs/shmc/prodcons.c](#).

References [L4_CV](#), and [L4_INLINE](#).

14.12.4.4 Consumer

Collaboration diagram for Consumer:



Functions

- long [l4shmc_enable_signal](#) (l4shmc_signal_t *signal)
Enable a signal.
- long [l4shmc_wait_any](#) (l4shmc_signal_t **retsignal)
Wait on any signal.
- long [l4shmc_wait_any_try](#) (l4shmc_signal_t **retsignal)
Check whether any waited signal has an event pending.
- long [l4shmc_wait_any_to](#) (l4_timeout_t timeout, l4shmc_signal_t **retsignal)
Wait for any signal with timeout.
- long [l4shmc_wait_signal](#) (l4shmc_signal_t *signal)
Wait on a specific signal.
- long [l4shmc_wait_signal_to](#) (l4shmc_signal_t *signal, l4_timeout_t timeout)
Wait on a specific signal, with timeout.
- long [l4shmc_wait_signal_try](#) (l4shmc_signal_t *signal)
Check whether a specific signal has an event pending.

14.12.4.4.1 Detailed Description

14.12.4.4.2 Function Documentation

14.12.4.4.2.1 l4shmc_enable_signal()

```
long l4shmc_enable_signal (  
    l4shmc_signal_t * signal)
```

Enable a signal.

Parameters

<i>signal</i>	Signal to enable.
---------------	-------------------

Return values

0	Success.
<0	Error.

A signal must be enabled before waiting when the consumer waits on any signal. Enabling is not needed if the consumer waits for a specific signal or chunk.

References [L4_CV](#).

14.12.4.4.2.2 l4shmc_wait_any()

```
long l4shmc_wait_any (
    l4shmc_signal_t ** retsignal) [inline]
```

Wait on any signal.

Parameters

out	<i>retsignal</i>	Signal received.
-----	------------------	------------------

Return values

0	Success.
<0	Error.

References [L4_CV](#), and [L4_INLINE](#).

14.12.4.4.2.3 l4shmc_wait_any_to()

```
long l4shmc_wait_any_to (
    l4_timeout_t timeout,
    l4shmc_signal_t ** retsignal)
```

Wait for any signal with timeout.

Parameters

	<i>timeout</i>	Timeout.
out	<i>retsignal</i>	Signal that has the event pending if any.

Return values

0	Success.
<0	Error.

The return code indicates whether an event was pending or not. Success means an event was pending, if an receive timeout error is returned no event was pending.

References [L4_CV](#), and [L4_INLINE](#).

14.12.4.4.2.4 l4shmc_wait_any_try()

```
long l4shmc_wait_any_try (  
    l4shmc_signal_t ** retsignal) [inline]
```

Check whether any waited signal has an event pending.

Parameters

out	<i>retsignal</i>	Signal that has the event pending if any.
-----	------------------	---

Return values

0	Success.
<0	Error.

The return code indicates whether an event was pending or not. Success means an event was pending, if an receive timeout error is returned no event was pending.

References [L4_CV](#).

14.12.4.4.2.5 l4shmc_wait_signal()

```
long l4shmc_wait_signal (  
    l4shmc_signal_t * signal) [inline]
```

Wait on a specific signal.

Parameters

<i>signal</i>	Signal to wait for.
---------------	---------------------

Return values

0	Success.
<0	Error.

Examples

[examples/libs/shmc/prodcons.c](#).

References [L4_CV](#).

14.12.4.4.2.6 l4shmc_wait_signal_to()

```
long l4shmc_wait_signal_to (  
    l4shmc_signal_t * signal,  
    l4_timeout_t timeout)
```

Wait on a specific signal, with timeout.

Parameters

<i>signal</i>	Signal to wait for.
<i>timeout</i>	Timeout.

Return values

0	Success.
<0	Error.

References [L4_CV](#), and [L4_INLINE](#).

14.12.4.4.2.7 l4shmc_wait_signal_try()

```
long l4shmc_wait_signal_try (  
    l4shmc_signal_t * signal) [inline]
```

Check whether a specific signal has an event pending.

Parameters

<i>signal</i>	Signal to check.
---------------	------------------

Return values

0	Success.
<0	Error.

The return code indicates whether an event was pending or not. Success means an event was pending, if an receive timeout error is returned no event was pending.

References [L4_CV](#), and [L4_INLINE](#).

14.13 Sigma0 API

Sigma0 API bindings.

Collaboration diagram for Sigma0 API:



Topics

- Internal constants ??
Internal sigma0 definitions.

Files

- file [sigma0.h](#)
Sigma0 interface.

Enumerations

- enum [l4sigma0_return_flags_t](#) {
[L4SIGMA0_OK](#) , [L4SIGMA0_NOTALIGNED](#) , [L4SIGMA0_IPCERROR](#) , [L4SIGMA0_NOFPAGE](#) ,
[L4SIGMA0_4](#) , [L4SIGMA0_5](#) , [L4SIGMA0_SMALLERFPAGE](#) }
Return flags of libsigma0 functions.

Functions

- [L4_BEGIN_DECLS](#) [l4_kernel_info_t](#) * [l4sigma0_map_kip](#) ([l4_cap_idx_t](#) sigma0, void *addr, unsigned log2↔_size)
Map the kernel info page from sigma0 to addr.
- int [l4sigma0_map_mem](#) ([l4_cap_idx_t](#) sigma0, [l4_addr_t](#) phys, [l4_addr_t](#) virt, [l4_addr_t](#) size)
Request a memory mapping from sigma0.
- int [l4sigma0_map_iomem](#) ([l4_cap_idx_t](#) sigma0, [l4_addr_t](#) phys, [l4_addr_t](#) virt, [l4_addr_t](#) size, int cached)
Request IO memory from sigma0.
- int [l4sigma0_map_anypage](#) ([l4_cap_idx_t](#) sigma0, [l4_addr_t](#) map_area, unsigned log2_map_size, [l4_addr_t](#) *base, unsigned sz)
Request an arbitrary free page of RAM.
- void [l4sigma0_debug_dump](#) ([l4_cap_idx_t](#) sigma0)
Request sigma0 to dump internal debug information.
- char const * [l4sigma0_map_errstr](#) (int err)
Get user readable error messages for the return codes.

14.13.1 Detailed Description

Sigma0 API bindings.

Convenience bindings for the Sigma0 protocol.

14.13.2 Enumeration Type Documentation

14.13.2.1 l4sigma0_return_flags_t

```
enum l4sigma0_return_flags_t
```

Return flags of libsigma0 functions.

Enumerator

L4SIGMA0_OK	Ok.
L4SIGMA0_NOTALIGNED	Phys, virt or size not aligned.
L4SIGMA0_IPCERROR	IPC error.
L4SIGMA0_NOFPAGE	No fpage received.
L4SIGMA0_SMALLERFPAGE	Superpage requested but smaller flexpage received.

Definition at line 74 of file [sigma0.h](#).

14.13.3 Function Documentation

14.13.3.1 l4sigma0_debug_dump()

```
void l4sigma0_debug_dump (
    l4_cap_idx_t sigma0)
```

Request sigma0 to dump internal debug information.

Parameters

<i>sigma0</i>	Capability selector for the sigma0 gate.
---------------	--

The debug information, such as internal memory maps, as well as statistics about the internal allocators is dumped to the kernel debugger.

References [L4_CV](#), and [L4_INLINE](#).

14.13.3.2 l4sigma0_map_anypage()

```
int l4sigma0_map_anypage (
    l4_cap_idx_t sigma0,
    l4_addr_t map_area,
    unsigned log2_map_size,
    l4_addr_t * base,
    unsigned sz)
```

Request an arbitrary free page of RAM.

Parameters

	<i>sigma0</i>	Capability selector for the sigma0 gate.
	<i>map_area</i>	The base address of the local virtual memory area where the page should be mapped.
	<i>log2_map_size</i>	The size of the requested page log 2 (the size in bytes is $2^{\log2_map_size}$). This must be at least the minimal page size. By specifying larger sizes the largest possible hardware page size will be used.
out	<i>base</i>	Physical address of the page received (i.e. the send base of the received mapping if any).
	<i>sz</i>	Size to map by the server in 2^{sz} bytes.

Return values

0	Success.
-L4SIGMA0_IPCERROR	IPC error.
-L4SIGMA0_NOFPAGE	No fpage received.

This function requests arbitrary free memory from sigma0. It should be used whenever spare memory is needed, instead of requesting specific physical memory with [l4sigma0_map_mem\(\)](#).

See [l4sigma0_map_errstr\(\)](#) to get a description of the return value.

References [L4_CV](#).

14.13.3.3 l4sigma0_map_errstr()

```
char const * l4sigma0_map_errstr (
    int err) [inline]
```

Get user readable error messages for the return codes.

Parameters

<i>err</i>	The error code reported by the <i>map</i> functions.
------------	--

Returns

A string containing the error message.

Definition at line 206 of file [sigma0.h](#).

References [L4_INLINE](#).

14.13.3.4 l4sigma0_map_iomem()

```
int l4sigma0_map_iomem (
    l4_cap_idx_t sigma0,
    l4_addr_t phys,
    l4_addr_t virt,
    l4_addr_t size,
    int cached)
```

Request IO memory from sigma0.

Parameters

<i>sigma0</i>	Capability selector for the sigma0 gate.
<i>phys</i>	The physical address to be requested (page aligned).
<i>virt</i>	The virtual address where the memory should be mapped to (page aligned).
<i>size</i>	The size of the IO memory area to be mapped (multiple of page size)
<i>cached</i>	Requests cacheable IO memory if 1 and uncached if 0.

Return values

0	Success.
-L4SIGMA0_NOTALIGNED	<i>phys</i> , <i>virt</i> , or <i>size</i> are not aligned.
-L4SIGMA0_IPCERROR	IPC error.
-L4SIGMA0_NOFPAGE	No fpage received.

This function is similar to [l4sigma0_map_mem\(\)](#), the difference is that it requests IO memory. IO memory is everything that is not known to be normal RAM. Also ACPI tables or the BIOS memory is treated as IO memory.

See [l4sigma0_map_errstr\(\)](#) to get a description of the return value.

References [L4_CV](#).

14.13.3.5 l4sigma0_map_kip()

```
L4_BEGIN_DECLS l4_kernel_info_t * l4sigma0_map_kip (
    l4_cap_idx_t sigma0,
    void * addr,
    unsigned log2_size)
```

Map the kernel info page from sigma0 to addr.

Parameters

<i>sigma0</i>	Capability selector for the sigma0 gate.
<i>addr</i>	Start of the receive window to receive KIP in.
<i>log2_size</i>	Size of the receive window to receive KIP in.

Returns

Address KIP was mapped to, 0 indicates an error.

14.13.3.6 l4sigma0_map_mem()

```
int l4sigma0_map_mem (
    l4_cap_idx_t sigma0,
    l4_addr_t phys,
    l4_addr_t virt,
    l4_addr_t size)
```

Request a memory mapping from sigma0.

Parameters

<i>sigma0</i>	Capability selector for the sigma0 gate.
<i>phys</i>	The physical address of the requested page (must be at least aligned to the minimum page size).
<i>virt</i>	The virtual address where the paged should be mapped in the local address space (must be at least aligned to the minimum page size).
<i>size</i>	The size of the requested page, this must be a multiple of the minimum page size.

Return values

<i>0</i>	Success.
<i>-L4SIGMA0_NOTALIGNED</i>	<i>phys</i> , <i>virt</i> , or <i>size</i> are not aligned.
<i>-L4SIGMA0_IPCERROR</i>	IPC error.
<i>-L4SIGMA0_NOFPAGE</i>	No fpage received.

This function only maps normal RAM. To map other memory, use [l4sigma0_map_iomem\(\)](#). See also there for the distinction between both memory types.

This is the direct method to request memory from sigma0. There is also the indirect method where sigma0 will answer page faults with a mapping that is one-to-one between the faulting virtual page and the backing physical page. See [L4::Pager::page_fault\(\)](#). For an overview of the memory hierarchy, see [Memory management - Data Spaces and the Region Map](#).

See [l4sigma0_map_errstr\(\)](#) to get a description of the return value.

References [L4_CV](#).

14.13.4 Internal constants

Internal sigma0 definitions.

Collaboration diagram for Internal constants:



Macros

- `#define SIGMA0_REQ_MAGIC ~0xFFUL`
Request magic.
- `#define SIGMA0_REQ_MASK ~0xFFUL`
Request mask.
- `#define SIGMA0_REQ_ID_MASK 0xF0`

- ID mask.*
- #define **SIGMA0_REQ_ID_FPAGE_RAM** 0x60
- RAM.*
- #define **SIGMA0_REQ_ID_FPAGE_IOMEM** 0x70
- I/O memory.*
- #define **SIGMA0_REQ_ID_FPAGE_IOMEM_CACHED** 0x80
- Cached I/O memory.*
- #define **SIGMA0_REQ_ID_FPAGE_ANY** 0x90
- Any.*
- #define **SIGMA0_REQ_ID_KIP** 0xA0
- KIP.*
- #define **SIGMA0_REQ_ID_DEBUG_DUMP** 0xC0
- Debug dump.*
- #define **SIGMA0_IS_MAGIC_REQ**(d1)
- Check if magic.*
- #define **SIGMA0_REQ**(x)
- Construct.*
- #define **SIGMA0_REQ_FPAGE_RAM** ([SIGMA0_REQ](#)(FPAGE_RAM))
- RAM.*
- #define **SIGMA0_REQ_FPAGE_IOMEM** ([SIGMA0_REQ](#)(FPAGE_IOMEM))
- I/O memory.*
- #define **SIGMA0_REQ_FPAGE_IOMEM_CACHED** ([SIGMA0_REQ](#)(FPAGE_IOMEM_CACHED))
- Cache I/O memory.*
- #define **SIGMA0_REQ_FPAGE_ANY** ([SIGMA0_REQ](#)(FPAGE_ANY))
- Any.*
- #define **SIGMA0_REQ_KIP** ([SIGMA0_REQ](#)(KIP))
- KIP.*
- #define **SIGMA0_REQ_DEBUG_DUMP** ([SIGMA0_REQ](#)(DEBUG_DUMP))
- Debug dump.*

14.13.4.1 Detailed Description

Internal sigma0 definitions.

14.14 Small C++ Template Library

Namespaces

- namespace [cxx](#)
- Our C++ library.*

Data Structures

- class [L4::Alloc_list](#)
A simple list-based allocator.
- class [cxx::List_item](#)
Basic list item.
- struct [cxx::Pair< First, Second >](#)
Pair of two values.
- class [cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >](#)
Basic slab allocator.
- class [cxx::Slab< Type, Slab_size, Max_free, Alloc >](#)
Slab allocator for object of type `Type`.
- class [cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >](#)
Merged slab allocator (allocators for objects of the same size are merged together).
- class [cxx::Slab_static< Type, Slab_size, Max_free, Alloc >](#)
Merged slab allocator (allocators for objects of the same size are merged together).
- class [cxx::Nothrow](#)
Helper type to distinguish the `operator new` version that does not throw exceptions.
- class [cxx::New_allocator< _Type >](#)
Standard allocator based on `operator new ()`.
- class [L4::String](#)
A null-terminated string container class.

Functions

- `template<typename A, typename ... ARGS>`
`constexpr A const & cxx::min (A const &a1, A const &a2, ARGS const &...a)`
Get the minimum of `a1` and `a2` up to `aN`.
- `template<typename A, typename ... ARGS>`
`constexpr A const & cxx::min (cxx::identity_t< A > const &a1, cxx::identity_t< A > const &a2, ARGS const &...a)`
Get the minimum of `a1` and `a2` up to `aN`.
- `template<typename A, typename ... ARGS>`
`constexpr A const & cxx::max (A const &a1, A const &a2, ARGS const &...a)`
Get the maximum of `a1` and `a2` up to `aN`.
- `template<typename A, typename ... ARGS>`
`constexpr A const & cxx::max (cxx::identity_t< A > const &a1, cxx::identity_t< A > const &a2, ARGS const &...a)`
Get the maximum of `a1` and `a2` up to `aN`.
- `template<typename T1>`
`T1 cxx::clamp (T1 v, T1 lo, T1 hi)`
Limit `v` to the range given by `lo` and `hi`.
- `void * operator new (size_t, void *mem, cxx::Nothrow const &) noexcept`
Simple placement new operator.
- `void * operator new (size_t, cxx::Nothrow const &) noexcept`
New operator that does not throw exceptions.
- `void operator delete (void *, cxx::Nothrow const &) noexcept`
Delete operator complementing the new operator not throwing exceptions.

14.14.1 Detailed Description

14.14.2 Function Documentation

14.14.2.1 clamp()

```
template<typename T1>
T1 cxx::clamp (
    T1 v,
    T1 lo,
    T1 hi) [inline]
```

Limit *v* to the range given by *lo* and *hi*.

Parameters

<i>v</i>	The value to clamp.
<i>lo</i>	The lower boundary to clamp <i>v</i> to.
<i>hi</i>	The upper boundary to clamp <i>v</i> to.

Definition at line 109 of file [minmax](#).

14.14.2.2 max() [1/2]

```
template<typename A, typename ... ARGS>
A const & cxx::max (
    A const & a1,
    A const & a2,
    ARGS const &... a) [constexpr]
```

Get the maximum of *a1* and *a2* up to *aN*.

Parameters

<i>a1</i>	The first value.
<i>a2</i>	The second value.
<i>...↔ a</i>	Arbitrary number of additional parameters.

Matches with automatic argument type deduction.

Definition at line 78 of file [minmax](#).

14.14.2.3 max() [2/2]

```
template<typename A, typename ... ARGS>
A const & cxx::max (
    cxx::identity_t< A > const & a1,
    cxx::identity_t< A > const & a2,
    ARGS const &... a) [constexpr]
```

Get the maximum of *a1* and *a2* up to *aN*.

Parameters

<i>a1</i>	The first value.
<i>a2</i>	The second value.
...↔ <i>a</i>	Arbitrary number of additional parameters.

Matches with explicit template type A.

Definition at line 93 of file [minmax](#).

14.14.2.4 min() [1/2]

```
template<typename A, typename ... ARGs>
A const & cxx::min (
    A const & a1,
    A const & a2,
    ARGs const &... a) [constexpr]
```

Get the minimum of a1 and a2 up to aN.

Parameters

<i>a1</i>	The first value.
<i>a2</i>	The second value.
...↔ <i>a</i>	Arbitrary number of additional parameters.

Matches with automatic argument type deduction.

Definition at line 36 of file [minmax](#).

14.14.2.5 min() [2/2]

```
template<typename A, typename ... ARGs>
A const & cxx::min (
    cxx::identity_t< A > const & a1,
    cxx::identity_t< A > const & a2,
    ARGs const &... a) [constexpr]
```

Get the minimum of a1 and a2 up to aN.

Parameters

<i>a1</i>	The first value.
<i>a2</i>	The second value.
...↔ <i>a</i>	Arbitrary number of additional parameters.

Matches with explicit template type A.

Definition at line 53 of file [minmax](#).

14.14.2.6 operator new()

```
void * operator new (
    size_t ,
    void * mem,
    cxx::Nothrow const & ) [inline], [noexcept]
```

Simple placement new operator.

Parameters

<i>mem</i>	the address of the memory block to place the new object.
------------	--

Returns

the address given by *mem*.

Definition at line 28 of file [std_alloc](#).

14.15 The L4Re IPC Framework

The mechanisms for IPC communication between [L4Re](#) applications.

Collaboration diagram for The L4Re IPC Framework:



Topics

- [Server-Side IPC framework](#) ??
Server-Side framework for implementing object-oriented servers.

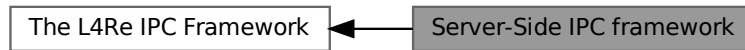
14.15.1 Detailed Description

The mechanisms for IPC communication between [L4Re](#) applications.

14.15.2 Server-Side IPC framework

Server-Side framework for implementing object-oriented servers.

Collaboration diagram for Server-Side IPC framework:



Namespaces

- namespace `L4::lpc_svr`
Helper classes for `L4::Server` instantiation.

Data Structures

- class `L4::lpc_svr::Server_iface`
Interface for server-loop related functions.
- class `L4::Basic_registry`
This registry returns the corresponding server object based on the label of an `lpc_gate`.
- struct `L4::lpc_svr::Ignore_errors`
Mix in for `LOOP_HOOKS` to ignore IPC errors.
- struct `L4::lpc_svr::Default_timeout`
Mix in for `LOOP_HOOKS` to use a 0 send and a infinite receive timeout.
- struct `L4::lpc_svr::Compound_reply`
Mix in for `LOOP_HOOKS` to always use compound reply and wait.
- struct `L4::lpc_svr::Default_setup_wait`
Mix in for `LOOP_HOOKS` for setup_wait no op.
- class `L4::lpc_svr::Br_manager_no_buffers`
Empty implementation of `Server_iface`.
- struct `L4::lpc_svr::Default_loop_hooks`
Default `LOOP_HOOKS`.
- class `L4::Server< LOOP_HOOKS >`
Basic server loop for handling client requests.
- class `L4::Server_object`
Abstract server object to be used with `L4::Server` and `L4::Basic_registry`.
- struct `L4::Server_object_t< IFACE, BASE >`
Base class (template) for server implementing server objects.
- struct `L4::Server_object_x< Derived, IFACE, BASE >`
Helper class to implement p_dispatch based server objects.
- struct `L4::lrq_handler_object`
Server object base class for handling IRQ messages.
- class `L4::lpc_svr::Timeout`
Callback interface for `Timeout_queue`.
- class `L4::lpc_svr::Timeout_queue`
Timeout queue to be used in l4re server loop.
- class `L4::lpc_svr::Timeout_queue_hooks< HOOKS, BR_MAN >`
Loop hooks mixin for integrating a timeout queue into the server loop.

Enumerations

- enum [L4::lpc_svr::Reply_mode](#) { [L4::lpc_svr::Reply_compound](#) , [L4::lpc_svr::Reply_separate](#) }
Reply mode for server loop.

14.15.2.1 Detailed Description

Server-Side framework for implementing object-oriented servers.

,

14.15.2.2 Enumeration Type Documentation

14.15.2.2.1 Reply_mode

enum [L4::Ipc_svr::Reply_mode](#)

Reply mode for server loop.

The reply mode specifies if the server loop shall do a compound reply and wait operation ([Reply_compound](#)), which is the most performant method. Note, `setup_wait()` is called before the reply. The other way is to call reply and wait separately and call `setup_wait` in between.

The actual mode is determined by the return value of the `before_reply()` hook in the `LOOP_HOOKS` of [L4::Server](#).

Enumerator

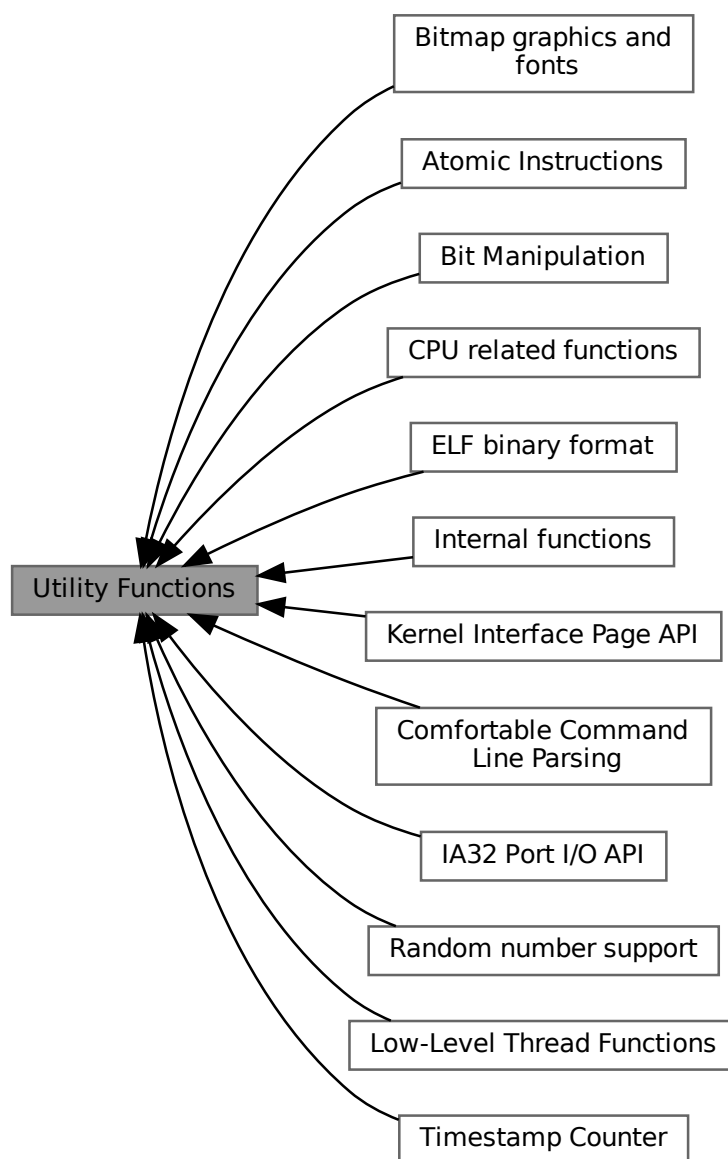
<code>Reply_compound</code>	Server shall use a compound reply and wait (fast).
<code>Reply_separate</code>	Server shall call reply and wait separately.

Definition at line 46 of file [ipc_server_loop](#).

14.16 Utility Functions

Utilities, generic file.

Collaboration diagram for Utility Functions:



Topics

- Bitmap graphics and fonts ??
This library provides some functions for bitmap handling in frame buffers.
- CPU related functions ??
- Timestamp Counter ??

- Atomic Instructions ??
- Internal functions ??
- Bit Manipulation ??
- ELF binary format ??
Functions and types related to ELF binaries.
- Kernel Interface Page API ??
- Comfortable Command Line Parsing ??
- Random number support ??
- Low-Level Thread Functions ??
- IA32 Port I/O API ??

Files

- file [rand.h](#)
Simple Pseudo-Random Number Generator.

Functions

- [L4_BEGIN_DECLS](#) long [l4util_splitlog2_hdl](#) ([l4_addr_t](#) start, [l4_addr_t](#) end, long(*handler)([l4_addr_t](#) s, [l4_addr_t](#) e, int log2size))
Split a range into log2 base and size aligned chunks.
- [l4_addr_t](#) [l4util_splitlog2_size](#) ([l4_addr_t](#) start, [l4_addr_t](#) end)
Return log2 base and size aligned length of a range.
- [L4_BEGIN_DECLS](#) [l4_timeout_s](#) [l4util_micros2l4to](#) ([l4_uint64_t](#) us) [L4_NOTHROW](#)
Calculate l4 timeouts.
- void [l4_sleep](#) ([l4_uint32_t](#) ms) [L4_NOTHROW](#)
Suspend thread for a period of ms milliseconds.
- void [l4_usleep](#) ([l4_uint64_t](#) us) [L4_NOTHROW](#)
Suspend thread for a period of us microseconds.
- void [l4_sleep_forever](#) (void) [L4_NOTHROW](#) [L4_NORETURN](#)
Go sleep and never wake up.
- void [l4_touch_ro](#) (const void *addr, unsigned size) [L4_NOTHROW](#)
Touch data area to force mapping (read-only).
- void [l4_touch_rw](#) (const void *addr, unsigned size) [L4_NOTHROW](#)
Touch data areas to force mapping (read-write).

14.16.1 Detailed Description

Utilities, generic file.

14.16.2 Function Documentation

14.16.2.1 l4_sleep()

```
void l4_sleep (
    l4_uint32_t ms)
```

Suspend thread for a period of *ms* milliseconds.

Parameters

<i>ms</i>	Time in milliseconds
-----------	----------------------

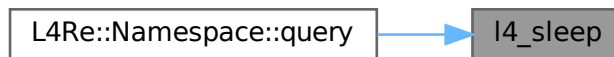
Examples

[examples/libs/libirq/async_isr.c](#), [examples/sys/aliens/main.c](#), [examples/sys/singlestep/main.c](#), and [examples/sys/start-with-exo/r](#)

References [L4_CV](#), and [L4_NOTHROW](#).

Referenced by [L4Re::Namespace::query\(\)](#).

Here is the caller graph for this function:



14.16.2.2 l4_touch_ro()

```
void l4_touch_ro (
    const void * addr,
    unsigned size) [inline]
```

Touch data area to force mapping (read-only).

Parameters

<i>addr</i>	Start of memory area to touch.
<i>size</i>	Size of area to touch.

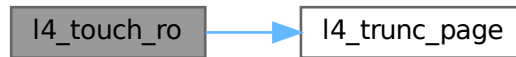
Examples

[examples/sys/singlestep/main.c](#).

Definition at line 92 of file [util.h](#).

References [L4_NOTHROW](#), [L4_PAGESIZE](#), and [l4_trunc_page\(\)](#).

Here is the call graph for this function:



14.16.2.3 l4_touch_rw()

```
void l4_touch_rw (
    const void * addr,
    unsigned size) [inline]
```

Touch data areas to force mapping (read-write).

Parameters

<i>addr</i>	Start of memory area to touch.
<i>size</i>	Size of area to touch.

Examples

[examples/sys/aliens/main.c](#), and [examples/sys/singlestep/main.c](#).

Definition at line 105 of file [util.h](#).

References [L4_NOTHROW](#), [L4_PAGESIZE](#), and [l4_trunc_page\(\)](#).

Here is the call graph for this function:



14.16.2.4 l4_usleep()

```
void l4_usleep (
    l4_uint64_t us)
```

Suspend thread for a period of *us* microseconds.

Parameters

<i>us</i>	Time in microseconds
-----------	----------------------

Note

The timer resolution of [L4](#) kernels is usually 1ms.

References [L4_CV](#), [L4_INLINE](#), [L4_NORETURN](#), and [L4_NOTHROW](#).

14.16.2.5 l4util_micros2l4to()

```
L4_BEGIN_DECLS l4_timeout_s l4util_micros2l4to (
    l4_uint64_t us)
```

Calculate l4 timeouts.

Parameters

<i>us</i>	time in microseconds. Special cases: <ul style="list-style-type: none"> • 0 -> timeout 0 • ~0U -> timeout NEVER
-----------	---

Returns

the corresponding l4_timeout value

Deprecated Use [l4_timeout_from_us\(\)](#).

References [L4_CV](#), and [L4_NOTHROW](#).

14.16.2.6 l4util_splitlog2_hdl()

```
L4_END_DECLS long l4util_splitlog2_hdl (
    l4_addr_t start,
    l4_addr_t end,
    long(* handler )(l4_addr_t s, l4_addr_t e, int log2size)) [inline]
```

Split a range into log2 base and size aligned chunks.

Parameters

<i>start</i>	Start of range
<i>end</i>	End of range (inclusive) (e.g. 2-4 is len 3)

<i>handler</i>	Handler function that is called with start and end (both inclusive) of the chunk. On success, the handler must return 0, if it returns !=0 the function will immediately return with the return code of the handler.
----------------	--

Returns

0 on success, != 0 otherwise

Definition at line 51 of file [splitlog2.h](#).

References [L4_EINVAL](#), and [l4util_splitlog2_size\(\)](#).

Here is the call graph for this function:



14.16.2.7 l4util_splitlog2_size()

```

l4_addr_t l4util_splitlog2_size (
    l4_addr_t start,
    l4_addr_t end) [inline]
  
```

Return log2 base and size aligned length of a range.

Parameters

<i>start</i>	Start of range
<i>end</i>	End of range (inclusive) (e.g. 2-4 is len 3)

Returns

length of elements in log2size (length is $1 \ll \log_2 \text{size}$)

Definition at line 70 of file [splitlog2.h](#).

Referenced by [l4util_splitlog2_hdl\(\)](#).

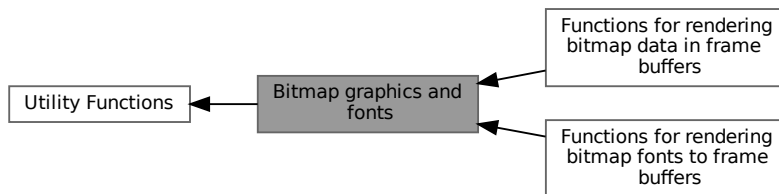
Here is the caller graph for this function:



14.16.3 Bitmap graphics and fonts

This library provides some functions for bitmap handling in frame buffers.

Collaboration diagram for Bitmap graphics and fonts:



Topics

- Functions for rendering bitmap data in frame buffers ??
- Functions for rendering bitmap fonts to frame buffers ??

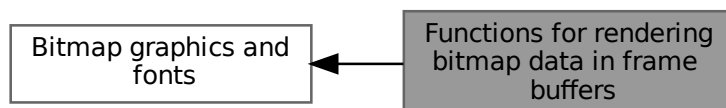
14.16.3.1 Detailed Description

This library provides some functions for bitmap handling in frame buffers.

Includes simple functions like filling or copying an area of the frame buffer going up to rendering text into the frame buffer using bitmap fonts.

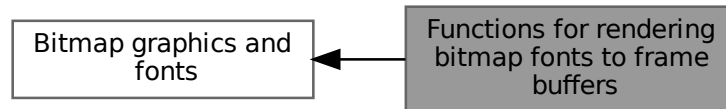
14.16.3.2 Functions for rendering bitmap data in frame buffers

Collaboration diagram for Functions for rendering bitmap data in frame buffers:



14.16.3.3 Functions for rendering bitmap fonts to frame buffers

Collaboration diagram for Functions for rendering bitmap fonts to frame buffers:



14.16.4 CPU related functions

Collaboration diagram for CPU related functions:



Functions

- int [l4util_cpu_has_cpuid](#) (void)
Check whether the CPU supports the "cpuid" instruction.
- unsigned int [l4util_cpu_capabilities](#) (void)
Returns the CPU capabilities if the "cpuid" instruction is available.
- unsigned int [l4util_cpu_capabilities_noccheck](#) (void)
Returns the CPU capabilities.
- void [l4util_cpu_cpuid](#) (unsigned long mode, unsigned long *eax, unsigned long *ebx, unsigned long *ecx, unsigned long *edx)
Generic CPUID access function.

14.16.4.1 Detailed Description

14.16.4.2 Function Documentation

14.16.4.2.1 l4util_cpu_capabilities()

```

unsigned int l4util_cpu_capabilities (
    void ) [inline]
  
```

Returns the CPU capabilities if the "cpuid" instruction is available.

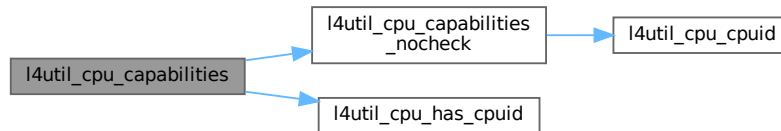
Returns

CPU capabilities if the "cpuid" instruction is available, 0 if the "cpuid" instruction is not supported.

Definition at line 95 of file [cpu.h](#).

References [l4util_cpu_capabilities_nocheck\(\)](#), and [l4util_cpu_has_cpuid\(\)](#).

Here is the call graph for this function:

**14.16.4.2.2 l4util_cpu_capabilities_nocheck()**

```
unsigned int l4util_cpu_capabilities_nocheck (
    void ) [inline]
```

Returns the CPU capabilities.

Returns

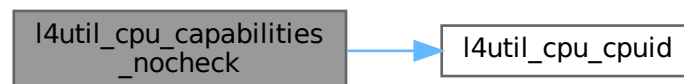
CPU capabilities.

Definition at line 84 of file [cpu.h](#).

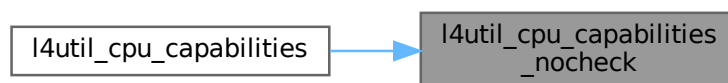
References [l4util_cpu_cpuid\(\)](#).

Referenced by [l4util_cpu_capabilities\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.16.4.2.3 l4util_cpu_has_cpuid()

```
int l4util_cpu_has_cpuid (
    void ) [inline]
```

Check whether the CPU supports the "cpuid" instruction.

Returns

1 if it has, 0 if it has not

Definition at line 64 of file [cpu.h](#).

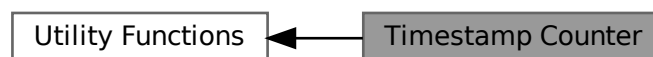
Referenced by [l4util_cpu_capabilities\(\)](#).

Here is the caller graph for this function:



14.16.5 Timestamp Counter

Collaboration diagram for Timestamp Counter:



Files

- file [rdtsc.h](#)
Timestamp counter related functions.
- file [rdtsc.h](#)
Timestamp counter related functions.

Functions

- [l4_cpu_time_t l4_rdtsc](#) (void)
Read current value of CPU-internal timestamp counter.
- [l4_uint32_t l4_rdtsc_32](#) (void)
Read the least significant 32 bit of the TSC.
- [l4_uint64_t l4_rdpmc](#) (int ecx)
Return current value of CPU-internal performance measurement counter.
- [l4_uint32_t l4_rdpmc_32](#) (int ecx)
Return the least significant 32 bit of a performance counter.
- [l4_uint64_t l4_tsc_to_ns](#) ([l4_cpu_time_t](#) tsc)
Convert timestamp to ns value.
- [l4_uint64_t l4_tsc_to_us](#) ([l4_cpu_time_t](#) tsc)
Convert timestamp into micro seconds value.
- void [l4_tsc_to_s_and_ns](#) ([l4_cpu_time_t](#) tsc, [l4_uint32_t](#) *s, [l4_uint32_t](#) *ns)
Convert timestamp to s.ns value.
- [l4_cpu_time_t l4_ns_to_tsc](#) ([l4_uint64_t](#) ns)
Convert nano seconds into CPU ticks.
- void [l4_busy_wait_ns](#) ([l4_uint64_t](#) ns)
Wait busy for a small amount of time.
- void [l4_busy_wait_us](#) ([l4_uint64_t](#) us)
Wait busy for a small amount of time.
- [l4_uint32_t l4_calibrate_tsc](#) ([l4_kernel_info_t](#) const *kip)
Determine scalers for timestamp calculations.
- [l4_uint32_t l4_tsc_init](#) ([l4_kernel_info_t](#) const *kip)
Initialize scaler for TSC calibrations from the kernel.
- [l4_uint32_t l4_get_hz](#) (void)
Get CPU frequency in Hz.

14.16.5.1 Detailed Description

14.16.5.2 Function Documentation

14.16.5.2.1 l4_busy_wait_ns()

```
void l4_busy_wait_ns (
    l4\_uint64\_t ns) [inline]
```

Wait busy for a small amount of time.

Parameters

<i>ns</i>	nano seconds to wait
-----------	----------------------

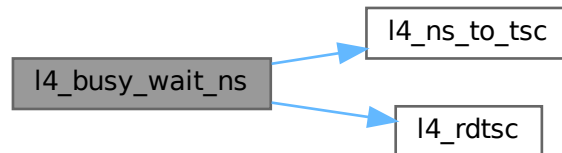
Attention

Not intended for any use!

Definition at line 262 of file [rdtsc.h](#).

References [l4_ns_to_tsc\(\)](#), and [l4_rdtsc\(\)](#).

Here is the call graph for this function:

**14.16.5.2.2 l4_busy_wait_us()**

```
void l4_busy_wait_us (
    l4_uint64_t us) [inline]
```

Wait busy for a small amount of time.

Parameters

<i>us</i>	micro seconds to wait
-----------	-----------------------

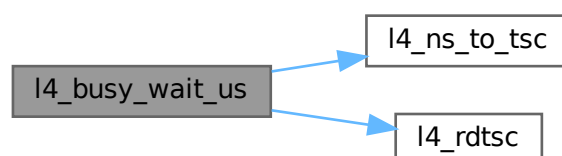
Attention

Not intended for any use!

Definition at line 272 of file [rdtsc.h](#).

References [l4_ns_to_tsc\(\)](#), and [l4_rdtsc\(\)](#).

Here is the call graph for this function:



14.16.5.2.3 l4_calibrate_tsc()

```
l4_uint32_t l4_calibrate_tsc (
    l4_kernel_info_t const * kip) [inline]
```

Determine scalars for timestamp calculations.

Determine some scalars to be able to convert between real time and CPU ticks. Just calls [l4_tsc_init\(\)](#).

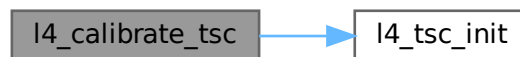
Examples

[examples/sys/aliens/main.c](#).

Definition at line 159 of file [rdtsc.h](#).

References [l4_tsc_init\(\)](#).

Here is the call graph for this function:



14.16.5.2.4 l4_get_hz()

```
l4_uint32_t l4_get_hz (
    void )
```

Get CPU frequency in Hz.

Returns

frequency in Hz

References [L4_END_DECLS](#), and [L4_INLINE](#).

14.16.5.2.5 l4_ns_to_tsc()

```
l4_cpu_time_t l4_ns_to_tsc (
    l4_uint64_t ns) [inline]
```

Convert nano seconds into CPU ticks.

Parameters

<i>ns</i>	nano seconds
-----------	--------------

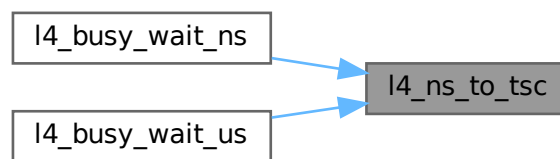
Returns

CPU ticks

Definition at line 248 of file [rdtsc.h](#).

Referenced by [l4_busy_wait_ns\(\)](#), and [l4_busy_wait_us\(\)](#).

Here is the caller graph for this function:

**14.16.5.2.6 l4_rdpmc()**

```
l4_uint64_t l4_rdpmc (
    int ecx) [inline]
```

Return current value of CPU-internal performance measurement counter.

Parameters

<i>ecx</i>	ECX value for the rdpmc instruction. For details see the Intel IA-32 Architectures Software Developer's Manual.
------------	---

Returns

64-bit PMC

Definition at line 175 of file [rdtsc.h](#).

14.16.5.2.7 l4_rdpmc_32()

```
l4_uint32_t l4_rdpmc_32 (
    int ecx) [inline]
```

Return the least significant 32 bit of a performance counter.

Useful for smaller differences, needs less cycles.

Definition at line 195 of file [rdtsc.h](#).

14.16.5.2.8 l4_rdtsc()

```
l4_cpu_time_t l4_rdtsc (
    void ) [inline]
```

Read current value of CPU-internal timestamp counter.

Returns

64-bit timestamp

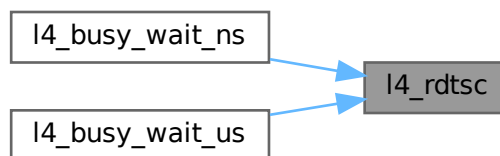
Examples

[examples/sys/aliens/main.c](#).

Definition at line 165 of file [rdtsc.h](#).

Referenced by [l4_busy_wait_ns\(\)](#), and [l4_busy_wait_us\(\)](#).

Here is the caller graph for this function:

**14.16.5.2.9 l4_rdtsc_32()**

```
l4_uint32_t l4_rdtsc_32 (
    void ) [inline]
```

Read the least significant 32 bit of the TSC.

Useful for smaller differences, needs less cycles.

Definition at line 185 of file [rdtsc.h](#).

14.16.5.2.10 l4_tsc_init()

```
l4_uint32_t l4_tsc_init (
    l4_kernel_info_t const * kip)
```

Initialize scaler for TSC calibrations from the kernel.

Initialize the scalers needed by [l4_tsc_to_ns\(\)](#)/[l4_ns_to_tsc\(\)](#) and so on. Use the kernel-provided frequency.

Parameters

<i>kip</i>	KIP pointer
------------	-------------

Returns

0 on error (no scalars exported by kernel) otherwise returns ($2^{32} / (\text{tsc per } \mu\text{sec})$). This value has the same semantics as the value returned by the `calibrate_delay_loop()` function of the Linux kernel.

References [L4_CV](#).

Referenced by [l4_calibrate_tsc\(\)](#).

Here is the caller graph for this function:



14.16.5.2.11 l4_tsc_to_ns()

```
l4_uint64_t l4_tsc_to_ns (
    l4_cpu_time_t tsc) [inline]
```

Convert timestamp to ns value.

Parameters

<i>tsc</i>	time value in CPU ticks
------------	-------------------------

Returns

time value in ns

Examples

[examples/sys/aliens/main.c](#).

Definition at line 205 of file [rdtsc.h](#).

14.16.5.2.12 l4_tsc_to_s_and_ns()

```
void l4_tsc_to_s_and_ns (
    l4_cpu_time_t tsc,
    l4_uint32_t * s,
    l4_uint32_t * ns) [inline]
```

Convert timestamp to s.ns value.

Parameters

	<i>tsc</i>	time value in CPU ticks
out	<i>s</i>	seconds
out	<i>ns</i>	nano seconds

Definition at line 233 of file [rdtsc.h](#).

14.16.5.2.13 l4_tsc_to_us()

```
l4_uint64_t l4_tsc_to_us (
    l4_cpu_time_t tsc) [inline]
```

Convert timestamp into micro seconds value.

Parameters

<i>tsc</i>	time value in CPU ticks
------------	-------------------------

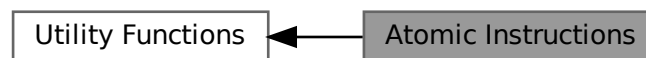
Returns

time value in micro seconds

Definition at line 219 of file [rdtsc.h](#).

14.16.6 Atomic Instructions

Collaboration diagram for Atomic Instructions:



Files

- file [atomic.h](#)
atomic operations header and generic implementations

Functions

- `int l4util_cmpxchg32` (volatile `l4_uint32_t` *dest, `l4_uint32_t` cmp_val, `l4_uint32_t` new_val)
Atomic compare and exchange (32 bit version).
- `int l4util_cmpxchg16` (volatile `l4_uint16_t` *dest, `l4_uint16_t` cmp_val, `l4_uint16_t` new_val)
Atomic compare and exchange (16 bit version).
- `int l4util_cmpxchg8` (volatile `l4_uint8_t` *dest, `l4_uint8_t` cmp_val, `l4_uint8_t` new_val)
Atomic compare and exchange (8 bit version).
- `int l4util_cmpxchg` (volatile `l4_umword_t` *dest, `l4_umword_t` cmp_val, `l4_umword_t` new_val)
Atomic compare and exchange (machine wide fields).
- `l4_uint32_t l4util_xchg32` (volatile `l4_uint32_t` *dest, `l4_uint32_t` val)
Atomic exchange (32 bit version).
- `l4_uint16_t l4util_xchg16` (volatile `l4_uint16_t` *dest, `l4_uint16_t` val)
Atomic exchange (16 bit version).
- `l4_uint8_t l4util_xchg8` (volatile `l4_uint8_t` *dest, `l4_uint8_t` val)
Atomic exchange (8 bit version).
- `l4_umword_t l4util_xchg` (volatile `l4_umword_t` *dest, `l4_umword_t` val)
Atomic exchange (machine wide fields).
- `void l4util_atomic_add` (volatile long *dest, long val)
Atomic add.
- `void l4util_atomic_inc` (volatile long *dest)
Atomic increment.

Atomic add/sub/and/or (8,16,32 bit version) without result

- `void l4util_add8` (volatile `l4_uint8_t` *dest, `l4_uint8_t` val)
- `void l4util_add16` (volatile `l4_uint16_t` *dest, `l4_uint16_t` val)
- `void l4util_add32` (volatile `l4_uint32_t` *dest, `l4_uint32_t` val)
- `void l4util_sub8` (volatile `l4_uint8_t` *dest, `l4_uint8_t` val)
- `void l4util_sub16` (volatile `l4_uint16_t` *dest, `l4_uint16_t` val)
- `void l4util_sub32` (volatile `l4_uint32_t` *dest, `l4_uint32_t` val)
- `void l4util_and8` (volatile `l4_uint8_t` *dest, `l4_uint8_t` val)
- `void l4util_and16` (volatile `l4_uint16_t` *dest, `l4_uint16_t` val)
- `void l4util_and32` (volatile `l4_uint32_t` *dest, `l4_uint32_t` val)
- `void l4util_or8` (volatile `l4_uint8_t` *dest, `l4_uint8_t` val)
- `void l4util_or16` (volatile `l4_uint16_t` *dest, `l4_uint16_t` val)
- `void l4util_or32` (volatile `l4_uint32_t` *dest, `l4_uint32_t` val)

Atomic add/sub/and/or operations (8,16,32 bit) with result

- `l4_uint8_t l4util_add8_res` (volatile `l4_uint8_t` *dest, `l4_uint8_t` val)
- `l4_uint16_t l4util_add16_res` (volatile `l4_uint16_t` *dest, `l4_uint16_t` val)
- `l4_uint32_t l4util_add32_res` (volatile `l4_uint32_t` *dest, `l4_uint32_t` val)
- `l4_uint8_t l4util_sub8_res` (volatile `l4_uint8_t` *dest, `l4_uint8_t` val)
- `l4_uint16_t l4util_sub16_res` (volatile `l4_uint16_t` *dest, `l4_uint16_t` val)
- `l4_uint32_t l4util_sub32_res` (volatile `l4_uint32_t` *dest, `l4_uint32_t` val)
- `l4_uint8_t l4util_and8_res` (volatile `l4_uint8_t` *dest, `l4_uint8_t` val)
- `l4_uint16_t l4util_and16_res` (volatile `l4_uint16_t` *dest, `l4_uint16_t` val)
- `l4_uint32_t l4util_and32_res` (volatile `l4_uint32_t` *dest, `l4_uint32_t` val)
- `l4_uint8_t l4util_or8_res` (volatile `l4_uint8_t` *dest, `l4_uint8_t` val)
- `l4_uint16_t l4util_or16_res` (volatile `l4_uint16_t` *dest, `l4_uint16_t` val)
- `l4_uint32_t l4util_or32_res` (volatile `l4_uint32_t` *dest, `l4_uint32_t` val)

Atomic inc/dec (8,16,32 bit) without result

- void [l4util_inc8](#) (volatile [l4_uint8_t](#) *dest)
- void [l4util_inc16](#) (volatile [l4_uint16_t](#) *dest)
- void [l4util_inc32](#) (volatile [l4_uint32_t](#) *dest)
- void [l4util_dec8](#) (volatile [l4_uint8_t](#) *dest)
- void [l4util_dec16](#) (volatile [l4_uint16_t](#) *dest)
- void [l4util_dec32](#) (volatile [l4_uint32_t](#) *dest)

Atomic inc/dec (8,16,32 bit) with result

- [l4_uint8_t](#) [l4util_inc8_res](#) (volatile [l4_uint8_t](#) *dest)
- [l4_uint16_t](#) [l4util_inc16_res](#) (volatile [l4_uint16_t](#) *dest)
- [l4_uint32_t](#) [l4util_inc32_res](#) (volatile [l4_uint32_t](#) *dest)
- [l4_uint8_t](#) [l4util_dec8_res](#) (volatile [l4_uint8_t](#) *dest)
- [l4_uint16_t](#) [l4util_dec16_res](#) (volatile [l4_uint16_t](#) *dest)
- [l4_uint32_t](#) [l4util_dec32_res](#) (volatile [l4_uint32_t](#) *dest)

14.16.6.1 Detailed Description**14.16.6.2 Function Documentation****14.16.6.2.1 l4util_add16()**

```
void l4util_add16 (  
    volatile l4\_uint16\_t * dest,  
    l4\_uint16\_t val)    [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Definition at line [472](#) of file [atomic.h](#).

14.16.6.2.2 l4util_add16_res()

```
l4\_uint16\_t l4util_add16_res (  
    volatile l4\_uint16\_t * dest,  
    l4\_uint16\_t val)    [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Returns

res

Definition at line [524](#) of file [atomic.h](#).

14.16.6.2.3 l4util_add32()

```
void l4util_add32 (  
    volatile l4_uint32_t * dest,  
    l4_uint32_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Definition at line 476 of file [atomic.h](#).

14.16.6.2.4 l4util_add32_res()

```
l4_uint32_t l4util_add32_res (  
    volatile l4_uint32_t * dest,  
    l4_uint32_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Returns

res

Definition at line 528 of file [atomic.h](#).

14.16.6.2.5 l4util_add8()

```
void l4util_add8 (  
    volatile l4_uint8_t * dest,  
    l4_uint8_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Definition at line 468 of file [atomic.h](#).

14.16.6.2.6 l4util_add8_res()

```
l4_uint8_t l4util_add8_res (  
    volatile l4_uint8_t * dest,  
    l4_uint8_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Returns

res

Definition at line 520 of file [atomic.h](#).

14.16.6.2.7 l4util_and16()

```
void l4util_and16 (  
    volatile l4_uint16_t * dest,  
    l4_uint16_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Definition at line 500 of file [atomic.h](#).

14.16.6.2.8 l4util_and16_res()

```
l4_uint16_t l4util_and16_res (  
    volatile l4_uint16_t * dest,  
    l4_uint16_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Returns

res

Definition at line 548 of file [atomic.h](#).

14.16.6.2.9 l4util_and32()

```
void l4util_and32 (  
    volatile l4_uint32_t * dest,  
    l4_uint32_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Definition at line 504 of file [atomic.h](#).

14.16.6.2.10 l4util_and32_res()

```
l4_uint32_t l4util_and32_res (  
    volatile l4_uint32_t * dest,  
    l4_uint32_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Returns

res

Definition at line 552 of file [atomic.h](#).

14.16.6.2.11 l4util_and8()

```
void l4util_and8 (  
    volatile l4_uint8_t * dest,  
    l4_uint8_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Definition at line 496 of file [atomic.h](#).

14.16.6.2.12 l4util_and8_res()

```
l4_uint8_t l4util_and8_res (  
    volatile l4_uint8_t * dest,  
    l4_uint8_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
-------------	---------------------

<i>val</i>	value to add/sub/and/or
------------	-------------------------

Returns

res

Definition at line 544 of file [atomic.h](#).

14.16.6.2.13 l4util_atomic_add()

```
void l4util_atomic_add (  
    volatile long * dest,  
    long val) [inline]
```

Atomic add.

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add

Definition at line 480 of file [atomic.h](#).

14.16.6.2.14 l4util_atomic_inc()

```
void l4util_atomic_inc (  
    volatile long * dest) [inline]
```

Atomic increment.

Parameters

<i>dest</i>	destination operand
-------------	---------------------

Definition at line 423 of file [atomic.h](#).

14.16.6.2.15 l4util_cmpxchg()

```
int l4util_cmpxchg (  
    volatile l4_umword_t * dest,  
    l4_umword_t cmp_val,  
    l4_umword_t new_val) [inline]
```

Atomic compare and exchange (machine wide fields).

Parameters

<i>dest</i>	destination operand
<i>cmp_val</i>	compare value
<i>new_val</i>	new value for dest

Returns

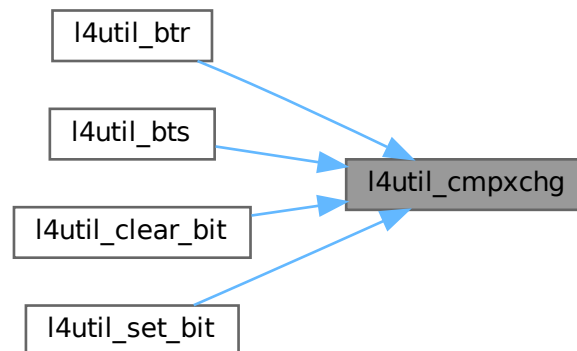
0 if comparison failed, 1 otherwise

Compare the value in *dest* with *cmp_val*, if equal set *dest* to *new_val*

Definition at line 379 of file [atomic.h](#).

Referenced by [l4util_btr\(\)](#), [l4util_bts\(\)](#), [l4util_clear_bit\(\)](#), and [l4util_set_bit\(\)](#).

Here is the caller graph for this function:



14.16.6.2.16 l4util_cmpxchg16()

```

int l4util_cmpxchg16 (
    volatile l4_uint16_t * dest,
    l4_uint16_t cmp_val,
    l4_uint16_t new_val) [inline]

```

Atomic compare and exchange (16 bit version).

Parameters

<i>dest</i>	destination operand
<i>cmp_val</i>	compare value
<i>new_val</i>	new value for dest

Returns

0 if comparison failed, !=0 otherwise

Compare the value in *dest* with *cmp_val*, if equal set *dest* to *new_val*

Definition at line 363 of file [atomic.h](#).

14.16.6.2.17 l4util_cmpxchg32()

```
L4_END_DECLS int l4util_cmpxchg32 (  
    volatile l4_uint32_t * dest,  
    l4_uint32_t cmp_val,  
    l4_uint32_t new_val) [inline]
```

Atomic compare and exchange (32 bit version).

Parameters

<i>dest</i>	destination operand
<i>cmp_val</i>	compare value
<i>new_val</i>	new value for dest

Returns

0 if comparison failed, !=0 otherwise

Compare the value in *dest* with *cmp_val*, if equal set *dest* to *new_val*

Definition at line 355 of file [atomic.h](#).

14.16.6.2.18 l4util_cmpxchg8()

```
int l4util_cmpxchg8 (  
    volatile l4_uint8_t * dest,  
    l4_uint8_t cmp_val,  
    l4_uint8_t new_val) [inline]
```

Atomic compare and exchange (8 bit version).

Parameters

<i>dest</i>	destination operand
<i>cmp_val</i>	compare value
<i>new_val</i>	new value for dest

Returns

0 if comparison failed, !=0 otherwise

Compare the value in *dest* with *cmp_val*, if equal set *dest* to *new_val*

Definition at line 371 of file [atomic.h](#).

14.16.6.2.19 l4util_dec16()

```
void l4util_dec16 (  
    volatile l4_uint16_t * dest) [inline]
```

Parameters

<i>dest</i>	destination operand
-------------	---------------------

Definition at line 431 of file [atomic.h](#).

14.16.6.2.20 l4util_dec16_res()

```
l4_uint16_t l4util_dec16_res (  
    volatile l4_uint16_t * dest) [inline]
```

Parameters

<i>dest</i>	destination operand
-------------	---------------------

Returns

res

Definition at line 456 of file [atomic.h](#).

14.16.6.2.21 l4util_dec32()

```
void l4util_dec32 (  
    volatile l4_uint32_t * dest) [inline]
```

Parameters

<i>dest</i>	destination operand
-------------	---------------------

Definition at line 435 of file [atomic.h](#).

14.16.6.2.22 l4util_dec32_res()

```
l4_uint32_t l4util_dec32_res (  
    volatile l4_uint32_t * dest) [inline]
```

Parameters

<i>dest</i>	destination operand
-------------	---------------------

Returns

res

Definition at line 460 of file [atomic.h](#).

14.16.6.2.23 l4util_dec8()

```
void l4util_dec8 (  
    volatile l4_uint8_t * dest) [inline]
```

Parameters

<i>dest</i>	destination operand
-------------	---------------------

Definition at line 427 of file [atomic.h](#).

14.16.6.2.24 l4util_dec8_res()

```
l4_uint8_t l4util_dec8_res (  
    volatile l4_uint8_t * dest) [inline]
```

Parameters

<i>dest</i>	destination operand
-------------	---------------------

Returns

res

Definition at line 452 of file [atomic.h](#).

14.16.6.2.25 l4util_inc16()

```
void l4util_inc16 (  
    volatile l4_uint16_t * dest) [inline]
```

Parameters

<i>dest</i>	destination operand
-------------	---------------------

Definition at line 415 of file [atomic.h](#).

14.16.6.2.26 l4util_inc16_res()

```
l4_uint16_t l4util_inc16_res (  
    volatile l4_uint16_t * dest) [inline]
```

Parameters

<i>dest</i>	destination operand
-------------	---------------------

Returns

res

Definition at line 444 of file [atomic.h](#).

14.16.6.2.27 l4util_inc32()

```
void l4util_inc32 (  
    volatile l4_uint32_t * dest) [inline]
```

Parameters

<i>dest</i>	destination operand
-------------	---------------------

Definition at line 419 of file [atomic.h](#).

14.16.6.2.28 l4util_inc32_res()

```
l4_uint32_t l4util_inc32_res (
    volatile l4_uint32_t * dest) [inline]
```

Parameters

<i>dest</i>	destination operand
-------------	---------------------

Returns

res

Definition at line 448 of file [atomic.h](#).

14.16.6.2.29 l4util_inc8()

```
void l4util_inc8 (
    volatile l4_uint8_t * dest) [inline]
```

Parameters

<i>dest</i>	destination operand
-------------	---------------------

Definition at line 411 of file [atomic.h](#).

14.16.6.2.30 l4util_inc8_res()

```
l4_uint8_t l4util_inc8_res (
    volatile l4_uint8_t * dest) [inline]
```

Parameters

<i>dest</i>	destination operand
-------------	---------------------

Returns

res

Definition at line 440 of file [atomic.h](#).

14.16.6.2.31 l4util_or16()

```
void l4util_or16 (
    volatile l4_uint16_t * dest,
    l4_uint16_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Definition at line 512 of file [atomic.h](#).

14.16.6.2.32 l4util_or16_res()

```
l4_uint16_t l4util_or16_res (  
    volatile l4_uint16_t * dest,  
    l4_uint16_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Returns

res

Definition at line 560 of file [atomic.h](#).

14.16.6.2.33 l4util_or32()

```
void l4util_or32 (  
    volatile l4_uint32_t * dest,  
    l4_uint32_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Definition at line 516 of file [atomic.h](#).

14.16.6.2.34 l4util_or32_res()

```
l4_uint32_t l4util_or32_res (  
    volatile l4_uint32_t * dest,  
    l4_uint32_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
-------------	---------------------

<i>val</i>	value to add/sub/and/or
------------	-------------------------

Returns

res

Definition at line 564 of file [atomic.h](#).

14.16.6.2.35 l4util_or8()

```
void l4util_or8 (  
    volatile l4_uint8_t * dest,  
    l4_uint8_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Definition at line 508 of file [atomic.h](#).

14.16.6.2.36 l4util_or8_res()

```
l4_uint8_t l4util_or8_res (  
    volatile l4_uint8_t * dest,  
    l4_uint8_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Returns

res

Definition at line 556 of file [atomic.h](#).

14.16.6.2.37 l4util_sub16()

```
void l4util_sub16 (  
    volatile l4_uint16_t * dest,  
    l4_uint16_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Definition at line 488 of file [atomic.h](#).

14.16.6.2.38 l4util_sub16_res()

```
l4_uint16_t l4util_sub16_res (  
    volatile l4_uint16_t * dest,  
    l4_uint16_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Returns

res

Definition at line 536 of file [atomic.h](#).

14.16.6.2.39 l4util_sub32()

```
void l4util_sub32 (  
    volatile l4_uint32_t * dest,  
    l4_uint32_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Definition at line 492 of file [atomic.h](#).

14.16.6.2.40 l4util_sub32_res()

```
l4_uint32_t l4util_sub32_res (  
    volatile l4_uint32_t * dest,  
    l4_uint32_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
-------------	---------------------

<i>val</i>	value to add/sub/and/or
------------	-------------------------

Returns

res

Definition at line 540 of file [atomic.h](#).

14.16.6.2.41 l4util_sub8()

```
void l4util_sub8 (  
    volatile l4_uint8_t * dest,  
    l4_uint8_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Definition at line 484 of file [atomic.h](#).

14.16.6.2.42 l4util_sub8_res()

```
l4_uint8_t l4util_sub8_res (  
    volatile l4_uint8_t * dest,  
    l4_uint8_t val) [inline]
```

Parameters

<i>dest</i>	destination operand
<i>val</i>	value to add/sub/and/or

Returns

res

Definition at line 532 of file [atomic.h](#).

14.16.6.2.43 l4util_xchg()

```
l4_umword_t l4util_xchg (  
    volatile l4_umword_t * dest,  
    l4_umword_t val) [inline]
```

Atomic exchange (machine wide fields).

Parameters

<i>dest</i>	destination operand
<i>val</i>	new value for dest

Returns

old value at destination

Definition at line 405 of file [atomic.h](#).

14.16.6.2.44 l4util_xchg16()

```
l4_uint16_t l4util_xchg16 (  
    volatile l4_uint16_t * dest,  
    l4_uint16_t val) [inline]
```

Atomic exchange (16 bit version).

Parameters

<i>dest</i>	destination operand
<i>val</i>	new value for dest

Returns

old value at destination

Definition at line 393 of file [atomic.h](#).

14.16.6.2.45 l4util_xchg32()

```
l4_uint32_t l4util_xchg32 (  
    volatile l4_uint32_t * dest,  
    l4_uint32_t val) [inline]
```

Atomic exchange (32 bit version).

Parameters

<i>dest</i>	destination operand
<i>val</i>	new value for dest

Returns

old value at destination

Definition at line 387 of file [atomic.h](#).

14.16.6.2.46 l4util_xchg8()

```
l4_uint8_t l4util_xchg8 (  
    volatile l4_uint8_t * dest,  
    l4_uint8_t val) [inline]
```

Atomic exchange (8 bit version).

Parameters

<i>dest</i>	destination operand
<i>val</i>	new value for dest

Returns

old value at destination

Definition at line 399 of file [atomic.h](#).

14.16.7 Internal functions

Collaboration diagram for Internal functions:



Functions

- void **base64_encode** (const char *infile, unsigned int in_size, char **outfile)
base-64-encode string infile
- void **base64_decode** (const char *infile, unsigned int in_size, char **outfile)
decode base-64-encoded string infile

14.16.7.1 Detailed Description

14.16.8 Bit Manipulation

Collaboration diagram for Bit Manipulation:



Files

- file [bitops_arch.h](#)
amd64 bit manipulation functions
- file [bitops.h](#)
bit manipulation functions
- file [bitops_arch.h](#)
x86 bit manipulation functions

Functions

- void [l4util_set_bit](#) (int b, volatile [l4_umword_t](#) *dest)
Set bit in memory.
- void [l4util_clear_bit](#) (int b, volatile [l4_umword_t](#) *dest)
Clear bit in memory.
- void [l4util_complement_bit](#) (int b, volatile [l4_umword_t](#) *dest)
Complement bit in memory.
- int [l4util_test_bit](#) (int b, const volatile [l4_umword_t](#) *dest)
Test bit (return value of bit).
- int [l4util_bts](#) (int b, volatile [l4_umword_t](#) *dest)
Bit test and set.
- int [l4util_btr](#) (int b, volatile [l4_umword_t](#) *dest)
Bit test and reset.
- int [l4util_btc](#) (int b, volatile [l4_umword_t](#) *dest)
Bit test and complement.
- int [l4util_bsr](#) ([l4_umword_t](#) word)
Bit scan reverse.
- int [l4util_bsf](#) ([l4_umword_t](#) word)
Bit scan forward.
- int [l4util_find_first_set_bit](#) (const void *dest, [l4_size_t](#) size)
Find the first set bit in a memory region.
- int [l4util_find_first_zero_bit](#) (const void *dest, [l4_size_t](#) size)
Find the first zero bit in a memory region.
- int [l4util_next_power2](#) (unsigned long val)
Find the next power of 2 for a given number.

14.16.8.1 Detailed Description

14.16.8.2 Function Documentation

14.16.8.2.1 l4util_bsf()

```
int l4util_bsf (
    l4\_umword\_t word) [inline]
```

Bit scan forward.

Parameters

<i>word</i>	value (machine size)
-------------	----------------------

Returns

index of least significant bit set in word, -1 if no bit is set (word == 0)

"bit scan forward", find least significant bit set in word.

Definition at line 316 of file [bitops.h](#).

14.16.8.2.2 l4util_bsr()

```
int l4util_bsr (  
    l4_umword_t word)    [inline]
```

Bit scan reverse.

Parameters

<i>word</i>	value (machine size)
-------------	----------------------

Returns

index of most significant set bit in word, -1 if no bit is set (word == 0)

"bit scan reverse", find most significant set bit in word (-> LOG2(word))

Definition at line 299 of file [bitops.h](#).

14.16.8.2.3 l4util_btc()

```
int l4util_btc (  
    int b,  
    volatile l4_umword_t * dest)    [inline]
```

Bit test and complement.

Parameters

<i>b</i>	bit position
<i>dest</i>	destination operand

Returns

Old value of bit *b*.

Complement bit *b* and return old value.

Definition at line 394 of file [bitops.h](#).

14.16.8.2.4 l4util_btr()

```
int l4util_btr (  
    int b,  
    volatile l4_umword_t * dest) [inline]
```

Bit test and reset.

Parameters

<i>b</i>	bit position
<i>dest</i>	destination operand

Returns

Old value of bit *b*.

Reset bit *b* and return old value.

Definition at line 278 of file [bitops.h](#).

References [l4util_cmpxchg\(\)](#).

Here is the call graph for this function:

**14.16.8.2.5 l4util_bts()**

```
int l4util_bts (  
    int b,  
    volatile l4_umword_t * dest) [inline]
```

Bit test and set.

Parameters

<i>b</i>	bit position
<i>dest</i>	destination operand

Returns

Old value of bit *b*.

Set the *b* bit of *dest* to 1 and return the old value.

Definition at line 256 of file [bitops.h](#).

References [l4util_cmpxchg\(\)](#).

Here is the call graph for this function:



14.16.8.2.6 l4util_clear_bit()

```
void l4util_clear_bit (  
    int b,  
    volatile l4_umword_t * dest) [inline]
```

Clear bit in memory.

Parameters

<i>b</i>	bit position
<i>dest</i>	destination operand

Definition at line 226 of file [bitops.h](#).

References [l4util_cmpxchg\(\)](#).

Here is the call graph for this function:



14.16.8.2.7 l4util_complement_bit()

```
void l4util_complement_bit (  
    int b,  
    volatile l4_umword_t * dest) [inline]
```

Complement bit in memory.

Parameters

<i>b</i>	bit position
<i>dest</i>	destination operand

Definition at line 359 of file [bitops.h](#).

14.16.8.2.8 l4util_find_first_set_bit()

```
int l4util_find_first_set_bit (  
    const void * dest,  
    l4_size_t size) [inline]
```

Find the first set bit in a memory region.

Parameters

<i>dest</i>	bit string
<i>size</i>	size of string in bits (must be a multiple of L4_MWORD_BITS!)

Returns

number of the first set bit, >= size if no bit is set

Definition at line 400 of file [bitops.h](#).

14.16.8.2.9 l4util_find_first_zero_bit()

```
int l4util_find_first_zero_bit (  
    const void * dest,  
    l4_size_t size) [inline]
```

Find the first zero bit in a memory region.

Parameters

<i>dest</i>	bit string
<i>size</i>	size of string in bits (must be a multiple of L4_MWORD_BITS!)

Returns

number of the first zero bit, >= size if no bit is set

Definition at line 333 of file [bitops.h](#).

14.16.8.2.10 l4util_next_power2()

```
int l4util_next_power2 (  
    unsigned long val) [inline]
```

Find the next power of 2 for a given number.

Parameters

<i>val</i>	initial value
------------	---------------

Returns

next-highest power of 2

Definition at line 373 of file [bitops.h](#).

14.16.8.2.11 l4util_set_bit()

```
void l4util_set_bit (
    int b,
    volatile l4_umword_t * dest) [inline]
```

Set bit in memory.

Parameters

<i>b</i>	bit position
<i>dest</i>	destination operand

Definition at line 207 of file [bitops.h](#).

References [l4util_cmpxchg\(\)](#).

Here is the call graph for this function:

**14.16.8.2.12 l4util_test_bit()**

```
int l4util_test_bit (
    int b,
    const volatile l4_umword_t * dest) [inline]
```

Test bit (return value of bit).

Parameters

<i>b</i>	bit position
<i>dest</i>	destination operand

Returns

Value of bit *b*.

Definition at line 244 of file [bitops.h](#).

14.16.9 ELF binary format

Functions and types related to ELF binaries.

Collaboration diagram for ELF binary format:



Files

- file [elf.h](#)
ELF definition.

Data Structures

- struct [Elf32_Ehdr](#)
ELF32 header.
- struct [Elf64_Ehdr](#)
ELF64 header.
- struct [Elf32_Shdr](#)
ELF32 section header.
- struct [Elf64_Shdr](#)
ELF64 section header.
- struct [Elf32_Phdr](#)
ELF32 program header.
- struct [Elf64_Phdr](#)
ELF64 program header.
- struct [Elf32_Dyn](#)
ELF32 dynamic entry.
- struct [Elf64_Dyn](#)
ELF64 dynamic entry.

- struct [Elf32_Rel](#)
ELF32 relocation entry w/o addend.
- struct [Elf32_Rela](#)
ELF32 relocation entry w/ addend.
- struct [Elf64_Rel](#)
ELF64 relocation entry w/o addend.
- struct [Elf64_Rela](#)
ELF64 relocation entry w/ addend.
- struct [Elf32_Sym](#)
ELF32 symbol table entry.
- struct [Elf64_Sym](#)
ELF64 symbol table entry.
- struct [Elf32_Auxv](#)
Auxiliary vector (32-bit).
- struct [Elf64_Auxv](#)
Auxiliary vector (64-bit).

Macros

- #define [ElfW](#)(type)
Use 64 or 32 bits types depending on the target architecture.
- #define [ELF32_R_SYM](#)(i)
Symbol table index.
- #define [ELF32_R_TYPE](#)(i)
- #define [ELF32_R_INFO](#)(s, t)
Create info from symbol table index + type.
- #define [ELF64_R_SYM](#)(i)
Symbol table index.
- #define [ELF64_R_TYPE](#)(i)
- #define [ELF64_R_INFO](#)(s, t)
Create info from symbol table index + type.
- #define [ELF32_ST_BIND](#)(i)
- #define [ELF32_ST_TYPE](#)(i)
- #define [ELF32_ST_INFO](#)(b, t)
Make info from bind + type.
- #define [ELF64_ST_BIND](#)(i)
- #define [ELF64_ST_TYPE](#)(i)
- #define [ELF64_ST_INFO](#)(b, t)
Make info from bind + type.

Typedefs

- typedef struct [Elf32_Auxv](#) [Elf32_Auxv](#)
Auxiliary vector (32-bit).
- typedef struct [Elf64_Auxv](#) [Elf64_Auxv](#)
Auxiliary vector (64-bit).

Enumerations

- enum { **EI_NIDENT** = 16 }
- enum **Elf_ETs** {
ET_NONE = 0 , **ET_REL** = 1 , **ET_EXEC** = 2 , **ET_DYN** = 3 ,
ET_CORE = 4 , **ET_LOPROC** = 0xff00 , **ET_HIPROC** = 0xffff }
- Object file type.*
- enum **Elf_EMs** {
EM_NONE = 0 , **EM_M32** = 1 , **EM_SPARC** = 2 , **EM_386** = 3 ,
EM_68K = 4 , **EM_88K** = 5 , **EM_860** = 7 , **EM_MIPS** = 8 ,
EM_MIPS_RS4_BE = 10 , **EM_SPARC64** = 11 , **EM_PARISC** = 15 , **EM_VPP500** = 17 ,
EM_SPARC32PLUS = 18 , **EM_960** = 19 , **EM_PPC** = 20 , **EM_V800** = 36 ,
EM_FR20 = 37 , **EM_RH32** = 38 , **EM_RCE** = 39 , **EM_ARM** = 40 ,
EM_ALPHA = 41 , **EM_SH** = 42 , **EM_SPARCV9** = 43 , **EM_TRICORE** = 44 ,
EM_ARC = 45 , **EM_H8_300** = 46 , **EM_H8_300H** = 47 , **EM_H8S** = 48 ,
EM_H8_500 = 49 , **EM_IA_64** = 50 , **EM_MIPS_X** = 51 , **EM_COLDFIRE** = 52 ,
EM_68HC12 = 53 , **EM_X86_64** = 62 , **EM_PDSP** = 63 , **EM_FX66** = 66 ,
EM_ST9PLUS = 67 , **EM_ST7** = 68 , **EM_68HC16** = 69 , **EM_68HC11** = 70 ,
EM_68HC08 = 71 , **EM_68HC05** = 72 , **EM_SVX** = 73 , **EM_ST19** = 74 ,
EM_VAX = 75 , **EM_CRIS** = 76 , **EM_JAVELIN** = 77 , **EM_FIREPATH** = 78 ,
EM_ZSP = 79 , **EM_MMIX** = 80 , **EM_HUANY** = 81 , **EM_PRISM** = 82 ,
EM_AVR = 83 , **EM_FR30** = 84 , **EM_D10V** = 85 , **EM_D30V** = 86 ,
EM_V850 = 87 , **EM_M32R** = 88 , **EM_MN10300** = 89 , **EM_MN10200** = 90 ,
EM_PJ = 91 , **EM_OPENRISC** = 92 , **EM_ARC_A5** = 93 , **EM_XTENSA** = 94 ,
EM_ALTERA_NIOS2 = 113 , **EM_AARCH64** = 183 , **EM_TILEPRO** = 188 , **EM_MICROBLAZE** = 189 ,
EM_TILEGX = 191 , **EM_RISCV** = 243 , **EM_NUM** = 244 }
- Required architecture.*
- enum **Elf_EVs** { **EV_NONE** = 0 , **EV_CURRENT** = 1 }
- Object file version.*
- enum **Elf_EIs** {
EI_MAG0 = 0 , **EI_MAG1** = 1 , **EI_MAG2** = 2 , **EI_MAG3** = 3 ,
EI_CLASS = 4 , **EI_DATA** = 5 , **EI_VERSION** = 6 , **EI_OSABI** = 7 ,
EI_ABIVERSION = 8 , **EI_PAD** = 9 }
- Identification Indices.*
- enum **Elf_MAGs** { **ELFMAG0** = 0x7f , **ELFMAG1** = 'E' , **ELFMAG2** = 'L' , **ELFMAG3** = 'F' }
- Magic number.*
- enum **Elf_CIASSs** { **ELFCLASSNONE** = 0 , **ELFCLASS32** = 1 , **ELFCLASS64** = 2 , **ELFCLASSNUM** = 3 }
- File class or capacity.*
- enum **Elf_DATAs** { **ELFDATANONE** = 0 , **ELFDATA2LSB** = 1 , **ELFDATA2MSB** = 2 , **ELFDATANUM** = 3 }
- Data encoding.*
- enum **Elf_OSABIs** {
ELFOSABI_NONE = 0 , **ELFOSABI_SYSV** = 0 , **ELFOSABI_HPUX** = 1 , **ELFOSABI_NETBSD** = 2 ,
ELFOSABI_LINUX = 3 , **ELFOSABI_SOLARIS** = 6 , **ELFOSABI_AIX** = 7 , **ELFOSABI_IRIX** = 8 ,
ELFOSABI_FREEBSD = 9 , **ELFOSABI_TRU64** = 10 , **ELFOSABI_MODESTO** = 11 , **ELFOSABI_OPENBSD**
= 12 ,
ELFOSABI_ARM = 97 , **ELFOSABI_STANDALONE** = 255 }
- Identify operating system and ABI to which the object is targeted.*
- enum **Elf_SHNs** {
SHN_UNDEF = 0 , **SHN_LORESERVE** = 0xff00 , **SHN_LOPROC** = 0xff00 , **SHN_HIPROC** = 0xff1f ,
SHN_ABS = 0xffff , **SHN_COMMON** = 0xffff2 , **SHN_HIRESERVE** = 0xffff }
- Special section indexes.*
- enum **Elf_SHTs** {
SHT_NULL = 0 , **SHT_PROGBITS** = 1 , **SHT_SYMTAB** = 2 , **SHT_STRTAB** = 3 ,
SHT_RELA = 4 , **SHT_HASH** = 5 , **SHT_DYNAMIC** = 6 , **SHT_NOTE** = 7 ,
SHT_NOBITS = 8 , **SHT_REL** = 9 , **SHT_SHLIB** = 10 , **SHT_DYNSYM** = 11 ,

SHT_INIT_ARRAY = 14 , SHT_FINI_ARRAY = 15 , SHT_PREINIT_ARRAY = 16 , SHT_GROUP = 17 ,
 SHT_SYMTAB_SHNDX = 18 , SHT_NUM = 19 , SHT_LOOS = 0x60000000 , SHT_HIOS = 0x6fffffff ,
 SHT_LOPROC = 0x70000000 , SHT_HIPROC = 0x7fffffff , SHT_LOUSER = 0x80000000 , SHT_HIUSER =
 0xffffffff }

Section type.

- enum **Elf_SHFs** {
 SHF_WRITE = 0x1 , SHF_ALLOC = 0x2 , SHF_EXECINSTR = 0x4 , SHF_MERGE = 0x10 ,
 SHF_STRINGS = 0x20 , SHF_INFO_LINK = 0x40 , SHF_OS_NONCONFORMING = 0x100 , SHF_GROUP
 = 0x200 ,
 SHF_TLS = 0x400 , SHF_MASKOS = 0x0ff00000 , SHF_MASKPROC = 0xf0000000 }

Section attribute flags.

- enum **Elf_PT**s {
 PT_NULL = 0 , PT_LOAD = 1 , PT_DYNAMIC = 2 , PT_INTERP = 3 ,
 PT_NOTE = 4 , PT_SHLIB = 5 , PT_PHDR = 6 , PT_TLS = 7 ,
 PT_NUM = 8 , PT_LOOS = 0x60000000 , PT_HIOS = 0x6fffffff , PT_LOPROC = 0x70000000 ,
 PT_HIPROC = 0x7fffffff , PT_GNU_EH_FRAME = PT_LOOS + 0x474e550 , PT_GNU_STACK = PT_LOOS
 + 0x474e551 , PT_GNU_RELRO = PT_LOOS + 0x474e552 ,
 PT_L4_STACK = PT_LOOS + 0x12 , PT_L4_AUX = PT_LOOS + 0x14 }

Segment types.

- enum **Elf_PF**s {
 PF_X = 0x1 , PF_W = 0x2 , PF_R = 0x4 , PF_MASKOS = 0x0ff00000 ,
 PF_MASKPROC = 0x7fffffff }

Segment permissions.

- enum **Elf_NT**s_core {
 NT_PRSTATUS = 1 , NT_FPREGSET = 2 , NT_PRPSINFO = 3 , NT_PRXREG = 4 ,
 NT_TASKSTRUCT = 4 , NT_PLATFORM = 5 , NT_AUXV = 6 , NT_GWINDOWS = 7 ,
 NT_ASRS = 8 , NT_PSTATUS = 10 , NT_PSINFO = 13 , NT_PRCRED = 14 ,
 NT_UTSNAME = 15 , NT_LWPSTATUS = 16 , NT_LWPSINFO = 17 , NT_PRFPXREG = 20 }

Legal values for note segment descriptor types for core files.

- enum **Elf_NT**s_obj { NT_VERSION = 1 }

Legal values for the note segment descriptor types for object files.

- enum **Elf_DT**s {
 DT_NULL = 0 , DT_NEEDED = 1 , DT_PLTRELSZ = 2 , DT_PLTGOT = 3 ,
 DT_HASH = 4 , DT_STRTAB = 5 , DT_SYMTAB = 6 , DT_RELA = 7 ,
 DT_RELASZ = 8 , DT_RELAENT = 9 , DT_STRSZ = 10 , DT_SYMENT = 11 ,
 DT_INIT = 12 , DT_FINI = 13 , DT_SONAME = 14 , DT_RPATH = 15 ,
 DT_SYMBOLIC = 16 , DT_REL = 17 , DT_RELSZ = 18 , DT_RELENT = 19 ,
 DT_PTRREL = 20 , DT_DEBUG = 21 , DT_TEXTREL = 22 , DT_JMPREL = 23 ,
 DT_BIND_NOW = 24 , DT_INIT_ARRAY = 25 , DT_FINI_ARRAY = 26 , DT_INIT_ARRAYSZ = 27 ,
 DT_FINI_ARRAYSZ = 28 , DT_RUNPATH = 29 , DT_FLAGS = 30 , DT_ENCODING = 32 ,
 DT_PREINIT_ARRAY = 32 , DT_PREINIT_ARRAYSZ = 33 , DT_NUM = 34 , DT_LOOS = 0x6000000d ,
 DT_HIOS = 0x6ffff000 , DT_LOPROC = 0x70000000 , DT_HIPROC = 0x7fffffff }

Dynamic Array Tags.

- enum **Elf_DF**s {
 DF_ORIGIN = 0x00000001 , DF_SYMBOLIC = 0x00000002 , DF_TEXTREL = 0x00000004 ,
 DF_BIND_NOW = 0x00000008 ,
 DF_STATIC_TLS = 0x00000010 }

Values of Elf32_Dyn.d_un.d_val, Elf64_Dyn.d_un.d_val in the DT_FLAGS entry.

- enum **Elf_DF_1**s {
 DF_1_NOW = 0x00000001 , DF_1_GLOBAL = 0x00000002 , DF_1_GROUP = 0x00000004 ,
 DF_1_NODELETE = 0x00000008 ,
 DF_1_LOADFLTR = 0x00000010 , DF_1_INITFIRST = 0x00000020 , DF_1_NOOPEN = 0x00000040 ,
 DF_1_ORIGIN = 0x00000080 ,
 DF_1_DIRECT = 0x00000100 , DF_1_TRANS = 0x00000200 , DF_1_INTERPOSE = 0x00000400 ,
 DF_1_NODEFLIB = 0x00000800 ,
 DF_1_NODUMP = 0x00001000 , DF_1_CONFALT = 0x00002000 , DF_1_ENDFILTEE = 0x00004000 ,


```
DF_1_DISPRELDNE = 0x00008000 ,
DF_1_DISPRELPND = 0x00010000 }
```

State flags selectable in the *Elf32_Dyn.d_un.d_val* / *Elf64_Dyn.d_un.d_val* element of the *DT_FLAGS_1* entry in the dynamic section.

- enum *Elf_DTF_1s*

Flags for the feature selection in *DT_FEATURE_1*.

- enum *Elf_DF_P1s* { *DF_P1_LAZYLOAD* = 0x00000001 , *DF_P1_GROUPPERM* = 0x00000002 }

Flags in the *DT_POSFLAG_1* entry effecting only the next *DT_** entry.

- enum *Elf_R_386_s* {
R_386_NONE = 0 , *R_386_32* = 1 , *R_386_PC32* = 2 , *R_386_GOT32* = 3 ,
R_386_PLT32 = 4 , *R_386_COPY* = 5 , *R_386_GLOB_DAT* = 6 , *R_386_JMP_SLOT* = 7 ,
R_386_RELATIVE = 8 , *R_386_GOTOFF* = 9 , *R_386_GOTPC* = 10 , *R_386_32PLT* = 11 ,
R_386_TLS_TPOFF = 14 , *R_386_TLS_IE* = 15 , *R_386_TLS_GOTIE* = 16 , *R_386_TLS_LE* = 17 ,
R_386_TLS_GD = 18 , *R_386_TLS_LDM* = 19 , *R_386_16* = 20 , *R_386_PC16* = 21 ,
R_386_8 = 22 , *R_386_PC8* = 23 , *R_386_TLS_GD_32* = 24 , *R_386_TLS_GD_PUSH* = 25 ,
R_386_TLS_GD_CALL = 26 , *R_386_TLS_GD_POP* = 27 , *R_386_TLS_LDM_32* = 28 , *R_386_TLS_LDM_PUSH*
= 29 ,
R_386_TLS_LDM_CALL = 30 , *R_386_TLS_LDM_POP* = 31 , *R_386_TLS_LDO_32* = 32 , *R_386_TLS_IE_32*
= 33 ,
R_386_TLS_LE_32 = 34 , *R_386_TLS_DTPMOD32* = 35 , *R_386_TLS_DTPOFF32* = 36 , *R_386_TLS_TPOFF32*
= 37 ,
R_386_NUM = 38 }

Relocation types (processor specific).

- enum *Elf_EF_ARM_s* { }

ARM specific declarations.

- enum *Elf_STT_ARM_s*

Additional symbol types for Thumb.

- enum *Elf_SHF_s_ARM* { *SHF_ARM_ENTRYSECT* = 0x10000000 , *SHF_ARM_COMDEF* = 0x80000000 }

ARM-specific values for *Elf32_Shdr.sh_flags* / *Elf64_Shdr.sh_flags*.

- enum *Elf_ARM_SBs* { *PF_ARM_SB* = 0x10000000 }

ARM-specific program header flags.

- enum *Elf_R_ARM_s* {
R_ARM_NONE = 0 , *R_ARM_PC24* = 1 , *R_ARM_ABS32* = 2 , *R_ARM_REL32* = 3 ,
R_ARM_PC13 = 4 , *R_ARM_ABS16* = 5 , *R_ARM_ABS12* = 6 , *R_ARM_THM_ABS5* = 7 ,
R_ARM_ABS8 = 8 , *R_ARM_SBREL32* = 9 , *R_ARM_THM_PC22* = 10 , *R_ARM_THM_PC8* = 11 ,
R_ARM_AMP_VCALL9 = 12 , *R_ARM_SWI24* = 13 , *R_ARM_THM_SWI8* = 14 , *R_ARM_XPC25* = 15 ,
R_ARM_THM_XPC22 = 16 , *R_ARM_COPY* = 20 , *R_ARM_GLOB_DAT* = 21 , *R_ARM_JUMP_SLOT* = 22 ,
R_ARM_RELATIVE = 23 , *R_ARM_GOTOFF* = 24 , *R_ARM_GOTPC* = 25 , *R_ARM_GOT32* = 26 ,
R_ARM_PLT32 = 27 , *R_ARM_ALU_PCREL_7_0* = 32 , *R_ARM_ALU_PCREL_15_8* = 33 , *R_ARM_↵*
ALU_PCREL_23_15 = 34 ,
R_ARM_LDR_SBREL_11_0 = 35 , *R_ARM_ALU_SBREL_19_12* = 36 , *R_ARM_ALU_SBREL_27_20* =
37 , *R_ARM_GNU_VTENTRY* = 100 ,
R_ARM_GNU_VTINHERIT = 101 , *R_ARM_THM_PC11* = 102 , *R_ARM_THM_PC9* = 103 , *R_ARM_↵*
RXPC25 = 249 ,
R_ARM_RSBREL32 = 250 , *R_ARM_THM_RPC22* = 251 , *R_ARM_RREL32* = 252 , *R_ARM_RABS22* =
253 ,
R_ARM_RPC24 = 254 , *R_ARM_RBASE* = 255 , *R_ARM_NUM* = 256 }

ARM relocations.

- enum *Elf_R_AARCH64_s* { *R_AARCH64_NONE* = 0 , *R_AARCH64_RELATIVE* = 1027 }

AARCH64 relocations.

- enum *Elf_R_X86_64_s* {
R_X86_64_NONE = 0 , *R_X86_64_64* = 1 , *R_X86_64_PC32* = 2 , *R_X86_64_GOT32* = 3 ,
R_X86_64_PLT32 = 4 , *R_X86_64_COPY* = 5 , *R_X86_64_GLOB_DAT* = 6 , *R_X86_64_JUMP_SLOT* = 7 ,
R_X86_64_RELATIVE = 8 , *R_X86_64_GOTPCREL* = 9 , *R_X86_64_32* = 10 , *R_X86_64_32S* = 11 ,
R_X86_64_16 = 12 , *R_X86_64_PC16* = 13 , *R_X86_64_8* = 14 , *R_X86_64_PC8* = 15 ,

```

R_X86_64_DTPMOD64 = 16 , R_X86_64_DTPOFF64 = 17 , R_X86_64_TPOFF64 = 18 , R_X86_64_TLSD
= 19 ,
R_X86_64_TLSD = 20 , R_X86_64_DTPOFF32 = 21 , R_X86_64_GOTTPOFF = 22 , R_X86_64_TPOFF32
= 23 ,
R_X86_64_NUM = 24 }

```

AMD x86-64 relocations.

- enum **Elf_STNs**

Symbol Table Entry.

- enum **Elf_STBs** {
STB_LOCAL = 0 , **STB_GLOBAL** = 1 , **STB_WEAK** = 2 , **STB_LOOS** = 10 ,
STB_HIOS = 12 , **STB_LOPROC** = 13 , **STB_HIPROC** = 15 }

Symbol Binding.

- enum **Elf_STTs** {
STT_NOTYPE = 0 , **STT_OBJECT** = 1 , **STT_FUNC** = 2 , **STT_SECTION** = 3 ,
STT_FILE = 4 , **STT_LOOS** = 10 , **STT_HIOS** = 12 , **STT_LOPROC** = 13 ,
STT_HIPROC = 15 }

Symbol Types.

- enum **Elf_ATs** {
AT_NULL = 0 , **AT_IGNORE** = 1 , **AT_EXECD** = 2 , **AT_PHDR** = 3 ,
AT_PHENT = 4 , **AT_PHNUM** = 5 , **AT_PAGESZ** = 6 , **AT_BASE** = 7 ,
AT_FLAGS = 8 , **AT_ENTRY** = 9 , **AT_NOTELF** = 10 , **AT_UID** = 11 ,
AT_EUID = 12 , **AT_GID** = 13 , **AT_EGID** = 14 , **AT_L4_AUX** = 0xf0 ,
AT_L4_ENV = 0xf1 }

Legal values for [Elf32_Auxv.atype](#) / [Elf64_Auxv.atype](#).

ELF types

- typedef **l4_uint32_t** **Elf32_Addr**
size 4 align 4
- typedef **l4_uint32_t** **Elf32_Off**
size 4 align 4
- typedef **l4_uint16_t** **Elf32_Half**
size 2 align 2
- typedef **l4_uint32_t** **Elf32_Word**
size 4 align 4
- typedef **l4_int32_t** **Elf32_Sword**
size 4 align 4
- typedef **l4_uint64_t** **Elf64_Addr**
size 8 align 8
- typedef **l4_uint64_t** **Elf64_Off**
size 8 align 8
- typedef **l4_uint16_t** **Elf64_Half**
size 2 align 2
- typedef **l4_uint32_t** **Elf64_Word**
size 4 align 4
- typedef **l4_int32_t** **Elf64_Sword**
size 4 align 4
- typedef **l4_uint64_t** **Elf64_Xword**
size 8 align 8
- typedef **l4_int64_t** **Elf64_Sxword**
size 8 align 8

14.16.9.1 Detailed Description

Functions and types related to ELF binaries.

14.16.9.2 Macro Definition Documentation

14.16.9.2.1 ELF32_R_TYPE

```
#define ELF32_R_TYPE(  
    i)
```

Value:

```
((unsigned char)(i))
```

See also

[Elf_R_386s](#).

Definition at line [663](#) of file [elf.h](#).

14.16.9.2.2 ELF32_ST_BIND

```
#define ELF32_ST_BIND(  
    i)
```

Value:

```
((i) >> 4)
```

See also

[Elf_STBs](#).

Definition at line [893](#) of file [elf.h](#).

14.16.9.2.3 ELF32_ST_TYPE

```
#define ELF32_ST_TYPE(  
    i)
```

Value:

```
((i) & 0xf)
```

See also

[Elf_STTs](#).

Definition at line [896](#) of file [elf.h](#).

14.16.9.2.4 ELF64_R_TYPE

```
#define ELF64_R_TYPE(  
    i)
```

Value:

```
((i) & 0xffffffffL)
```

See also

[Elf_R_386s](#).

Definition at line 671 of file [elf.h](#).

14.16.9.2.5 ELF64_ST_BIND

```
#define ELF64_ST_BIND(  
    i)
```

Value:

```
((i) >> 4)
```

See also

[Elf_STBs](#)

Definition at line 902 of file [elf.h](#).

14.16.9.2.6 ELF64_ST_TYPE

```
#define ELF64_ST_TYPE(  
    i)
```

Value:

```
((i) & 0xf)
```

See also

[Elf_STTs](#)

Definition at line 905 of file [elf.h](#).

14.16.9.3 Enumeration Type Documentation

14.16.9.3.1 anonymous enum

```
anonymous enum
```

Enumerator

EL_NIDENT	Number of characters.
-----------	-----------------------

Definition at line 117 of file [elf.h](#).

14.16.9.3.2 Elf_ARM_SBs

enum [Elf_ARM_SBs](#)

ARM-specific program header flags.

Enumerator

PF_ARM_SB	Segment contains the location addressed by the static base.
-----------	---

Definition at line 770 of file [elf.h](#).

14.16.9.3.3 Elf_ATs

enum [Elf_ATs](#)

Legal values for [Elf32_Auxv.atype](#) / [Elf64_Auxv.atype](#).

Enumerator

AT_NULL	End of vector.
AT_IGNORE	Entry should be ignored.
AT_EXECFD	File descriptor of program.
AT_PHDR	Program headers for program.
AT_PHEENT	Size of program header entry.
AT_PHNUM	Number of program headers.
AT_PAGESZ	System page size.
AT_BASE	Base address of interpreter.
AT_FLAGS	Flags.
AT_ENTRY	Entry point of program.
AT_NOTELF	Program is not ELF.
AT_UID	Real UID.
AT_EUID	Effective UID.
AT_GID	Real GID.
AT_EGID	Effective GID.
AT_L4_AUX	L4Re AUX section.
AT_L4_ENV	L4Re ENV section.

Definition at line 939 of file [elf.h](#).

14.16.9.3.4 Elf_CLASSES

enum `Elf_CLASSES`

File class or capacity.

Enumerator

ELFCLASSNONE	Invalid class.
ELFCLASS32	32-bit object
ELFCLASS64	64-bit object
ELFCLASSNUM	Mask for 32-bit or 64-bit class.

Definition at line 298 of file [elf.h](#).

14.16.9.3.5 Elf_DATAs

enum [Elf_DATAs](#)

Data encoding.

Enumerator

ELFDATANONE	invalid data encoding
ELFDATA2LSB	0x01020304 => [0x04 0x03 0x02 0x01]
ELFDATA2MSB	0x01020304 => [0x01 0x02 0x03 0x04]
ELFDATANUM	Mask for valid data encoding.

Definition at line 307 of file [elf.h](#).

14.16.9.3.6 Elf_DF_1s

enum [Elf_DF_1s](#)

State flags selectable in the Elf32_Dyn.d_un.d_val / Elf64_Dyn.d_un.d_val element of the DT_FLAGS_1 entry in the dynamic section.

Enumerator

DF_1_NOW	Set RTLD_NOW for this object.
DF_1_GLOBAL	Set RTLD_GLOBAL for this object.
DF_1_GROUP	Set RTLD_GROUP for this object.
DF_1_NODELETE	Set RTLD_NODELETE for this object.
DF_1_LOADFLTR	Trigger filtee loading at runtime.
DF_1_INITFIRST	Set RTLD_INITFIRST for this object.
DF_1_NOOPEN	Set RTLD_NOOPEN for this object.
DF_1_ORIGIN	\$ORIGIN must be handled.
DF_1_DIRECT	Direct binding enabled.
DF_1_INTERPOSE	Object is used to interpose.
DF_1_NODEFLIB	Ignore default lib search path.
DF_1_NODUMP	Object can't be dldump'ed.

DF_1_CONFALT	Configuration alternative created.
DF_1_ENDFILTEE	Filtee terminates filters search.
DF_1_DISPRELDNE	Disp reloc applied at build time.
DF_1_DISPRELPND	Disp reloc applied at run-time.

Definition at line 594 of file [elf.h](#).

14.16.9.3.7 Elf_DF_P1s

enum [Elf_DF_P1s](#)

Flags in the DT_POSFLAG_1 entry effecting only the next DT_* entry.

Enumerator

DF_P1_LAZYLOAD	Lazyload following object.
DF_P1_GROUPPERM	Symbols from next object are not generally available.

Definition at line 623 of file [elf.h](#).

14.16.9.3.8 Elf_DFs

enum [Elf_DFs](#)

Values of Elf32_Dyn.d_un.d_val, Elf64_Dyn.d_un.d_val in the DT_FLAGS entry.

Enumerator

DF_ORIGIN	Object may use DF_ORIGIN.
DF_SYMBOLIC	Symbol resolutions starts here.
DF_TEXTREL	Object contains text relocations.
DF_BIND_NOW	No lazy binding for this object.
DF_STATIC_TLS	Module uses the static TLS model.

Definition at line 581 of file [elf.h](#).

14.16.9.3.9 Elf_DTs

enum [Elf_DTs](#)

Dynamic Array Tags.

See also

[Elf32_Dyn.d_tag](#), [Elf64_Dyn.d_tag](#).

Enumerator

DT_NULL	end of _DYNAMIC array
DT_NEEDED	name of a needed library
DT_PLTRELSZ	total size of relocation entry
DT_PLTGOT	address assoc with prog link table
DT_HASH	address of symbol hash table
DT_STRTAB	address of string table
DT_SYMTAB	address of symbol table
DT_RELA	address of relocation table
DT_RELASZ	total size of relocation table
DT_RELAENT	size of DT_RELA relocation entry
DT_STRSZ	size of the string table
DT_SYMENT	size of a symbol table entry
DT_INIT	address of initialization function
DT_FINI	address of termination function
DT_SONAME	name of the shared object
DT_RPATH	search library path
DT_SYMBOLIC	alter symbol resolution algorithm
DT_REL	address of relocation table
DT_RELSZ	total size of DT_REL relocation table
DT_RELENT	size of the DT_REL relocation entry
DT_PTRREL	type of relocation entry
DT_DEBUG	for debugging purposes
DT_TEXTREL	at least on entry changes r/o section
DT_JMPREL	address of relocation entries
DT_BIND_NOW	Process relocations of object.
DT_INIT_ARRAY	Array with addresses of init fct.
DT_FINI_ARRAY	Array with addresses of fini fct.
DT_INIT_ARRAYSZ	Size in bytes of DT_INIT_ARRAY.
DT_FINI_ARRAYSZ	Size in bytes of DT_FINI_ARRAY.
DT_RUNPATH	Library search path.
DT_FLAGS	Flags for the object being loaded.
DT_ENCODING	Start of encoded range.
DT_PREINIT_ARRAY	Array with addresses of preinit fct.
DT_PREINIT_ARRAYSZ	size in bytes of DT_PREINIT_ARRAY
DT_NUM	Number used.
DT_LOOS	Start of OS-specific.
DT_HIOS	End of OS-specific.
DT_LOPROC	processor-specific
DT_HIPROC	processor-specific

Definition at line 535 of file [elf.h](#).

14.16.9.3.10 Elf_EF_ARM_s

enum [Elf_EF_ARM_s](#)

ARM specific declarations.

Processor specific flags for the ELF header e_flags field.

Enumerator

EF_ARM_ALIGN8	8-bit structure alignment is in use
---------------	-------------------------------------

Definition at line [730](#) of file [elf.h](#).

14.16.9.3.11 Elf_EIs

enum [Elf_EIs](#)

Identification Indices.

See also

[Elf32_Ehdr.e_ident](#), [Elf64_Ehdr.e_ident](#)

Enumerator

EI_MAG0	file id 0
EI_MAG1	file id 1
EI_MAG2	file id 2
EI_MAG3	file id 3
EI_CLASS	file class
EI_DATA	data encoding
EI_VERSION	file version
EI_OSABI	Operating system / ABI identification.
EI_ABIVERSION	ABI version.
EI_PAD	start of padding bytes

Definition at line [274](#) of file [elf.h](#).

14.16.9.3.12 Elf_EMs

enum [Elf_EMs](#)

Required architecture.

See also

[Elf32_Ehdr.e_machine](#), [Elf64_Ehdr.e_machine](#)

Enumerator

EM_NONE	no machine
EM_M32	AT&T WE 32100.
EM_SPARC	SPARC.
EM_386	Intel 80386.
EM_68K	Motorola 68000.
EM_88K	Motorola 88000.
EM_860	Intel 80860.
EM_MIPS	MIPS RS3000 big-endian.
EM_MIPS_RS4_BE	MIPS RS4000 big-endian.
EM_SPARC64	SPARC 64-bit.
EM_PARISC	HP PA-RISC.
EM_VPP500	Fujitsu VPP500.
EM_SPARC32PLUS	Sun's V8plus.
EM_960	Intel 80960.
EM_PPC	PowerPC.
EM_V800	NEC V800.
EM_FR20	Fujitsu FR20.
EM_RH32	TRW RH-32.
EM_RCE	Motorola RCE.
EM_ARM	Advanced RISC Machines ARM.
EM_ALPHA	Digital Alpha.
EM_SH	Hitachi SuperH.
EM_SPARCV9	SPARC v9 64-bit.
EM_TRICORE	Siemens Tricore embedded processor.
EM_ARC	Argonaut RISC Core, Argonaut Techn Inc.
EM_H8_300	Hitachi H8/300.
EM_H8_300H	Hitachi H8/300H.
EM_H8S	Hitachi H8/S.
EM_H8_500	Hitachi H8/500.
EM_IA_64	HP/Intel IA-64.
EM_MIPS_X	Stanford MIPS-X.
EM_COLDFIRE	Motorola Coldfire.
EM_68HC12	Motorola M68HC12.
EM_X86_64	Advanced Micro Devices x86-64.
EM_PDSP	Sony DSP Processor.
EM_FX66	Siemens FX66 microcontroller.
EM_ST9PLUS	STMicroelectronics ST9+ 8/16 mc.
EM_ST7	STmicroelectronics ST7 8 bit mc.
EM_68HC16	Motorola MC68HC16 microcontroller.
EM_68HC11	Motorola MC68HC11 microcontroller.
EM_68HC08	Motorola MC68HC08 microcontroller.
EM_68HC05	Motorola MC68HC05 microcontroller.

EM_SVX	Silicon Graphics SVx.
EM_ST19	STMicroelectronics ST19 8 bit mc.
EM_VAX	Digital VAX.
EM_CRIS	Axis Communications 32-bit embedded processor.
EM_JAVELIN	Infineon Technologies 32-bit embedded processor.
EM_FIREPATH	Element 14 64-bit DSP Processor.
EM_ZSP	LSI Logic 16-bit DSP Processor.
EM_MMIX	Donald Knuth's educational 64-bit processor.
EM_HUANY	Harvard University machine-independent object files.
EM_PRISM	SiTera Prism.
EM_AVR	Atmel AVR 8-bit microcontroller.
EM_FR30	Fujitsu FR30.
EM_D10V	Mitsubishi D10V.
EM_D30V	Mitsubishi D30V.
EM_V850	NEC v850.
EM_M32R	Mitsubishi M32R.
EM_MN10300	Matsushita MN10300.
EM_MN10200	Matsushita MN10200.
EM_PJ	picoJava
EM_OPENRISC	OpenRISC 32-bit embedded processor.
EM_ARC_A5	ARC Cores Tangent-A5.
EM_XTENSA	Tensilica Xtensa Architecture.
EM_ALTERA_NIOS2	Altera Nios II.
EM_AARCH64	ARM AARCH64.
EM_TILEPRO	Tilera TILEPro.
EM_MICROBLAZE	Xilinx MicroBlaze.
EM_TILEGX	Tilera TILE-Gx.
EM_RISCV	RISC-V.

Definition at line 183 of file [elf.h](#).

14.16.9.3.13 Elf_ETs

```
enum Elf_ETs
```

Object file type.

See also

[Elf32_Ehdr.e_type](#), [Elf64_Ehdr.e_type](#)

Enumerator

ET_NONE	no file type
ET_REL	relocatable file
ET_EXEC	executable file
ET_DYN	shared object file
ET_CORE	core file
ET_LOPROC	processor-specific
ET_HIPROC	processor-specific

Definition at line 168 of file [elf.h](#).

14.16.9.3.14 Elf_EVs

enum [Elf_EVs](#)

Object file version.

See also

[Elf32_Ehdr.e_version](#), [Elf64_Ehdr.e_version](#)

Enumerator

EV_NONE	Invalid version.
EV_CURRENT	Current version.

Definition at line 266 of file [elf.h](#).

14.16.9.3.15 Elf_MAGs

enum [Elf_MAGs](#)

Magic number.

Enumerator

ELFMAG0	e_ident[EI_MAG0]
ELFMAG1	e_ident[EI_MAG1]
ELFMAG2	e_ident[EI_MAG2]
ELFMAG3	e_ident[EI_MAG3]

Definition at line 289 of file [elf.h](#).

14.16.9.3.16 Elf_NTscore

enum [Elf_NTscore](#)

Legal values for note segment descriptor types for core files.

Enumerator

NT_PRSTATUS	Contains copy of prstatus struct.
NT_FPREGSET	Contains copy of fpregset struct.
NT_PRPSINFO	Contains copy of prpsinfo struct.
NT_PRXREG	Contains copy of prxregset struct.
NT_TASKSTRUCT	Contains copy of task structure.
NT_PLATFORM	String from sysinfo(SI_PLATFORM).
NT_AUXV	Contains copy of auxv array.
NT_GWINDOWS	Contains copy of gwindows struct.
NT_ASRS	Contains copy of asrset struct.
NT_PSTATUS	Contains copy of pstatus struct.
NT_PSINFO	Contains copy of psinfo struct.
NT_PRCRED	Contains copy of prcred struct.
NT_UTSNAME	Contains copy of utsname struct.
NT_LWPSTATUS	Contains copy of lwpstatus struct.
NT_LWPSINFO	Contains copy of lwpinfo struct.
NT_PRFPXREG	Contains copy of fprxregset struct.

Definition at line 486 of file [elf.h](#).

14.16.9.3.17 Elf_NTs_obj

enum [Elf_NTs_obj](#)

Legal values for the note segment descriptor types for object files.

Enumerator

NT_VERSION	Contains a version string.
------------	----------------------------

Definition at line 507 of file [elf.h](#).

14.16.9.3.18 Elf_OSABIs

enum [Elf_OSABIs](#)

Identify operating system and ABI to which the object is targeted.

Enumerator

ELFOSABI_NONE	UNIX System V ABI.
ELFOSABI_SYSV	Alias.
ELFOSABI_HPUX	HP-UX.
ELFOSABI_NETBSD	NetBSD.

ELFOSABI_LINUX	Linux.
ELFOSABI_SOLARIS	Sun Solaris.
ELFOSABI_AIX	IBM AIX.
ELFOSABI_IRIX	SGI Irix.
ELFOSABI_FREEBSD	FreeBSD.
ELFOSABI_TRU64	Compaq TRU64 UNIX.
ELFOSABI_MODESTO	Novell Modesto.
ELFOSABI_OPENBSD	OpenBSD.
ELFOSABI_ARM	ARM.
ELFOSABI_STANDALONE	Standalone (embedded) application.

Definition at line 316 of file [elf.h](#).

14.16.9.3.19 ELF_PFs

enum [ELF_PFs](#)

Segment permissions.

Enumerator

PF_X	Executable.
PF_W	Write.
PF_R	Read.
PF_MASKOS	OS-specific.
PF_MASKPROC	Processor-specific.

Definition at line 476 of file [elf.h](#).

14.16.9.3.20 Elf_PTs

enum [Elf_PTs](#)

Segment types.

Enumerator

PT_NULL	array is unused
PT_LOAD	loadable
PT_DYNAMIC	dynamic linking information
PT_INTERP	path to interpreter

PT_NOTE	auxiliary information
PT_SHLIB	reserved
PT_PHDR	location of the pht itself
PT_TLS	Thread-local storage segment.
PT_NUM	Number of defined types.
PT_LOOS	OS-specific.
PT_HIOS	OS-specific.
PT_LOPROC	processor-specific
PT_HIPROC	processor-specific
PT_GNU_EH_FRAME	EH frame information.
PT_GNU_STACK	Flags for stack.
PT_GNU_RELRO	Read only after reloc.
PT_L4_STACK	Address of the stack.
PT_L4_AUX	Address of the AUX structures.

Definition at line 451 of file [elf.h](#).

14.16.9.3.21 Elf_R_386_s

enum [Elf_R_386_s](#)

Relocation types (processor specific).

Enumerator

R_386_NONE	none
R_386_32	$S + A$.
R_386_PC32	$S + A - P$.
R_386_GOT32	$G + A - P$.
R_386_PLT32	$L + A - P$.
R_386_COPY	none
R_386_GLOB_DAT	S .
R_386_JMP_SLOT	S .
R_386_RELATIVE	$B + A$.
R_386_GOTOFF	$S + A - GOT$.
R_386_GOTPC	$GOT + A - P$.
R_386_TLS_TPOFF	Offset in static TLS block.
R_386_TLS_IE	Address of GOT entry for static TLS block offset.
R_386_TLS_GOTIE	GOT entry for static TLS block offset.
R_386_TLS_LE	Offset relative to static TLS block.
R_386_TLS_GD	Direct 32 bit for GNU version of general dynamic thread local data.
R_386_TLS_LDM	Direct 32 bit for GNU version of local dynamic thread local data in LE code.
R_386_TLS_GD_32	Direct 32 bit for general dynamic thread local data.

R_386_TLS_GD_PUSH	Tag for pushl in GD TLS code.
R_386_TLS_GD_CALL	Relocation for call to __tls_get_addr().
R_386_TLS_GD_POP	Tag for popl in GD TLS code.
R_386_TLS_LDM_32	Direct 32 bit for local dynamic thread local data in LE code.
R_386_TLS_LDM_PUSH	Tag for pushl in LDM TLS code.
R_386_TLS_LDM_CALL	Relocation for call to __tls_get_addr() in LDM code.
R_386_TLS_LDM_POP	Tag for popl in LDM TLS code.
R_386_TLS_LDO_32	Offset relative to TLS block.
R_386_TLS_IE_32	GOT entry for negated static TLS block offset.
R_386_TLS_LE_32	Negated offset relative to static TLS block.
R_386_TLS_DTPMOD32	ID of module containing symbol.
R_386_TLS_DTPOFF32	Offset in TLS block.
R_386_TLS_TPOFF32	Negated offset in static TLS block.
R_386_NUM	Keep this the last entry.

Definition at line 677 of file [elf.h](#).

14.16.9.3.22 Elf_R_AARCH64_s

enum [Elf_R_AARCH64_s](#)

AARCH64 relocations.

Enumerator

R_AARCH64_NONE	No reloc.
----------------	-----------

Definition at line 825 of file [elf.h](#).

14.16.9.3.23 Elf_R_ARM_s

enum [Elf_R_ARM_s](#)

ARM relocations.

Enumerator

R_ARM_NONE	No reloc.
R_ARM_PC24	PC relative 26 bit branch.
R_ARM_ABS32	Direct 32 bit.
R_ARM_REL32	PC relative 32 bit.
R_ARM_ABS16	Direct 16 bit.
R_ARM_ABS12	Direct 12 bit.

R_ARM_ABS8	Direct 8 bit.
R_ARM_COPY	Copy symbol at runtime.
R_ARM_GLOB_DAT	Create GOT entry.
R_ARM_JUMP_SLOT	Create PLT entry.
R_ARM_RELATIVE	Adjust by program base.
R_ARM_GOTOFF	32 bit offset to GOT
R_ARM_GOTPC	32 bit PC relative offset to GOT
R_ARM_GOT32	32 bit GOT entry
R_ARM_PLT32	32 bit PLT address
R_ARM_THM_PC11	thumb unconditional branch
R_ARM_THM_PC9	thumb conditional branch
R_ARM_NUM	Keep this the last entry.

Definition at line 777 of file [elf.h](#).

14.16.9.3.24 Elf_R_X86_64_s

enum [Elf_R_X86_64_s](#)

AMD x86-64 relocations.

Enumerator

R_X86_64_NONE	No reloc.
R_X86_64_64	Direct 64 bit.
R_X86_64_PC32	PC relative 32 bit signed.
R_X86_64_GOT32	32 bit GOT entry
R_X86_64_PLT32	32 bit PLT address
R_X86_64_COPY	Copy symbol at runtime.
R_X86_64_GLOB_DAT	Create GOT entry.
R_X86_64_JUMP_SLOT	Create PLT entry.
R_X86_64_RELATIVE	Adjust by program base.
R_X86_64_GOTPCREL	32 bit signed PC relative offset to GOT
R_X86_64_32	Direct 32 bit zero extended.
R_X86_64_32S	Direct 32 bit sign extended.
R_X86_64_16	Direct 16 bit zero extended.
R_X86_64_PC16	16 bit sign extended pc relative
R_X86_64_8	Direct 8 bit sign extended.
R_X86_64_PC8	8 bit sign extended pc relative
R_X86_64_DTPOFF64	ID of module containing symbol.
R_X86_64_DTPOFF64	Offset in module's TLS block.
R_X86_64_TPOFF64	Offset in initial TLS block.

R_X86_64_TLSD	32 bit signed PC relative offset to two GOT entries for GD symbol
R_X86_64_TLSD	32 bit signed PC relative offset to two GOT entries for LD symbol
R_X86_64_DTPOFF32	Offset in TLS block.
R_X86_64_GOTPOFF	32 bit signed PC relative offset to GOT entry for IE symbol
R_X86_64_TPOFF32	Offset in initial TLS block.

Definition at line 832 of file [elf.h](#).

14.16.9.3.25 Elf_SHF_s_ARM

enum [Elf_SHF_s_ARM](#)

ARM-specific values for [Elf32_Shdr.sh_flags](#) / [Elf64_Shdr.sh_flags](#).

Enumerator

SHF_ARM_ENTRYSECT	Section contains an entry point.
SHF_ARM_COMDEF	Section may be multiply defined in the input to a link step.

Definition at line 762 of file [elf.h](#).

14.16.9.3.26 Elf_SHFs

enum [Elf_SHFs](#)

Section attribute flags.

Enumerator

SHF_WRITE	writable during execution
SHF_ALLOC	section occupies virt memory
SHF_EXECINSTR	code section
SHF_MERGE	Might be merged.
SHF_STRINGS	Contains nul-terminated strings.
SHF_INFO_LINK	`sh_info' contains SHT index */ SHF_LINK_ORDER = 0x80, /**< Preserve order after combining
SHF_OS_NONCONFORMING	Non-standard OS-specific handling required.
SHF_GROUP	Section is member of a group.
SHF_TLS	Section hold thread-local data.
SHF_MASKOS	OS-specific.
SHF_MASKPROC	processor-specific mask

Definition at line 406 of file [elf.h](#).

14.16.9.3.27 Elf_SHNs

enum [Elf_SHNs](#)

Special section indexes.

Enumerator

SHN_UNDEF	undefined section header entry
SHN_LORESERVE	lower bound of reserved indexes
SHN_LOPROC	lower bound of proc spec entr
SHN_HIPROC	upper bound of proc spec entr
SHN_ABS	absolute values for ref
SHN_COMMON	common symbols
SHN_HIRESERVE	upper bound of reserved indexes

Definition at line 335 of file [elf.h](#).

14.16.9.3.28 Elf_SHTs

enum [Elf_SHTs](#)

Section type.

Enumerator

SHT_NULL	inactive section header
SHT_PROGBITS	information defined by program
SHT_SYMTAB	symbol table
SHT_STRTAB	string table
SHT_RELA	reloc entries w/ explicit addends
SHT_HASH	symbol hash table
SHT_DYNAMIC	information for dynamic linking
SHT_NOTE	information that marks the file
SHT_NOBITS	occupies no space in the file
SHT_REL	reloc entries w/o explicit addends
SHT_SHLIB	reserved + unspecified semantics
SHT_DYNSYM	symbol table (dynamic
SHT_INIT_ARRAY	Array of constructors.
SHT_FINI_ARRAY	Array of destructors.
SHT_PREINIT_ARRAY	Array of pre-constructors.
SHT_GROUP	Section group.
SHT_SYMTAB_SHNDX	Extended section indices.
SHT_NUM	Number of defined types.
SHT_LOOS	Start OS-specific.
SHT_HIOS	End OS-specific.
SHT_LOPROC	Start processor-specific.
SHT_HIPROC	End processor-specific.
SHT_LOUSER	Start application-specific.
SHT_HIUSER	End application-specific.

Definition at line 377 of file [elf.h](#).

14.16.9.3.29 Elf_STBs

enum [Elf_STBs](#)

Symbol Binding.

See also

[ELF32_ST_BIND](#), [ELF64_ST_BIND](#)

Enumerator

STB_LOCAL	not visible outside object file
STB_GLOBAL	visible to all objects being combined
STB_WEAK	resemble global symbols
STB_LOOS	OS-specific.
STB_HIOS	OS-specific.
STB_LOPROC	Processor-specific.
STB_HIPROC	Processor-specific.

Definition at line [912](#) of file [elf.h](#).

14.16.9.3.30 Elf_STTs

enum [Elf_STTs](#)

Symbol Types.

See also

[ELF32_ST_TYPE](#), [ELF64_ST_TYPE](#)

Enumerator

STT_NOTYPE	symbol's type not specified
STT_OBJECT	associated with a data object
STT_FUNC	associated with a function or other code
STT_SECTION	associated with a section
STT_FILE	source file name associated with object
STT_LOOS	OS-specific.
STT_HIOS	OS-specific.
STT_LOPROC	processor-specific
STT_HIPROC	processor-specific

Definition at line [925](#) of file [elf.h](#).

14.16.10 Kernel Interface Page API

Collaboration diagram for Kernel Interface Page API:



Files

- file [kip.h](#)

Macros

- `#define l4util_kip_for_each_feature(s)`
Cycle through kernel features given in the KIP.

Functions

- `L4_BEGIN_DECLS int l4util_kip_kernel_has_feature (l4_kernel_info_t const *k, char const *str)`
Check if kernel supports a feature.
- `unsigned long l4util_kip_kernel_abi_version (l4_kernel_info_t const *k)`
Return kernel ABI version.

14.16.10.1 Detailed Description

14.16.10.2 Macro Definition Documentation

14.16.10.2.1 l4util_kip_for_each_feature

```
#define l4util_kip_for_each_feature(  
    s)
```

Value:

`l4_kip_for_each_feature(s)`

Cycle through kernel features given in the KIP.

Cycles through all KIP kernel feature strings. `s` must be a character pointer (`char const *`) initialized with `l4_kip_version_string()`.

Deprecated Use `l4_kip_for_each_feature()`.

Definition at line 58 of file [kip.h](#).

14.16.10.3 Function Documentation

14.16.10.3.1 l4util_kip_kernel_abi_version()

```
unsigned long l4util_kip_kernel_abi_version (  
    l4_kernel_info_t const * k)
```

Return kernel ABI version.

Parameters

<i>k</i>	Pointer to the kernel info page (KIP).
----------	--

Returns

Kernel ABI version.

References [L4_CV](#), and [L4_END_DECLS](#).

14.16.10.3.2 l4util_kip_kernel_has_feature()

```
L4_BEGIN_DECLS int l4util_kip_kernel_has_feature (  
    l4_kernel_info_t const * k,  
    char const * str)
```

Check if kernel supports a feature.

Parameters

<i>k</i>	Pointer to the kernel info page (KIP).
<i>str</i>	Feature name to check.

Returns

1 if the kernel supports the feature, 0 if not.

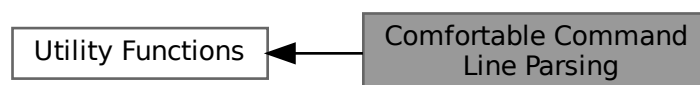
Checks the feature field in the KIP for the given string.

Deprecated Use [l4_kip_kernel_has_feature\(\)](#).

References [L4_CV](#).

14.16.11 Comfortable Command Line Parsing

Collaboration diagram for Comfortable Command Line Parsing:



Typedefs

- typedef void(* **parse_cmd_fn_t**) (int)
Function type for PARSE_CMD_FN.
- typedef void(* **parse_cmd_fn_arg_t**) (int, const char *, int)
Function type for PARSE_CMD_FN_ARG.

Enumerations

- enum **parse_cmd_type**
Types for parsing.

Functions

- [L4_BEGIN_DECLS](#) int **parse_cmdline** (int *argc, const char ***argv, int arg0,...)
Parse the command-line for specified arguments and store the values into variables.

14.16.11.1 Detailed Description

14.16.11.2 Function Documentation

14.16.11.2.1 **parse_cmdline()**

```
L4_BEGIN_DECLS int parse_cmdline (
    int * argc,
    const char *** argv,
    int arg0,
    ...)
```

Parse the command-line for specified arguments and store the values into variables.

This Functions gets the command-line, and a list of command-descriptors. Then, the command-line is parsed according to the given descriptors, storing strings, switches and numeric arguments at given addresses, and possibly calling specified functions. A default help descriptor is added. Its purpose is to present a short command overview in the case the given command-line does not fit to the descriptors.

Each command-descriptor has the following form:

short option char, long option name, comment, type, val, addr.

The *short option char* specifies the short form of the described option. The short form will be recognized after a single dash, or in a group of short options preceeded by a single dash. Specify ' ' if no short form should be used.

The *long option name* specifies the long form of the described option. The long form will be recognized after two dashes. Specify 0 if no long form should be used for this option.

The *comment* is a string that will be used when presenting the short command-line help.

The *type* specifies, if the option should be recognized as

- a number (PARSE_CMD_INT),

- a switch (`PARSE_CMD_SWITCH`),
- a string (`PARSE_CMD_STRING`),
- a function call (`PARSE_CMD_FN`, `PARSE_CMD_FN_ARG`),
- an increment/decrement operator (`PARSE_CMD_INC`, `PARSE_CMD_DEC`).

If `type` is `PARSE_CMD_INT`, the option requires a second argument on the command-line after the option. This argument is parsed as a number. It can be preceeded by 0x to present a hex-value or by 0 to present an octal form. `addr` is interpreted as an int-pointer. The scanned argument from the command-line is stored in this pointer.

If `type` is `PARSE_CMD_SWITCH`, `addr` must be a pointer to int, and the value from `val` is stored at this pointer.

With `PARSE_CMD_STRING`, an additional argument is expected at the cmdline. `addr` must be a pointer to `const char*`, and a pointer to the argument on the command line is stored at this pointer. The value in `val` is a default value, which is stored at `addr` if the corresponding option is not given on the command line.

With `PARSE_CMD_FN_ARG`, `addr` is interpreted as a function pointer of type `parse_cmd_fn_t`. It will be called with `val` as argument if the corresponding option is found.

If `type` is `PARSE_CMD_FN_ARG`, `addr` is as a function pointer of type `parse_cmd_fn_arg_t`, and handled similar to `PARSE_CMD_FN`. An additional argument is expected at the command line, however. It is given to the called function as 2nd argument, and parsed as an integer as with `PARSE_CMD_INT` as a third argument.

If `type` is `PARSE_CMD_INC` or `PARSE_CMD_DEC`, `addr` is interpreted as an int-pointer. The value of `val` is stored to this pointer first. For every occurrence of the option in the command line, the integer referenced by `addr` is incremented or decremented, respectively.

The list of command-descriptors is terminated by specifying a binary 0 for the short option char.

Note: The short option char 'h' and the long option name "help" must not be specified. They are used for the default help descriptor and produce a short command-options help when specified on the command-line.

Parameters

<code>argc</code>	pointer to number of command line parameters as passed to main
<code>argv</code>	pointer to array of command line parameters as passed to main
<code>arg0</code>	format list describing the command line options to parse for

Returns

0 if the command-line was successfully parsed, otherwise:

- -1 if the given descriptors are somehow wrong.
- -2 if not enough memory was available to hold temporary structs.
- -3 if the given command-line args did not meet the specified set.
- -4 if the help-option was given.

Upon return, `argc` and `argv` point to a list of arguments that were not scanned as arguments. See `getoptlong` for details on scanning.

References [L4_CV](#), and [L4_END_DECLS](#).

14.16.12 Random number support

Collaboration diagram for Random number support:



Functions

- `l4_uint32_t l4util_rand` (void)
Deliver next random number.
- void `l4util_srand` (`l4_uint32_t` seed)
Initialize random number generator.

14.16.12.1 Detailed Description

14.16.12.2 Function Documentation

14.16.12.2.1 l4util_rand()

```
l4_uint32_t l4util_rand (
    void )
```

Deliver next random number.

Returns

A new random number

References [L4_CV](#).

14.16.12.2.2 l4util_srand()

```
void l4util_srand (
    l4_uint32_t seed)
```

Initialize random number generator.

Parameters

<i>seed</i>	Value to initialize
-------------	---------------------

References [L4_END_DECLS](#).

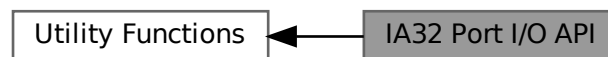
14.16.13 Low-Level Thread Functions

Collaboration diagram for Low-Level Thread Functions:



14.16.14 IA32 Port I/O API

Collaboration diagram for IA32 Port I/O API:



Functions

- [L4_BEGIN_DECLS](#) `int l4util_ioport_map (l4_cap_idx_t sigma0id, unsigned port_start, unsigned log2size)`
Map a range of I/O ports.
- [l4_uint8_t](#) `l4util_in8 (l4_uint16_t port)`
Read byte from I/O port.
- [l4_uint16_t](#) `l4util_in16 (l4_uint16_t port)`
Read 16-bit-value from I/O port.
- [l4_uint32_t](#) `l4util_in32 (l4_uint16_t port)`
Read 32-bit-value from I/O port.
- `void l4util_ins8 (l4_uint16_t port, l4_umword_t addr, l4_umword_t count)`
Read a block of 8-bit-values from I/O ports.
- `void l4util_ins16 (l4_uint16_t port, l4_umword_t addr, l4_umword_t count)`
Read a block of 16-bit-values from I/O ports.
- `void l4util_ins32 (l4_uint16_t port, l4_umword_t addr, l4_umword_t count)`
Read a block of 32-bit-values from I/O ports.
- `void l4util_out8 (l4_uint8_t value, l4_uint16_t port)`
Write byte to I/O port.
- `void l4util_out16 (l4_uint16_t value, l4_uint16_t port)`
Write 16-bit-value to I/O port.
- `void l4util_out32 (l4_uint32_t value, l4_uint16_t port)`
Write 32-bit-value to I/O port.
- `void l4util_outs8 (l4_uint16_t port, l4_umword_t addr, l4_umword_t count)`

Write a block of bytes to I/O port.

- void `l4util_outs16` (`l4_uint16_t` port, `l4_umword_t` addr, `l4_umword_t` count)

Write a block of 16-bit-values to I/O port.

- void `l4util_outs32` (`l4_uint16_t` port, `l4_umword_t` addr, `l4_umword_t` count)

Write block of 32-bit-values to I/O port.

- void `l4util_iodelay` (void)

delay I/O port access by writing to port 0x80

14.16.14.1 Detailed Description

14.16.14.2 Function Documentation

14.16.14.2.1 l4util_in16()

```
l4_uint16_t l4util_in16 (
    l4_uint16_t port) [inline]
```

Read 16-bit-value from I/O port.

Parameters

<i>port</i>	I/O port address
-------------	------------------

Returns

value

Definition at line 176 of file [port_io.h](#).

14.16.14.2.2 l4util_in32()

```
l4_uint32_t l4util_in32 (
    l4_uint16_t port) [inline]
```

Read 32-bit-value from I/O port.

Parameters

<i>port</i>	I/O port address
-------------	------------------

Returns

value

Definition at line 184 of file [port_io.h](#).

14.16.14.2.3 l4util_in8()

```
L4_END_DECLS l4_uint8_t l4util_in8 (
    l4_uint16_t port) [inline]
```

Read byte from I/O port.

Parameters

<i>port</i>	I/O port address
-------------	------------------

Returns

value

Definition at line 168 of file [port_io.h](#).

14.16.14.2.4 l4util_ins16()

```
void l4util_ins16 (  
    l4_uint16_t port,  
    l4_umword_t addr,  
    l4_umword_t count) [inline]
```

Read a block of 16-bit-values from I/O ports.

Parameters

<i>port</i>	I/O port address
<i>addr</i>	address of buffer
<i>count</i>	number of I/O operations

Definition at line 201 of file [port_io.h](#).

14.16.14.2.5 l4util_ins32()

```
void l4util_ins32 (  
    l4_uint16_t port,  
    l4_umword_t addr,  
    l4_umword_t count) [inline]
```

Read a block of 32-bit-values from I/O ports.

Parameters

<i>port</i>	I/O port address
<i>addr</i>	address of buffer
<i>count</i>	number of I/O operations

Definition at line 210 of file [port_io.h](#).

14.16.14.2.6 l4util_ins8()

```
void l4util_ins8 (  
    l4_uint16_t port,  
    l4_umword_t addr,  
    l4_umword_t count) [inline]
```

Read a block of 8-bit-values from I/O ports.

Parameters

<i>port</i>	I/O port address
<i>addr</i>	address of buffer
<i>count</i>	number of I/O operations

Definition at line 192 of file [port_io.h](#).

14.16.14.2.7 l4util_ioport_map()

```
L4_END_DECLS int l4util_ioport_map (  
    l4_cap_idx_t sigma0id,  
    unsigned port_start,  
    unsigned log2size) [inline]
```

Map a range of I/O ports.

Parameters

<i>sigma0id</i>	I/O port service (sigma0).
<i>port_start</i>	(Start) Port to request.
<i>log2size</i>	Log2size of range to request.

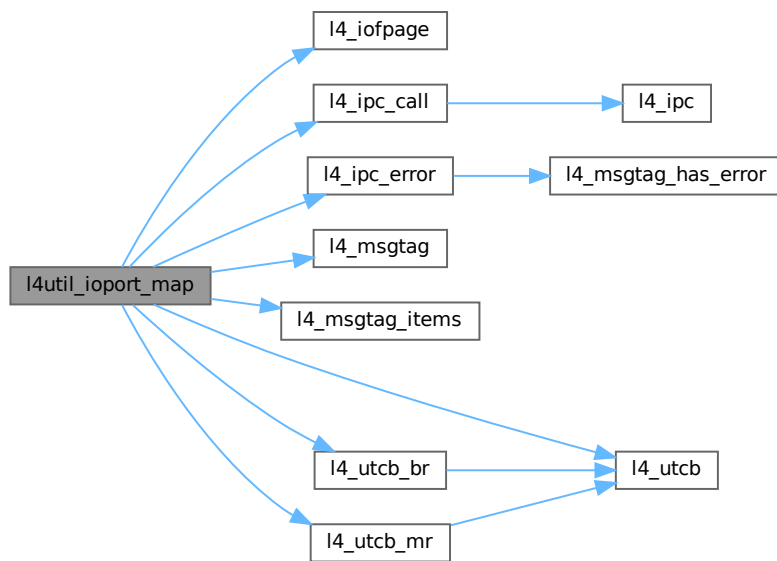
Returns

IPC result: 0 if the range could be successfully mapped on error: IPC failure, or -L4_ENOENT if nothing mapped

Definition at line 54 of file [port_io.h](#).

References [l4_buf_regs_t::bdr](#), [l4_buf_regs_t::br](#), [L4_ENOENT](#), [l4_iofpage\(\)](#), [l4_ipc_call\(\)](#), [l4_ipc_error\(\)](#), [L4_IPC_NEVER](#), [L4_ITEM_MAP](#), [l4_msgtag\(\)](#), [l4_msgtag_items\(\)](#), [L4_PROTO_IO_PAGE_FAULT](#), [l4_utcb\(\)](#), [l4_utcb_br\(\)](#), [l4_utcb_mr\(\)](#), [l4_msg_regs_t::mr](#), and [l4_fpage_t::raw](#).

Here is the call graph for this function:



14.16.14.2.8 l4util_out16()

```
void l4util_out16 (
    l4_uint16_t value,
    l4_uint16_t port) [inline]
```

Write 16-bit-value to I/O port.

Parameters

<i>port</i>	I/O port address
<i>value</i>	value to write

Definition at line 225 of file [port_io.h](#).

14.16.14.2.9 l4util_out32()

```
void l4util_out32 (
    l4_uint32_t value,
    l4_uint16_t port) [inline]
```

Write 32-bit-value to I/O port.

Parameters

<i>port</i>	I/O port address
<i>value</i>	value to write

Definition at line 231 of file [port_io.h](#).

14.16.14.2.10 l4util_out8()

```
void l4util_out8 (  
    l4_uint8_t value,  
    l4_uint16_t port) [inline]
```

Write byte to I/O port.

Parameters

<i>port</i>	I/O port address
<i>value</i>	value to write

Definition at line 219 of file [port_io.h](#).

14.16.14.2.11 l4util_outs16()

```
void l4util_outs16 (  
    l4_uint16_t port,  
    l4_umword_t addr,  
    l4_umword_t count) [inline]
```

Write a block of 16-bit-values to I/O port.

Parameters

<i>port</i>	I/O port address
<i>addr</i>	address of buffer
<i>count</i>	number of I/O operations

Definition at line 246 of file [port_io.h](#).

14.16.14.2.12 l4util_outs32()

```
void l4util_outs32 (  
    l4_uint16_t port,  
    l4_umword_t addr,  
    l4_umword_t count) [inline]
```

Write block of 32-bit-values to I/O port.

Parameters

<i>port</i>	I/O port address
<i>addr</i>	address of buffer
<i>count</i>	number of I/O operations

Definition at line 255 of file [port_io.h](#).

14.16.14.2.13 l4util_outs8()

```
void l4util_outs8 (
    l4_uint16_t port,
    l4_umword_t addr,
    l4_umword_t count) [inline]
```

Write a block of bytes to I/O port.

Parameters

<i>port</i>	I/O port address
<i>addr</i>	address of buffer
<i>count</i>	number of I/O operations

Definition at line 237 of file [port_io.h](#).

14.17 Virtio Net Switch

A virtual network switch that can be used as defined in the virtio protocol.

Data Structures

- class [Mac_addr](#)
A wrapper class around the value of a MAC address.
- class [Mac_table< Size >](#)
Mac_table manages a 1:n association between ports and MAC addresses.
- class [Switch_factory](#)
The IPC interface for creating ports.
- class [L4virtio_port](#)
A Port on the Virtio Net Switch.
- class [Net_transfer](#)
A network request to only a single destination.
- class [Virtio_net_request](#)
Abstraction for a network request.
- class [Virtio_switch](#)
The Virtio switch contains all ports and processes network requests.
- struct [Buffer](#)
Data buffer used to transfer packets.
- class [Virtio_vlan_mangle](#)
Class for VLAN packet rewriting.

14.17.1 Detailed Description

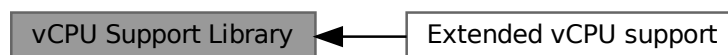
A virtual network switch that can be used as defined in the virtio protocol.

The abstraction of a single connection with a network device (also called client) from the switch's perspective is a port. A client can register multiple ports on the switch. The communication between a client and the switch happens via IRQs, MMIO and shared memory as defined by the Virtio protocol. The switch supports VLANs and ports can be either 'access' or 'trunk' ports. The optionally available monitor port receives network traffic from all ports, and the monitor can not send.

14.18 vCPU Support Library

vCPU handling functionality.

Collaboration diagram for vCPU Support Library:



Topics

- Extended vCPU support ??
Extended vCPU handling functionality.

Data Structures

- class [L4vcpu::State](#)
C++ implementation of state word in the vCPU area.
- class [L4vcpu::Vcpu](#)
C++ implementation of the vCPU save state area.

Functions

- void [l4vcpu_irq_disable](#) ([l4_vcpu_state_t](#) *vcpu) [L4_NOTHROW](#)
Disable a vCPU for event delivery.
- unsigned [l4vcpu_irq_disable_save](#) ([l4_vcpu_state_t](#) *vcpu) [L4_NOTHROW](#)
Disable a vCPU for event delivery and return previous state.
- void [l4vcpu_irq_enable](#) ([l4_vcpu_state_t](#) *vcpu, [l4_utcb_t](#) *utcb, [l4vcpu_event_hndl_t](#) do_event_work_cb, [l4vcpu_setup_ipc_t](#) setup_ipc) [L4_NOTHROW](#)
Enable a vCPU for event delivery.
- void [l4vcpu_irq_restore](#) ([l4_vcpu_state_t](#) *vcpu, unsigned s, [l4_utcb_t](#) *utcb, [l4vcpu_event_hndl_t](#) do_event_work_cb, [l4vcpu_setup_ipc_t](#) setup_ipc) [L4_NOTHROW](#)

Restore a previously saved IRQ/event state.

- void [l4vcpu_wait_for_event](#) ([l4_vcpu_state_t](#) *vcpu, [l4_utcb_t](#) *utcb, [l4vcpu_event_hndl_t](#) do_event_work←_cb, [l4vcpu_setup_ipc_t](#) setup_ipc) [L4_NOTHROW](#)

Wait for event.

- void [l4vcpu_print_state](#) (const [l4_vcpu_state_t](#) *vcpu, const char *prefix) [L4_NOTHROW](#)

Print the state of a vCPU.

- int [l4vcpu_is_irq_entry](#) ([l4_vcpu_state_t](#) const *vcpu) [L4_NOTHROW](#)

Return whether the entry reason was an IRQ/IPC message.

- int [l4vcpu_is_page_fault_entry](#) ([l4_vcpu_state_t](#) const *vcpu) [L4_NOTHROW](#)

Return whether the entry reason was a page fault.

14.18.1 Detailed Description

vCPU handling functionality.

This library provides convenience functionality on top of the l4sys vCPU interface to ease programming. It wraps commonly used code and abstracts architecture depends parts as far as reasonable.

14.18.2 Function Documentation

14.18.2.1 l4vcpu_irq_disable()

```
void l4vcpu_irq_disable (
    l4\_vcpu\_state\_t * vcpu) [inline]
```

Disable a vCPU for event delivery.

Parameters

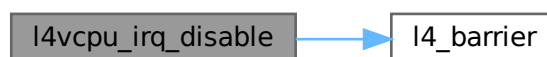
<i>vcpu</i>	Pointer to vCPU area.
-------------	-----------------------

Definition at line 201 of file [vcpu.h](#).

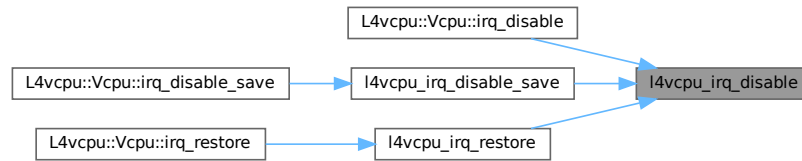
References [l4_barrier\(\)](#), [L4_NOTHROW](#), and [L4_VCPU_F_IRQ](#).

Referenced by [L4vcpu::Vcpu::irq_disable\(\)](#), [l4vcpu_irq_disable_save\(\)](#), and [l4vcpu_irq_restore\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.18.2.2 l4vcpu_irq_disable_save()

```

unsigned l4vcpu_irq_disable_save (
    l4_vcpu_state_t * vcpu) [inline]
  
```

Disable a vCPU for event delivery and return previous state.

Parameters

<i>vcpu</i>	Pointer to vCPU area.
-------------	-----------------------

Returns

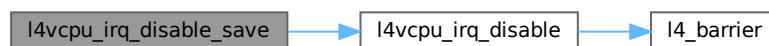
IRQ state before disabling IRQs.

Definition at line 209 of file [vcpu.h](#).

References [L4_NOTHROW](#), and [l4vcpu_irq_disable\(\)](#).

Referenced by [L4vcpu::Vcpu::irq_disable_save\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.18.2.3 l4vcpu_irq_enable()

```
void l4vcpu_irq_enable (
    l4_vcpu_state_t * vcpu,
    l4_utcb_t * utcb,
    l4vcpu_event_hndl_t do_event_work_cb,
    l4vcpu_setup_ipc_t setup_ipc) [inline]
```

Enable a vCPU for event delivery.

Parameters

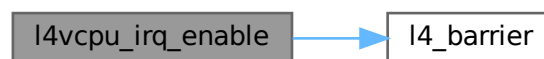
<i>vcpu</i>	Pointer to vCPU area.
<i>utcb</i>	Utcbl pointer of the calling vCPU.
<i>do_event_work_cb</i>	Call-back function that is called in case an event (such as an interrupt) is pending.
<i>setup_ipc</i>	Function call-back that is called right before any IPC operation, and before event delivery is enabled.

Definition at line 232 of file [vcpu.h](#).

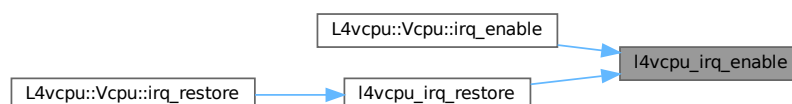
References [l4_barrier\(\)](#), [L4_IPC_BOTH_TIMEOUT_0](#), [L4_LIKELY](#), [L4_NOTHROW](#), [L4_VCPU_F_IRQ](#), and [L4_VCPU_SF_IRQ_PENDING](#).

Referenced by [L4vcpu::Vcpu::irq_enable\(\)](#), and [l4vcpu_irq_restore\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.18.2.4 l4vcpu_irq_restore()

```
void l4vcpu_irq_restore (
    l4_vcpu_state_t * vcpu,
    unsigned s,
    l4_utcb_t * utcb,
    l4vcpu_event_hndl_t do_event_work_cb,
    l4vcpu_setup_ipc_t setup_ipc) [inline]
```

Restore a previously saved IRQ/event state.

Parameters

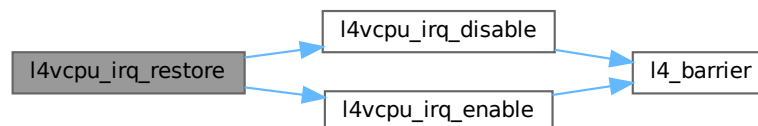
<i>vcpu</i>	Pointer to vCPU area.
<i>s</i>	IRQ state to be restored.
<i>utcb</i>	Utcbl pointer of the calling vCPU.
<i>do_event_work_cb</i>	Call-back function that is called in case an event (such as an interrupt) is pending after enabling.
<i>setup_ipc</i>	Function call-back that is called right before any IPC operation, and before event delivery is enabled.

Definition at line 257 of file [vcpu.h](#).

References [L4_NOTHROW](#), [L4_VCPU_F_IRQ](#), [l4vcpu_irq_disable\(\)](#), and [l4vcpu_irq_enable\(\)](#).

Referenced by [L4vcpu::Vcpu::irq_restore\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



14.18.2.5 l4vcpu_is_irq_entry()

```
int l4vcpu_is_irq_entry (  
    l4_vcpu_state_t const * vcpu) [inline]
```

Return whether the entry reason was an IRQ/IPC message.

Parameters

<i>vcpu</i>	Pointer to vCPU area.
-------------	-----------------------

return 0 if not, !=0 otherwise.

References [L4_CV](#), [L4_INLINE](#), and [L4_NOTHROW](#).

Referenced by [L4vcpu::Vcpu::is_irq_entry\(\)](#).

Here is the caller graph for this function:



14.18.2.6 l4vcpu_is_page_fault_entry()

```
int l4vcpu_is_page_fault_entry (  
    l4_vcpu_state_t const * vcpu) [inline]
```

Return whether the entry reason was a page fault.

Parameters

<i>vcpu</i>	Pointer to vCPU area.
-------------	-----------------------

return 0 if not, !=0 otherwise.

References [L4_CV](#), and [L4_NOTHROW](#).

Referenced by [L4vcpu::Vcpu::is_page_fault_entry\(\)](#).

Here is the caller graph for this function:



14.18.2.7 l4vcpu_print_state()

```
void l4vcpu_print_state (
    const l4_vcpu_state_t * vcpu,
    const char * prefix)
```

Print the state of a vCPU.

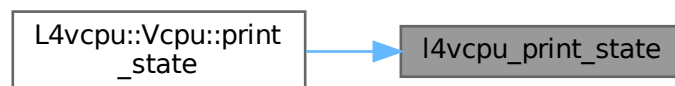
Parameters

<i>vcpu</i>	Pointer to vCPU area.
<i>prefix</i>	A prefix for each line printed.

References [L4_CV](#), [L4_INLINE](#), and [L4_NOTHROW](#).

Referenced by [L4vcpu::Vcpu::print_state\(\)](#).

Here is the caller graph for this function:



14.18.2.8 l4vcpu_wait_for_event()

```
void l4vcpu_wait_for_event (
    l4_vcpu_state_t * vcpu,
    l4_utcb_t * utcb,
    l4vcpu_event_hndl_t do_event_work_cb,
    l4vcpu_setup_ipc_t setup_ipc) [inline]
```

Wait for event.

Parameters

<i>vcpu</i>	Pointer to vCPU area.
<i>utcb</i>	UtcB pointer of the calling vCPU.
<i>do_event_work_cb</i>	Call-back function that is called when the vCPU awakes and needs to handle an event/↔ IRQ.
<i>setup_ipc</i>	Function call-back that is called right before any IPC operation.

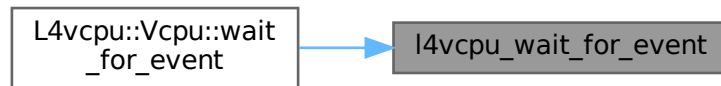
Note that event delivery remains disabled after this function returns.

Definition at line 270 of file [vcpu.h](#).

References [L4_IPC_NEVER](#), and [L4_NOTHROW](#).

Referenced by [L4vcpu::Vcpu::wait_for_event\(\)](#).

Here is the caller graph for this function:



14.18.3 Extended vCPU support

Extended vCPU handling functionality.

Collaboration diagram for Extended vCPU support:



Functions

- [int l4vcpu_ext_alloc](#) ([l4_vcpu_state_t](#) **vcpu, [l4_addr_t](#) *ext_state, [l4_cap_idx_t](#) task, [l4_cap_idx_t](#) regmgr)
[L4_NOTHROW](#)

Allocate state area for an extended vCPU.

14.18.3.1 Detailed Description

Extended vCPU handling functionality.

14.18.3.2 Function Documentation

14.18.3.2.1 l4vcpu_ext_alloc()

```

int l4vcpu_ext_alloc (
    l4_vcpu_state_t ** vcpu,
    l4_addr_t * ext_state,
    l4_cap_idx_t task,
    l4_cap_idx_t regmgr)
  
```

Allocate state area for an extended vCPU.

Parameters

out	<i>vcpu</i>	Allocated vcpu-state area.
out	<i>ext_state</i>	Allocated extended vcpu-state area.
	<i>task</i>	Task to use for allocation.
	<i>regmgr</i>	Region manager to use for allocation.

Returns

0 for success, error code otherwise

References [L4_CV](#), [L4_INLINE](#), and [L4_NOTHROW](#).

Chapter 15

Namespace Documentation

15.1 cxx Namespace Reference

Our C++ library.

Namespaces

- namespace [Bits](#)
Internal helpers for the cxx package.

Data Structures

- class [Avl_map](#)
AVL tree based associative container.
- class [Avl_set](#)
AVL set for simple comparable items.
- class [Avl_tree_node](#)
Node of an AVL tree.
- class [Avl_tree](#)
A generic AVL tree.
- class [Bitfield](#)
Definition for a member (part) of a bit field.
- class [Bitmap_base](#)
Basic bitmap abstraction.
- class [Bitmap](#)
A static bitmap.
- class [H_list_item_t](#)
Basic element type for a double-linked [H_list](#).
- class [H_list](#)
General double-linked list of unspecified [cxx::H_list_item](#) elements.
- struct [H_list_t](#)
Double-linked list of typed [H_list_item_t](#) elements.
- class [List_item](#)
Basic list item.

- class [List](#)
Doubly linked list, with internal allocation.
- class [List_alloc](#)
Standard list-based allocator.
- struct [Pair](#)
Pair of two values.
- class [Pair_first_compare](#)
Comparison functor for [Pair](#).
- class [Ref_ptr](#)
A reference-counting pointer with automatic cleanup.
- struct [Ref_obj_list_item](#)
Item for list linked via [cxx::Ref_ptr](#) with default reference counting.
- class [Base_slab](#)
Basic slab allocator.
- class [Slab](#)
Slab allocator for object of type `Type`.
- class [Base_slab_static](#)
Merged slab allocator (allocators for objects of the same size are merged together).
- class [Slab_static](#)
Merged slab allocator (allocators for objects of the same size are merged together).
- class [S_list](#)
Simple single-linked list.
- class [static_vector](#)
Simple encapsulation for a dynamically allocated array.
- class [Nothrow](#)
Helper type to distinguish the `operator new` version that does not throw exceptions.
- class [New_allocator](#)
Standard allocator based on `operator new ()`.
- struct [Lt_functor](#)
Generic comparator class that defaults to the less-than operator.
- class [String](#)
Allocation free string class with explicit length field.
- class [Weak_ref_base](#)
Generic (base) weak reference to some object.
- class [Weak_ref](#)
Typed weak reference to an object of type `T`.

Typedefs

- typedef [H_list_item_t](#)< void > [H_list_item](#)
Untyped list item.
- template<typename T>
using [Ref_ptr_list_item](#) = [Bits::Smart_ptr_list_item](#)<T, [cxx::Ref_ptr](#)<T> >
Item for list linked with [cxx::Ref_ptr](#).
- template<typename T>
using [Ref_ptr_list](#) = [Bits::Smart_ptr_list](#)<[Ref_ptr_list_item](#)<T> >
Single-linked list where elements are connected via a [cxx::Ref_ptr](#).
- template<typename T>
using [Unique_ptr_list_item](#) = [Bits::Smart_ptr_list_item](#)<T, [cxx::unique_ptr](#)<T> >
Item for list linked with [cxx::unique_ptr](#).
- template<typename T>
using [Unique_ptr_list](#) = [Bits::Smart_ptr_list](#)<[Unique_ptr_list_item](#)<T> >
Single-linked list where elements are connected with a [cxx::unique_ptr](#).

Functions

- `template<typename A, typename ... ARGS>`
`constexpr A const & min (A const &a1, A const &a2, ARGS const &...a)`
Get the minimum of a1 and a2 up to aN.
- `template<typename A, typename ... ARGS>`
`constexpr A const & min (cxx::identity_t< A > const &a1, cxx::identity_t< A > const &a2, ARGS const &...a)`
Get the minimum of a1 and a2 up to aN.
- `template<typename A, typename ... ARGS>`
`constexpr A const & max (A const &a1, A const &a2, ARGS const &...a)`
Get the maximum of a1 and a2 up to aN.
- `template<typename A, typename ... ARGS>`
`constexpr A const & max (cxx::identity_t< A > const &a1, cxx::identity_t< A > const &a2, ARGS const &...a)`
Get the maximum of a1 and a2 up to aN.
- `template<typename T1>`
`T1 clamp (T1 v, T1 lo, T1 hi)`
Limit v to the range given by lo and hi.
- `template<typename T>`
`constexpr T gcd (T a, T b)`
Compute the greatest common divisor of two unsigned values.
- `template<typename T>`
`constexpr T lcm (T a, T b)`
Compute the least common multiple of two unsigned values.
- `template<typename T>`
`T access_once (T const *a)`
Read the value at an address at most once.
- `template<typename T, typename VAL>`
`void write_now (T *a, VAL &&val)`
Write a value at an address exactly once.

15.1.1 Detailed Description

Our C++ library.

Small Low-Level C++ Library.

Strings.

Various kinds of C++ utilities.

15.1.2 Function Documentation

15.1.2.1 [access_once\(\)](#)

```
template<typename T>
T cxx::access_once (
    T const * a) [inline]
```

Read the value at an address at most once.

The read might be omitted if the result is not used by any code unless `typename` contains `volatile`. If the read operation has side effects and must not be omitted, use different means like [L4drivers::Mmio_register_block](#) or similar.

The compiler is disallowed to reuse a previous read at the same address, for example:

```
val1 = *a;
val2 = access_once(a); // compiler may not replace this by val2 = val1;
```

The compiler is also disallowed to repeat the read, for example:

```
val1 = access_once(a);
val2 = val1; // compiler may not replace this by val2 = *a;
```

The above implies that the compiler is also disallowed to move the read out of or into loops.

Note

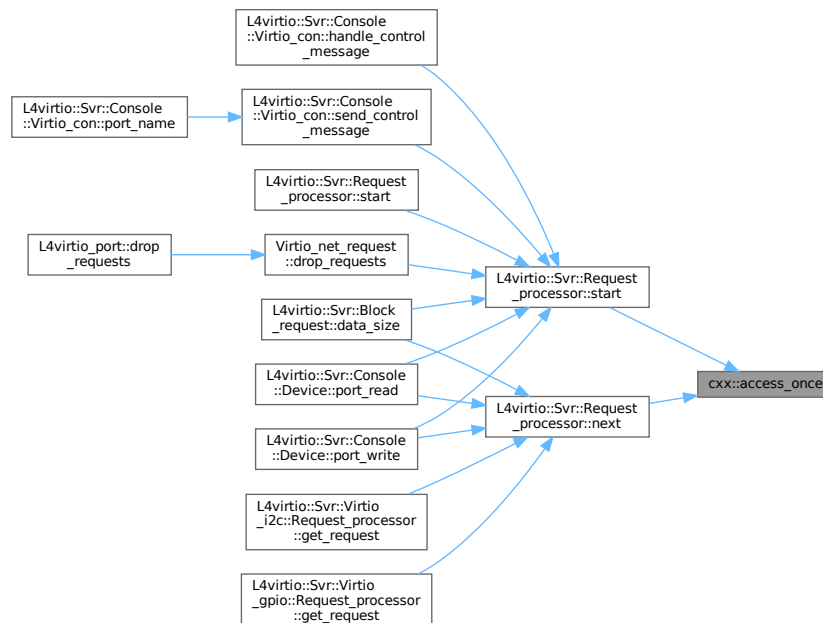
The read might still be moved relative to other code.

The value might be read from a hardware cache, not from RAM.

Definition at line 40 of file [utils](#).

Referenced by [L4virtio::Svr::Request_processor::next\(\)](#), and [L4virtio::Svr::Request_processor::start\(\)](#).

Here is the caller graph for this function:



15.1.2.2 gcd()

```
template<typename T>
T cxx::gcd (
    T a,
    T b) [constexpr]
```

Compute the greatest common divisor of two unsigned values.

This uses the Euclidean modulo algorithm.

Note

Contrary to the C++17 specification, this implementation requires the same unsigned type of both arguments and returns the same type (not a common type). This should be fine for most practical use cases.

Contrary to the C++17 specification, this implementation does not accept signed arguments and negative literals.

Template Parameters

<i>T</i>	Unsigned integer type of the values.
----------	--------------------------------------

Parameters

<i>a</i>	First input.
<i>b</i>	Second input.

Returns

Greatest common divisor of the input values.

Definition at line 34 of file [numeric](#).

Referenced by [lcm\(\)](#).

Here is the caller graph for this function:



15.1.2.3 lcm()

```
template<typename T>
T cxx::lcm (
    T a,
    T b) [constexpr]
```

Compute the least common multiple of two unsigned values.

This uses the greatest common divisor to compute the least common multiple.

Note

Contrary to the C++17 specification, this implementation requires the same unsigned type of both arguments and returns the same type (not a common type). This should be fine for most practical use cases.

Contrary to the C++17 specification, this implementation does not accept signed arguments and negative literals.

Template Parameters

<i>T</i>	Unsigned integer type of the values.
----------	--------------------------------------

Parameters

<i>a</i>	First input.
<i>b</i>	Second input.

Returns

Least common multiple of the input values.

Definition at line 68 of file [numeric](#).

References [gcd\(\)](#).

Here is the call graph for this function:



15.1.2.4 write_now()

```
template<typename T, typename VAL>
void cxx::write_now (
    T * a,
    VAL && val) [inline]
```

Write a value at an address exactly once.

The compiler is disallowed to skip the write, for example:

```
*a = val;
write_now(a, val); // compiler may not skip this line
```

The compiler is also disallowed to repeat the write.

The above implies that the compiler is also disallowed to move the write out of or into loops.

Note

The write might still be moved relative to other code.

The value might be written just to a hardware cache for the moment, not immediately to RAM.

Definition at line 71 of file [utils](#).

15.2 cxx::Bits Namespace Reference

Internal helpers for the cxx package.

Data Structures

- struct [Avl_map_get_key](#)
Key-getter for [Avl_map](#).
- struct [Avl_set_get_key](#)
Internal, key-getter for [Avl_set](#) nodes.
- class [Base_avl_set](#)
Internal: AVL set with internally managed nodes.
- class [Bst](#)
Basic binary search tree (BST).
- struct [Direction](#)
The direction to go in a binary search tree.
- class [Bst_node](#)
Basic type of a node in a binary search tree (BST).
- class [Basic_list](#)
Internal: Common functions for all head-based list implementations.
- class [Smart_ptr_list_item](#)
List item for an arbitrary item in a [Smart_ptr_list](#).
- class [Smart_ptr_list](#)
List of smart-pointer-managed objects.

15.2.1 Detailed Description

Internal helpers for the cxx package.

15.3 L4 Namespace Reference

[L4](#) low-level kernel interface.

Namespaces

- namespace [Typeid](#)
Definition of interface data-type helpers.
- namespace [lpc](#)
IPC related functionality.
- namespace [lpc_svr](#)
Helper classes for [L4::Server](#) instantiation.
- namespace [Types](#)
[L4](#) basic type helpers for C++.

Data Structures

- struct [Type_info](#)
Dynamic Type Information for [L4Re](#) Interfaces.
- struct [Kobject_typeid](#)
[Meta](#) object for handling access to type information of [Kobjects](#).
- struct [Kobject_typeid< void >](#)
Minimalistic ID for `void` interface.
- class [Kobject_t](#)
Helper class to create an [L4Re](#) interface class that is derived from a single base class.
- class [Kobject_2t](#)
Helper class to create an [L4Re](#) interface class that is derived from two base classes (see [L4::Kobject_t](#)).
- struct [Kobject_3t](#)
Helper class to create an [L4Re](#) interface class that is derived from three base classes (see [L4::Kobject_t](#)).
- struct [Kobject_demand](#)
Get the combined server-side resource requirements for all type `T...`
- struct [Proto_t](#)
Data type for defining protocol numbers.
- struct [Kobject_x](#)
Generic [Kobject](#) inheritance template.
- class [Vm](#)
Virtual machine host address space.
- class [Arm_smccc](#)
Wrapper for function calls that follow the ARM SMC/HVC calling convention.
- class [Cap](#)
C++ interface for capabilities.
- class [Cap_base](#)
Base class for all kinds of capabilities.
- struct [Epiface](#)

- Base class for interface implementations.*

 - struct [Epiface_t0](#)

Epiface mixin for generic Kobject-based interfaces.
 - struct [Irqep_t](#)

Epiface implementation for interrupt handlers.
 - class [Registry_iface](#)

Abstract interface for object registries.
 - struct [Epiface_t](#)

Epiface implementation for Kobject-based interface implementations.
 - class [Basic_registry](#)

This registry returns the corresponding server object based on the label of an [lpc_gate](#).
 - class [Server](#)

Basic server loop for handling client requests.
 - class [Debugger](#)

C++ kernel debugger API.
 - class [Exception](#)

Exception interface.
 - class [Factory](#)

C++ Factory interface, see [Factory](#) for the C interface.
 - class [Iommu](#)

Interface for IO-MMUs used for DMA remapping.
 - class [lpc_gate](#)

The C++ IPC gate interface, see [IPC-Gate API](#) for the C interface.
 - class [Irq_eoi](#)

Interface for sending an unmask message to an object.
 - struct [Triggerable](#)

Interface that allows an object to be triggered by some source.
 - class [Irq](#)

C++ *Irq* interface, see [IRQs](#) for the C interface.
 - class [Icu](#)

C++ *Icu* interface, see [Interrupt controller](#) for the C interface.
 - class [Kobject](#)

Base class for all kinds of kernel objects and remote objects, referenced by capabilities.
 - class [Meta](#)

Meta interface that shall be implemented by each [L4Re](#) object and gives access to the dynamic type information for [L4Re](#) objects.
 - class [Io_pager](#)

Io_pager interface.
 - class [Pager](#)

Pager interface including the [Io_pager](#) interface.
 - class [Platform_control](#)

L4 C++ interface for controlling platform-wide properties, see [Platform Control C API](#) for the C interface.
 - class [Rcv_endpoint](#)

Interface for kernel objects that allow to receive IPC from them.
 - class [Scheduler](#)

C++ interface of the [Scheduler](#) kernel object, see [Scheduler](#) for the C interface.
 - struct [Semaphore](#)

C++ Kernel-provided semaphore interface, see [Kernel-provided semaphore](#) for the C interface.
 - class [Smart_cap](#)

Smart capability class.
 - class [Task](#)

- C++ interface of the [Task](#) kernel object, see [Task](#) for the C interface.*
- class [Thread](#)

C++ [L4](#) kernel thread interface, see [Thread](#) for the C interface.
 - class [Thread_group](#)

C++ [L4](#) kernel thread group interface, see [Thread groups](#) for the C interface.
 - class [Vcon](#)

C++ [L4](#) [Vcon](#) interface, see [Virtual Console](#) for the C interface.
 - class [Poll_timeout_kipclock](#)

A polling timeout based on the [L4Re](#) clock.
 - class [Alloc_list](#)

A simple list-based allocator.
 - class [IOModifier](#)

Modifier class for the IO stream.
 - class [Exception_tracer](#)

Back-trace support for exceptions.
 - class [Base_exception](#)

Base class for all exceptions, thrown by the [L4Re](#) framework.
 - class [Runtime_error](#)

[Exception](#) for an abstract runtime error.
 - class [Out_of_memory](#)

[Exception](#) signalling insufficient memory.
 - class [Element_already_exists](#)

[Exception](#) for duplicate element insertions.
 - class [Unknown_error](#)

[Exception](#) for an unknown condition.
 - class [Element_not_found](#)

[Exception](#) for a failed lookup (element not found).
 - class [Invalid_capability](#)

Indicates that an invalid object was invoked.
 - class [Com_error](#)

Error conditions during IPC.
 - class [Bounds_error](#)

Access out of bounds.
 - class [Server_object](#)

Abstract server object to be used with [L4::Server](#) and [L4::Basic_registry](#).
 - struct [Server_object_t](#)

Base class (template) for server implementing server objects.
 - struct [Server_object_x](#)

*Helper class to implement *p_dispatch* based server objects.*
 - struct [Irq_handler_object](#)

[Server](#) object base class for handling IRQ messages.
 - class [Lock_guard](#)

Basic lock guard implementation that prevents forgotten unlocks on exit paths from a method or a block of code.
 - class [String](#)

A null-terminated string container class.
 - class [Poll_timeout_counter](#)

Evaluate an expression for a maximum number of times.
 - class [Uart_apb](#)

Driver for the Advanced Peripheral Bus (APB) UART from the Cortex-M System Design Kit (CMSDK).
 - class [Uart](#)

[Uart](#) driver abstraction.

Typedefs

- typedef int **Opcode**
Data type for RPC opcodes.

Enumerations

- enum { **PROTO_ANY** = 0 , **PROTO_EMPTY** = -19 }

Functions

- template<typename T>
Type_info const * **kobject_typeid** () noexcept
*Get the **L4::Type_info** for the **L4Re** interface given in **T**.*
- template<typename T, typename F>
Cap< T > **cap_dynamic_cast** (**Cap**< F > const &c) noexcept
dynamic_cast for capabilities.
- template<typename T, typename F>
Cap< T > **cap_cast** (**Cap**< F > const &c) noexcept
static_cast for capabilities.
- template<typename T, typename F>
Cap< T > **cap_reinterpret_cast** (**Cap**< F > const &c) noexcept
reinterpret_cast for capabilities.
- template<typename T>
constexpr T **trunc_order** (T val, unsigned char order)
Round a value down so the given number of lsb is zero.
- template<typename T>
constexpr T **round_order** (T val, unsigned char order)
Round a value up so the given number of lsb is zero.
- template<typename T, typename F, typename SMART>
Smart_cap< T, SMART > **cap_cast** (**Smart_cap**< F, SMART > const &c) noexcept
static_cast for (smart) capabilities.
- template<typename T, typename F, typename SMART>
Smart_cap< T, SMART > **cap_reinterpret_cast** (**Smart_cap**< F, SMART > const &c) noexcept
reinterpret_cast for (smart) capabilities.
- void **throw_ipc_exception** (**L4::Cap**< void > const &o, **l4_msgtag_t** const &err, **l4_utcb_t** *utcb)
*Throw an **L4** IPC error as exception.*
- void **throw_ipc_exception** (void const *o, **l4_msgtag_t** const &err, **l4_utcb_t** *utcb)
*Throw an **L4** IPC error as exception.*

Variables

- **IOModifier** const **hex**
Modifies the stream to print numbers as hexadecimal values.
- **IOModifier** const **dec**
Modifies the stream to print numbers as decimal values.
- **BasicOStream** **cout**
Standard output stream.
- **BasicOStream** **cerr**
Standard error stream.

15.3.1 Detailed Description

[L4](#) low-level kernel interface.

15.3.2 Enumeration Type Documentation

15.3.2.1 anonymous enum

anonymous enum

Enumerator

PROTO_ANY	Default protocol used by Kobject_t and Kobject_x .
PROTO_EMPTY	Empty protocol for empty APIs.

Definition at line 44 of file [__typeinfo.h](#).

15.3.3 Function Documentation

15.3.3.1 cap_cast() [1/2]

```
template<typename T, typename F>
Cap< T > L4::cap_cast (
    Cap< F > const & c) [inline], [noexcept]
```

static_cast for capabilities.

Template Parameters

<i>T</i>	The target type of the capability
<i>F</i>	The source type (and is usually implicitly set)

Parameters

<i>c</i>	The source capability that shall be casted
----------	--

Returns

A capability typed to the interface *T*.

The use of this cast operator is similar to the `static_cast<>()` for C++ pointers. It does the same type checking and adjustments like C++ does on pointers.

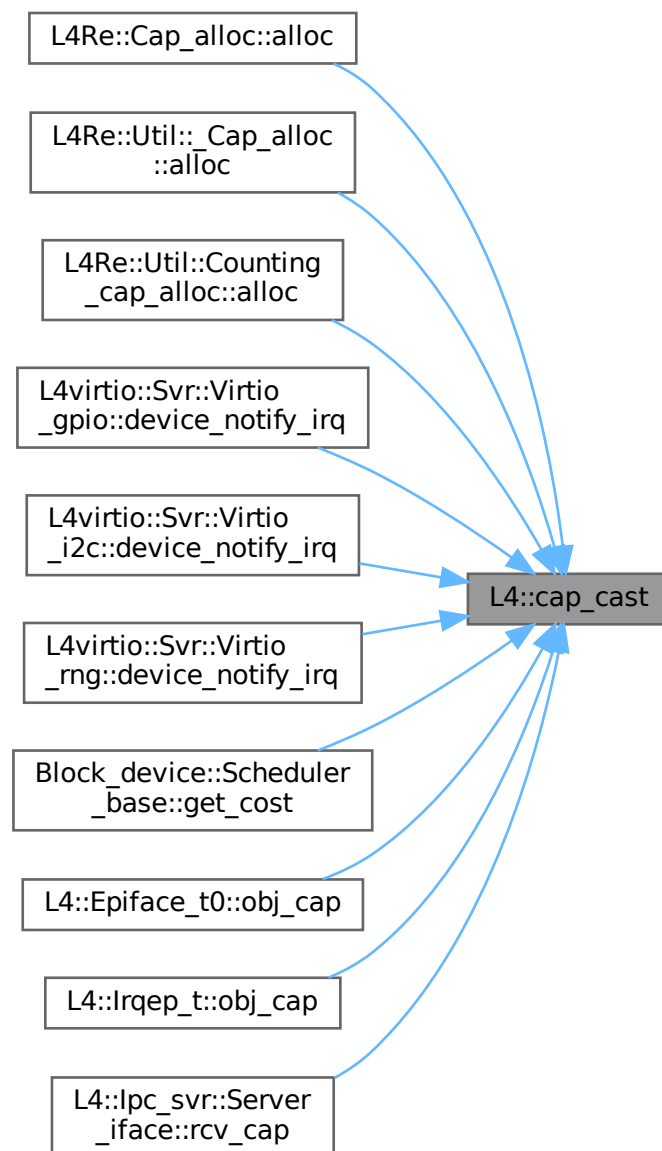
Example code:


```
L4::Cap<L4::Kobject> obj = ... ;
L4::Cap<L4::Icu> icu = L4::cap_cast<L4::Icu>(obj);
```

Definition at line 416 of file [capability.h](#).

Referenced by [L4Re::Cap_alloc::alloc\(\)](#), [L4Re::Util::_Cap_alloc::alloc\(\)](#), [L4Re::Util::Counting_cap_alloc< COUNTERTYPE, Dbg >::a](#), [L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::device_notify_irq\(\)](#), [L4virtio::Svr::Virtio_i2c< Request_handler, Epiface >::d](#), [L4virtio::Svr::Virtio_rng< Rnd_state, Epiface >::device_notify_irq\(\)](#), [Block_device::Scheduler_base< DEV >::get_cost\(\)](#), [L4::Epiface_t0< RPC_IFACE, BASE >::obj_cap\(\)](#), [L4::Irqep_t< Derived, BASE, bool >::obj_cap\(\)](#), and [L4::lpc_svr::Server_iface::rcv](#).

Here is the caller graph for this function:



15.3.3.2 `cap_cast()` [2/2]

```
template<typename T, typename F, typename SMART>
Smart_cap< T, SMART > L4::cap_cast (
    Smart_cap< F, SMART > const & c) [inline], [noexcept]
```

`static_cast` for (smart) capabilities.

Template Parameters

<i>T</i>	Type to cast the capability to.
<i>F</i>	(implicit) Type of the passed capability.
<i>SMART</i>	(implicit) Class implementing the Smart_cap interface.

Parameters

<i>c</i>	Capability to be casted.
----------	--------------------------

Returns

A smart capability with new type *T*.

Definition at line [192](#) of file [smart_capability](#).

15.3.3.3 `cap_dynamic_cast()`

```
template<typename T, typename F>
Cap< T > L4::cap_dynamic_cast (
    Cap< F > const & c) [inline], [noexcept]
```

`dynamic_cast` for capabilities.

Template Parameters

<i>T</i>	The target type of the capability.
<i>F</i>	The source type (is usually implicitly set).

Parameters

<i>c</i>	The source capability that shall be casted.
----------	---

Return values

Cap<T>	Capability of target interface <i>T</i> .
------------------------------	---

<code>L4_INVALID_CAP</code>	<code>c</code> does not support the target interface <code>T</code> or the <code>L4::Meta</code> interface.
-----------------------------	---

The use of this cast operator is similar to the `dynamic_cast<>()` for C++ pointers. It also induces overhead, because it uses the meta interface (`L4::Meta`) to do runtime type checking.

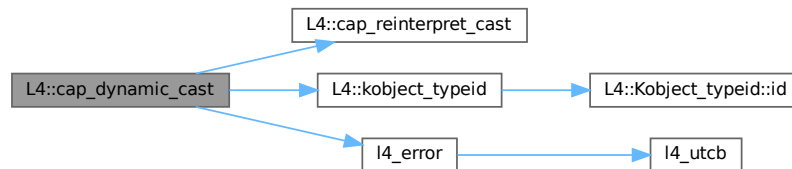
Example code:

```
L4::Cap<L4::Kobject> obj = ... ;
L4::Cap<L4::Icu> icu = L4::cap_dynamic_cast<L4::Icu>(obj);
```

Definition at line 116 of file [capability](#).

References [cap_reinterpret_cast\(\)](#), [L4::Cap_base::Invalid](#), [kobject_typeid\(\)](#), and [l4_error\(\)](#).

Here is the call graph for this function:



15.3.3.4 cap_reinterpret_cast() [1/2]

```
template<typename T, typename F>
Cap< T > L4::cap_reinterpret_cast (
    Cap< F > const & c) [inline], [noexcept]
```

`reinterpret_cast` for capabilities.

Template Parameters

<i>T</i>	The target type of the capability
<i>F</i>	The source type (and is usually implicitly set)

Parameters

<i>c</i>	The source capability that shall be casted
----------	--

Returns

A capability typed to the interface `T`.

The use of this cast operator is similar to the `reinterpret_cast<>()` for C++ pointers. It does not do any type checking or type adjustment.

Example code:

```
L4::Cap<L4::Kobject> obj = ... ;
L4::Cap<L4::Icu> icu = L4::cap_reinterpret_cast<L4::Icu>(obj);
```

Definition at line 447 of file [capability.h](#).

Referenced by [cap_dynamic_cast\(\)](#).

Here is the caller graph for this function:

**15.3.3.5 cap_reinterpret_cast() [2/2]**

```
template<typename T, typename F, typename SMART>
Smart_cap< T, SMART > L4::cap_reinterpret_cast (
    Smart_cap< F, SMART > const & c) [inline], [noexcept]
```

`reinterpret_cast` for (smart) capabilities.

Template Parameters

<i>T</i>	Type to cast the capability to.
<i>F</i>	(implicit) Type of the passed capability.
<i>SMART</i>	(implicit) Class implementing the Smart_cap interface.

Parameters

<i>c</i>	Capability to be casted.
----------	--------------------------

Returns

A smart capability with new type `T`.

Definition at line 211 of file [smart_capability](#).

15.3.3.6 round_order()

```
template<typename T>
T L4::round_order (
    T val,
    unsigned char order) [constexpr]
```

Round a value up so the given number of lsb is zero.

Template Parameters

<i>T</i>	The type of the value (shall be some integral type).
----------	--

Parameters

<i>val</i>	The value to round up to the next multiple of 2^{order} .
<i>order</i>	order (2^{order}) to round up to.

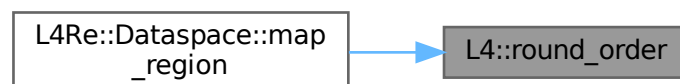
Returns

val rounded up to the next 2^{order} .

Definition at line 32 of file [consts](#).

Referenced by [L4Re::Dataspace::map_region\(\)](#).

Here is the caller graph for this function:



15.3.3.7 trunc_order()

```
template<typename T>
T L4::trunc_order (
    T val,
    unsigned char order) [constexpr]
```

Round a value down so the given number of lsb is zero.

Template Parameters

<i>T</i>	The type of the value (shall be some integral type).
----------	--

Parameters

<i>val</i>	The value where the given lsb shall be masked.
<i>order</i>	the number of least significant bits (lsb) to mask.

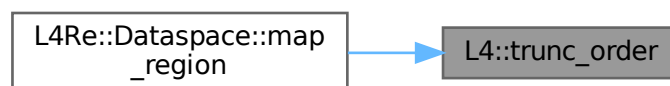
Returns

val with order lsb masked to zero.

Definition at line 18 of file [consts](#).

Referenced by [L4Re::Dataspace::map_region\(\)](#).

Here is the caller graph for this function:



15.4 L4::lpc Namespace Reference

IPC related functionality.

Namespaces

- namespace [Msg](#)
IPC Message related functionality.

Data Structures

- struct [Array_ref](#)
Array reference data type for arrays located in the message.
- struct [Array](#)
Array data type for dynamically sized arrays in RPCs.
- struct [Array_in_buf](#)
Server-side copy in buffer for Array.
- struct [Call](#)
RPC attribute for a standard RPC call.

- struct [Call_zero_send_timeout](#)
RPC attribute for an RPC call, with zero send timeout.
- struct [Call_t](#)
RPC attribute for an RPC call with required rights.
- struct [Send_only](#)
RPC attribute for a send-only RPC.
- struct [Ret_array](#)
Dynamically sized output array of type T.
- struct [Out](#)
Mark an argument as a output value in an RPC signature.
- struct [In_out](#)
Mark an argument as in-out argument.
- struct [As_value](#)
Pass the argument as plain data value.
- struct [Opt](#)
Attribute for defining an optional RPC argument.
- class [Small_buf](#)
A receive item for receiving a single object capability.
- class [Gen_fpage](#)
Generic RPC base for typed message items.
- class [Snd_fpage](#)
Send item or return item.
- class [Rcv_fpage](#)
Non-small receive item.
- class [Cap](#)
Capability type for RPC interfaces (see [L4::Cap<T>](#)).
- class [Varg](#)
Variably sized RPC argument.
- class [Varg_list](#)
Self-contained list of variable-sized RPC parameters.
- class [Varg_list_ref](#)
List of variable-sized RPC parameters as received by the server.
- class [Str_cp_in](#)
Abstraction for extracting a zero-terminated string from an [lpc::lstream](#).
- class [Msg_ptr](#)
Pointer to an element of type T in an [lpc::lstream](#).
- class [lstream](#)
Input stream for IPC unmarshalling.
- class [Ostream](#)
Output stream for IPC marshalling.
- class [lostream](#)
Input/Output stream for IPC [un]marshalling.

Typedefs

- typedef unsigned short **Array_len_default**
Default type for passing length of an array.

Functions

- template<typename T>
Cap< T > **make_cap** (L4::Cap< T > cap, unsigned rights) noexcept
Make an L4::lpc::Cap<T> for the given capability and rights.
- template<typename T>
Cap< T > **make_cap_rw** (L4::Cap< T > cap) noexcept
Make an L4::lpc::Cap<T> for the given capability with L4_CAP_FPAGE_RW rights.
- template<typename T>
Cap< T > **make_cap_rws** (L4::Cap< T > cap) noexcept
Make an L4::lpc::Cap<T> for the given capability with L4_CAP_FPAGE_RWS rights.
- template<typename T>
Cap< T > **make_cap_full** (L4::Cap< T > cap) noexcept
Make an L4::IPC::Cap<T> for the given capability with full fpage and object-specific rights.
- template<typename T>
Internal::Buf_cp_out< T > **buf_cp_out** (T const *v, unsigned long size)
Insert an array into an lpc::Ostream.
- template<typename T>
Internal::Buf_cp_in< T > **buf_cp_in** (T *v, unsigned long &size)
Extract an array from an lpc::Istream.
- template<typename T>
Str_cp_in< T > **str_cp_in** (T *v, unsigned long &size)
Create a Str_cp_in for the given values.
- template<typename T>
Msg_ptr< T > **msg_ptr** (T *&p)
Create an Msg_ptr to adjust the given pointer.
- template<typename T>
Internal::Buf_in< T > **buf_in** (T *&v, unsigned long &size)
Return a pointer to stream array data.
- template<typename T>
T read (Istream &s)
Read a value out of a stream.

15.4.1 Detailed Description

IPC related functionality.

15.4.2 Function Documentation

15.4.2.1 buf_cp_in()

```
template<typename T>
Internal::Buf_cp_in< T > L4::Ipc::buf_cp_in (
    T * v,
    unsigned long & size)
```

Extract an array from an lpc::Istream.

Parameters

	<i>v</i>	Pointer to the array that shall receive the values from the lpc::lstream .
<i>in, out</i>	<i>size</i>	Input: the number of elements the array can take at most Output: the number of elements found in the stream.

[buf_cp_in\(\)](#) can be used to extract an array from an [lpc::lstream](#). This is the counterpart [buf_cp_out\(\)](#). The data from the received message is thereby copied to the given buffer and size is set to the number of elements found in the stream. To avoid the copy operation [buf_in\(\)](#) may be used instead.

See also

[buf_in\(\)](#) and [buf_cp_out\(\)](#).

Definition at line 159 of file [ipc_stream](#).

15.4.2.2 buf_cp_out()

```
template<typename T>
Internal::Buf_cp_out< T > L4::Ipc::buf_cp_out (
    T const * v,
    unsigned long size)
```

Insert an array into an [lpc::Ostream](#).

Parameters

<i>v</i>	Pointer to the array that shall be inserted into an lpc::Ostream .
<i>size</i>	Number of elements in the array.

This function inserts an array (e.g. a string) into an [lpc::Ostream](#). The data is copied to the stream. On insertion into the [lpc::Ostream](#) exactly the given number of elements of type T are copied to the message buffer, this means the source buffer is no longer referenced after insertion into the stream.

See also

The counterpart is either [buf_cp_in\(\)](#) or [buf_in\(\)](#).

Definition at line 100 of file [ipc_stream](#).

15.4.2.3 buf_in()

```
template<typename T>
Internal::Buf_in< T > L4::Ipc::buf_in (
    T *& v,
    unsigned long & size)
```

Return a pointer to stream array data.

Parameters

out	<i>v</i>	Pointer to the array within the lpc::lstream .
out	<i>size</i>	The number of elements found in the stream.

This routine provides a possibility to extract an array from an [lpc::lstream](#), without extra copy overhead. In contrast to [buf_cp_in\(\)](#) the data is not copied to a buffer, but a pointer to the array is returned. The user must make sure the UTCB is not used for other purposes while the returned pointer is still in use.

The mechanism is comparable to that of [Msg_ptr](#), however it handles arrays inserted with [buf_cp_out\(\)](#).

See also

[buf_cp_in\(\)](#) and [buf_cp_out\(\)](#).

Definition at line 310 of file [ipc_stream](#).

15.4.2.4 make_cap()

```
template<typename T>
Cap< T > L4::Ipc::make_cap (
    L4::Cap< T > cap,
    unsigned rights) [noexcept]
```

Make an [L4::lpc::Cap<T>](#) for the given capability and rights.

Template Parameters

<i>T</i>	(IMPLICIT) type of the referenced interface
----------	---

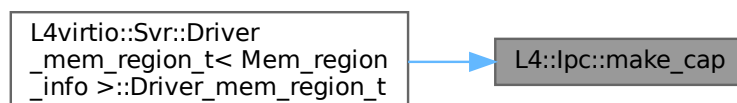
Parameters

<i>cap</i>	source capability (L4::Cap<T>)
<i>rights</i>	rights mask that shall be applied on transfer.

Definition at line 785 of file [ipc_types](#).

Referenced by [L4virtio::Svr::Driver_mem_region_t< Mem_region_info >::Driver_mem_region_t\(\)](#).

Here is the caller graph for this function:



15.4.2.5 make_cap_full()

```
template<typename T>
Cap< T > L4::Ipc::make_cap_full (
    L4::Cap< T > cap) [noexcept]
```

Make an L4::IPC::Cap<T> for the given capability with full fpage and object-specific rights.

Template Parameters

<i>T</i>	(implicit) type of the referenced interface
----------	---

Parameters

<i>cap</i>	source capability (L4::Cap<T>)
------------	--------------------------------

See also

[L4_cap_fpage_rights](#)

[L4_obj_fpage_ctl](#)

Note

Full rights (including object-specific rights) are required when mapping an IPC gate where the receiver should become the server, i.e. where the receiver wants to call [L4::ipc_gate::bind_thread\(\)](#) or [L4::ipc_gate::bind_snd_destination\(\)](#).

Definition at line 824 of file [ipc_types](#).

References [L4_CAP_FPAGE_RWSD](#), and [L4_FPAGE_C_OBJ_RIGHTS](#).

15.4.2.6 make_cap_rw()

```
template<typename T>
Cap< T > L4::Ipc::make_cap_rw (
    L4::Cap< T > cap) [noexcept]
```

Make an L4::ipc::Cap<T> for the given capability with [L4_CAP_FPAGE_RW](#) rights.

Template Parameters

<i>T</i>	(IMPLICIT) type of the referenced interface
----------	---

Parameters

<i>cap</i>	source capability (L4::Cap<T>)
------------	--------------------------------

Examples

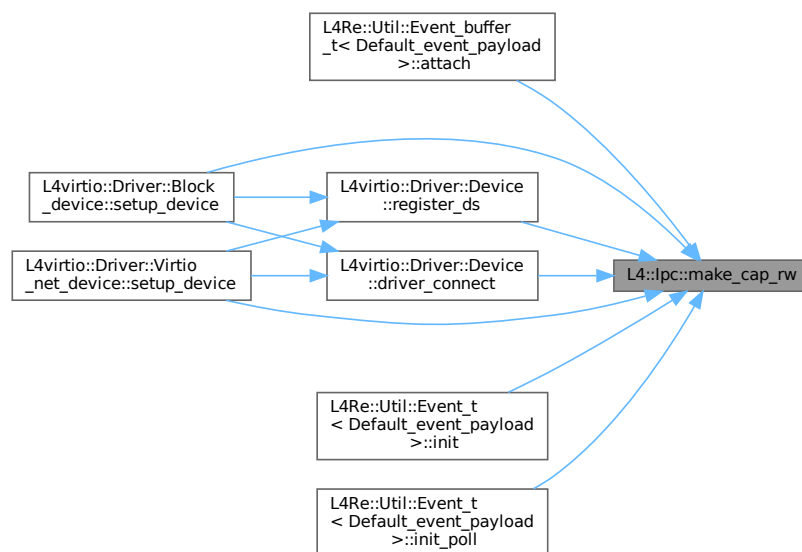
[examples/libs/l4re/c++/mem_alloc/ma+rm.cc](#), [examples/libs/l4re/c++/shared_ds/ds_clnt.cc](#), and [examples/libs/l4re/c++/shared_](#)

Definition at line 795 of file [ipc_types](#).

References [L4_CAP_FPAGE_RW](#).

Referenced by [L4Re::Util::Event_buffer_t< Default_event_payload >::attach\(\)](#), [L4virtio::Driver::Device::driver_connect\(\)](#), [L4Re::Util::Event_t< Default_event_payload >::init\(\)](#), [L4Re::Util::Event_t< Default_event_payload >::init_poll\(\)](#), [L4virtio::Driver::Device::register_ds\(\)](#), [L4virtio::Driver::Block_device::setup_device\(\)](#), and [L4virtio::Driver::Virtio_net_device::setup_de](#)

Here is the caller graph for this function:



15.4.2.7 make_cap_rws()

```

template<typename T>
Cap< T > L4::Ipc::make_cap_rws (
    L4::Cap< T > cap) [noexcept]

```

Make an `L4::ipc::Cap<T>` for the given capability with `L4_CAP_FPAGE_RWS` rights.

Template Parameters

<code>T</code>	(IMPLICIT) type of the referenced interface
----------------	---

Parameters

<i>cap</i>	source capability (L4::Cap<T>)
------------	--

Definition at line 805 of file [ipc_types](#).

References [L4_CAP_FPAGE_RWS](#).

15.4.2.8 msg_ptr()

```
template<typename T>
Msg_ptr< T > L4::IpC::msg_ptr (
    T *& p)
```

Create an [Msg_ptr](#) to adjust the given pointer.

This function makes it more convenient to extract pointers to data in the message buffer itself from an [IpC::Istream](#). This may be used to avoid copy out of large data structures. (See [Msg_ptr](#).)

Definition at line 252 of file [ipc_stream](#).

15.4.2.9 read()

```
template<typename T>
T L4::IpC::read (
    Istream & s) [inline]
```

Read a value out of a stream.

Parameters

<i>s</i>	An Istream .
----------	------------------------------

Returns

The value of type *T*.

The stream position is progressed accordingly.

Definition at line 1289 of file [ipc_stream](#).

15.4.2.10 str_cp_in()

```
template<typename T>
Str_cp_in< T > L4::IpC::str_cp_in (
    T * v,
    unsigned long & size)
```

Create a [Str_cp_in](#) for the given values.

Parameters

	<i>v</i>	Pointer to the array that shall receive the values from the lpc::lstream .
<i>in, out</i>	<i>size</i>	Input: the number of elements the array can take at most Output: the number of elements found in the stream.

This function makes it more convenient to extract arrays from an [lpc::lstream](#) (

See also

[Str_cp_in](#).)

Definition at line 213 of file [ipc_stream](#).

15.5 L4::lpc::Msg Namespace Reference

IPC Message related functionality.

Data Structures

- struct [Elem](#)< [Array](#)< A, LEN > >
Array as input arguments.
- struct [Elem](#)< [Array](#)< A, LEN > & >
Array as output argument.
- struct [Elem](#)< [Array_ref](#)< A, LEN > & >
Array_ref as output argument.
- struct [Dir_in](#)
Marker type for input values.
- struct [Dir_out](#)
Marker type for output values.
- struct [Cls_data](#)
Marker type for data values.
- struct [Cls_item](#)
Marker type for item values.
- struct [Cls_buffer](#)
Marker type for receive buffer values.
- struct [Do_in_data](#)
Marker for Input data.
- struct [Do_out_data](#)
Marker for Output data.
- struct [Do_in_items](#)
Marker for Input items.
- struct [Do_out_items](#)
Marker for Output items.
- struct [Do_rcv_buffers](#)
Marker for receive buffers.
- struct [Svr_val_ops](#)
Defines client-side handling of 'MTYPE' as RPC argument.
- struct [Is_valid_rpc_type](#)

Type trait defining a valid RPC parameter type.

- struct [Svr_arg_pack](#)

Server-side RPC arguments data structure used to provide arguments to the server-side implementation of an RPC function.

- struct [True](#)

True meta value.

- struct [False](#)

False meta value.

Enumerations

- enum {
[Word_bytes](#) = sizeof(l4_umword_t) , [Item_words](#) = 2 , [Item_bytes](#) = Word_bytes * Item_words , [Mr_words](#) = L4_UTCB_GENERIC_DATA_SIZE ,
[Mr_bytes](#) = Word_bytes * Mr_words , [Br_bytes](#) = Word_bytes * L4_UTCB_GENERIC_BUFFERS_SIZE }

Functions

- constexpr unsigned long [align_to](#) (unsigned long bytes, unsigned long align) noexcept
Pad bytes to the given alignment align (in bytes).
- template<typename T>
 constexpr unsigned long [align_to](#) (unsigned long bytes) noexcept
Pad bytes to the alignment of the type T.
- template<typename T>
 constexpr bool [check_size](#) (unsigned offset, unsigned limit) noexcept
Check if there is enough space for T from offset to limit.
- template<typename T, typename CTYPE>
 bool [check_size](#) (unsigned offset, unsigned limit, CTYPE cnt) noexcept
Check if there is enough space for an array of T from offset to limit.
- template<typename T>
 int [msg_add](#) (char *msg, unsigned offs, unsigned limit, T v) noexcept
Add some data to a message at offs.
- template<typename T>
 int [msg_get](#) (char *msg, unsigned offs, unsigned limit, T &v) noexcept
Get some data from a message at offs.

15.5.1 Detailed Description

IPC Message related functionality.

15.5.2 Enumeration Type Documentation

15.5.2.1 anonymous enum

anonymous enum

Enumerator

Word_bytes	number of bytes for one message word
Item_words	number of message words for one message item
Item_bytes	number of bytes for one message item
Mr_words	number of message words available in the UTCB
Mr_bytes	number of bytes available in the UTCB message registers
Br_bytes	number of bytes available in the UTCB buffer registers

Definition at line 85 of file [ipc_basics](#).

15.5.3 Function Documentation

15.5.3.1 `align_to()` [1/2]

```
template<typename T>
unsigned long L4::Ipc::Msg::align_to (
    unsigned long bytes) [constexpr], [noexcept]
```

Pad *bytes* to the alignment of the type *T*.

Template Parameters

<i>T</i>	The data type used for the alignment
----------	--------------------------------------

Parameters

<i>bytes</i>	The value to add the padding to
--------------	---------------------------------

Returns

bytes padded to achieve the alignment of *T*.

Definition at line 40 of file [ipc_basics](#).

References [align_to\(\)](#).

Here is the call graph for this function:



15.5.3.2 align_to() [2/2]

```
unsigned long L4::Ipc::Msg::align_to (
    unsigned long bytes,
    unsigned long align) [constexpr], [noexcept]
```

Pad bytes to the given alignment *align* (in bytes).

Parameters

<i>bytes</i>	The input value in bytes
<i>align</i>	The alignment value in bytes

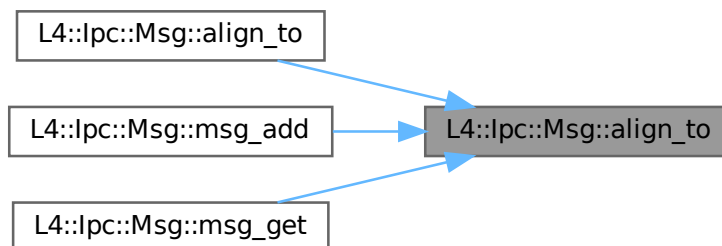
Returns

the result after padding *bytes* to *align*.

Definition at line 30 of file [ipc_basics](#).

Referenced by [align_to\(\)](#), [msg_add\(\)](#), and [msg_get\(\)](#).

Here is the caller graph for this function:



15.5.3.3 check_size() [1/2]

```
template<typename T>
bool L4::Ipc::Msg::check_size (
    unsigned offset,
    unsigned limit) [constexpr], [noexcept]
```

Check if there is enough space for T from offset to limit.

Template Parameters

<i>T</i>	The data type that shall be fitted at <i>offset</i>
----------	---

Parameters

<i>offset</i>	The current offset in bytes (must already be padded if desired).
<i>limit</i>	The limit in bytes that must not be exceeded after adding the size of <i>T</i> .

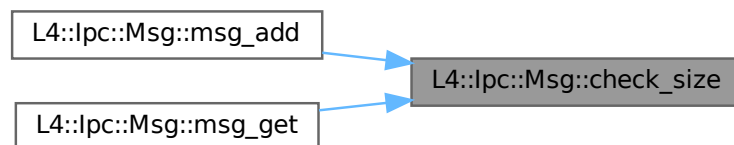
Returns

true if the limit will not be exceeded, false else.

Definition at line 53 of file [ipc_basics](#).

Referenced by [msg_add\(\)](#), and [msg_get\(\)](#).

Here is the caller graph for this function:



15.5.3.4 check_size() [2/2]

```

template<typename T, typename CTYPE>
bool L4::Ipc::Msg::check_size (
    unsigned offset,
    unsigned limit,
    CTYPE cnt) [inline], [noexcept]
  
```

Check if there is enough space for an array of T from offset to limit.

Template Parameters

<i>T</i>	The data type that shall be fitted at <i>offset</i>
<i>CTYPE</i>	Type of the <i>cnt</i> parameter

Parameters

<i>offset</i>	The current offset in bytes (must already be padded if desired).
<i>limit</i>	The limit in bytes that must not be exceeded after adding <i>cnt</i> times the size of <i>T</i> .
<i>cnt</i>	The number of elements of type <i>T</i> that shall be put at <i>offset</i> .

Returns

true if the limit will not be exceeded, false else.

Definition at line 71 of file [ipc_basics](#).

References [L4_UNLIKELY](#).

15.5.3.5 msg_add()

```
template<typename T>
int L4::IpC::Msg::msg_add (
    char * msg,
    unsigned offs,
    unsigned limit,
    T v) [inline], [noexcept]
```

Add some data to a message at offs.

Template Parameters

<i>T</i>	The type of the data to add
----------	-----------------------------

Parameters

<i>msg</i>	pointer to the start of the message
<i>offs</i>	The current offset within the message, this shall be padded to the alignment of <i>T</i> if <i>v</i> is added.
<i>limit</i>	The size limit in bytes that offset must not exceed.
<i>v</i>	The value to add to the message

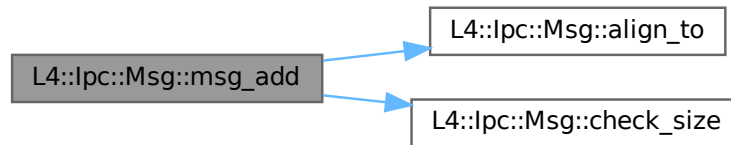
Returns

The new offset when successful, a negative value if the given limit will be exceeded.

Definition at line 114 of file [ipc_basics](#).

References [align_to\(\)](#), [check_size\(\)](#), [L4_MSGTOOLONG](#), and [L4_UNLIKELY](#).

Here is the call graph for this function:



15.5.3.6 msg_get()

```

template<typename T>
int L4::Ipc::Msg::msg_get (
    char * msg,
    unsigned offs,
    unsigned limit,
    T & v) [inline], [noexcept]
  
```

Get some data from a message at offs.

Template Parameters

<i>T</i>	The type of the data to read
----------	------------------------------

Parameters

<i>msg</i>	Pointer to the start of the message
<i>offs</i>	The current offset within the message, this shall be padded to the alignment of <i>T</i> if a <i>v</i> can be read.
<i>limit</i>	The size limit in bytes that offset must not exceed.
<i>v</i>	A reference to receive the value from the message

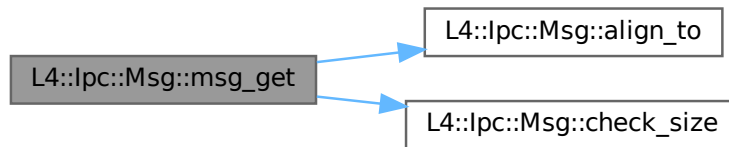
Returns

The new offset when successful, a negative value if the given limit will be exceeded.

Definition at line 135 of file `ipc_basics`.

References [align_to\(\)](#), [check_size\(\)](#), [L4_EMSGTOOSHORT](#), and [L4_UNLIKELY](#).

Here is the call graph for this function:



15.6 L4::lpc_svr Namespace Reference

Helper classes for [L4::Server](#) instantiation.

Data Structures

- class [Server_iface](#)
Interface for server-loop related functions.
- struct [Ignore_errors](#)
Mix in for LOOP_HOOKS to ignore IPC errors.
- struct [Default_timeout](#)
Mix in for LOOP_HOOKS to use a 0 send and a infinite receive timeout.
- struct [Compound_reply](#)
Mix in for LOOP_HOOKS to always use compound reply and wait.
- struct [Default_setup_wait](#)
Mix in for LOOP_HOOKS for setup_wait no op.
- struct [Direct_dispatch](#)
Direct dispatch helper, for forwarding dispatch calls to a registry R.
- struct [Direct_dispatch< R * >](#)
Direct dispatch helper, for forwarding dispatch calls via a pointer to a registry R.
- struct [Exc_dispatch](#)
Dispatch helper wrapping try {} catch {} around the dispatch call.
- struct [Dbg_dispatch](#)
Dispatch helper that, in addition to what [Exc_dispatch](#) does, prints exception messages.
- class [Br_manager_no_buffers](#)
Empty implementation of [Server_iface](#).
- struct [Default_loop_hooks](#)
Default LOOP_HOOKS.
- class [Timeout](#)
Callback interface for [Timeout_queue](#).
- class [Timeout_queue](#)
[Timeout](#) queue to be used in l4re server loop.
- class [Timeout_queue_hooks](#)
Loop hooks mixin for integrating a timeout queue into the server loop.

Enumerations

- enum [Reply_mode](#) { [Reply_compound](#) , [Reply_separate](#) }
Reply mode for server loop.

15.6.1 Detailed Description

Helper classes for [L4::Server](#) instantiation.

15.7 L4::Typeid Namespace Reference

Definition of interface data-type helpers.

Data Structures

- struct [P_dispatch](#)
Use for protocol based dispatch stage.
- struct [Raw_ipc](#)
RPCs list for passing raw incoming IPC to the server object.
- struct [Rpc](#)
Standard list of RPCs of an interface.
- struct [Rpc_code](#)
List of RPCs of an interface using a special opcode type.
- struct [Rpc_nocode](#)
List of RPCs of an interface using a single operation without an opcode.
- struct [Rpc_sys](#)
List of RPCs typically used for kernel interfaces.

15.7.1 Detailed Description

Definition of interface data-type helpers.

Note

These type helpers are intended for internal use, if you look for standard C++ type traits use the `<type_traits>` header for the standard C++ library or use `<l4/cxx/type_traits>`.

15.8 L4::Types Namespace Reference

[L4](#) basic type helpers for C++.

Data Structures

- class [Flags](#)
Template for defining typical [Flags](#) bitmaps.
- struct [Int_for_size](#)
Metafunction to get an unsigned integral type for the given size.
- struct [Int_for_type](#)
Metafunction to get an integral type of the same size as T .
- struct [Flags_ops_t](#)
Mixin class to define a set of friend bitwise operators on DT .
- struct [Flags_t](#)
Template type to define a flags type with bitwise operations.
- struct [Bool](#)
Boolean meta type.
- struct [False](#)
[False](#) meta value.
- struct [True](#)
[True](#) meta value.
- struct [Same](#)
Compare two data types for equality.

15.8.1 Detailed Description

[L4](#) basic type helpers for C++.

15.9 L4Re Namespace Reference

[L4Re](#) C++ Interfaces.

Namespaces

- namespace [Vfs](#)
Virtual file system for interfaces in POSIX libc.
- namespace [Util](#)
Documentation of the [L4](#) Runtime Environment utility functionality in C++.

Data Structures

- class [Cap_alloc](#)
Capability allocator interface.
- class [Smart_cap_auto](#)
Helper for [Unique_cap](#) and [Unique_del_cap](#).
- class [Smart_count_cap](#)
Helper for [Ref_cap](#) and [Ref_del_cap](#).
- class [Console](#)
[Console](#) class.
- class [Dataspace](#)

- *Interface for memory-like objects.*
- class [Debug_obj](#)
 - *Debug interface.*
- class [Dma_space](#)
 - *Managed DMA Address Space.*
- class [Env](#)
 - *C++ interface of the initial environment that is provided to an [L4](#) task.*
- class [Event](#)
 - *Event class.*
- struct [Default_event_payload](#)
 - *Default event stream payload.*
- class [Event_buffer_t](#)
 - *Event buffer class.*
- class [Inhibitor](#)
 - *Set of inhibitor locks, which inhibit specific actions when held.*
- class [Itas](#)
 - *Interface to the ITAS.*
- class [Log](#)
 - *Log interface class.*
- class [Mem_alloc](#)
 - *Memory allocation interface.*
- struct [Mmio_space](#)
 - *Interface for memory-like address space accessible via IPC.*
- class [Namespace](#)
 - *Name-space interface.*
- class [Parent](#)
 - *Parent interface.*
- struct [Random](#)
 - *Low-bandwidth interface for random number generators.*
- class [Rm](#)
 - *Region map.*

Typedefs

- `template<typename T>`
`using Shared_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_ALL_SPACES>>`
Shared capability that implements automatic free and unmap of the capability selector.
- `template<typename T>`
`using shared_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_ALL_SPACES>>`
Shared capability that implements automatic free and unmap of the capability selector.
- `template<typename T>`
`using Shared_del_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_DELETE_OBJ>>`
Shared capability that implements automatic free and unmap+delete of the capability selector.
- `template<typename T>`
`using shared_del_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_DELETE_OBJ>>`
Shared capability that implements automatic free and unmap+delete of the capability selector.
- `template<typename T>`
`using Unique_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_ALL_SPACES>>`
Unique capability that implements automatic free and unmap of the capability selector.
- `template<typename T>`
`using unique_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_ALL_SPACES>>`

Unique capability that implements automatic free and unmap of the capability selector.

- template<typename T>
using [Unique_del_cap](#) = L4::Detail::Unique_cap_impl<T, [Smart_cap_auto](#)<[L4_FP_DELETE_OBJ](#)>>

Unique capability that implements automatic free and unmap+delete of the capability selector.

- template<typename T>
using [unique_del_cap](#) = L4::Detail::Unique_cap_impl<T, [Smart_cap_auto](#)<[L4_FP_DELETE_OBJ](#)>>

Unique capability that implements automatic free and unmap+delete of the capability selector.

Functions

- void [throw_error](#) (long err, char const *extra="")
Generate C++ exception.
- long [chksys](#) (long err, char const *extra="", long ret=0)
Generate C++ exception on error.
- long [chksys](#) ([l4_msgtag_t](#) const &t, char const *extra="", [l4_utcb_t](#) *utcb=[l4_utcb](#)(), long ret=0)
Generate C++ exception on error.
- long [chksys](#) ([l4_msgtag_t](#) const &t, [l4_utcb_t](#) *utcb, char const *extra="")
Generate C++ exception on error.
- template<typename T>
[T chkcap](#) (T &&cap, char const *extra="", long err=[L4_ENOMEM](#))
Check for valid capability or raise C++ exception.
- [l4_msgtag_t chkipc](#) ([l4_msgtag_t](#) tag, char const *extra="", [l4_utcb_t](#) *utcb=[l4_utcb](#)())
Test a message tag for IPC errors.
- template<typename T>
[Shared_cap](#)< T > [make_shared_cap](#) ([L4Re::Cap_alloc](#) *ca)
Allocate a capability slot and wrap it in a [Shared_cap](#).
- template<typename T>
[Shared_del_cap](#)< T > [make_shared_del_cap](#) ([L4Re::Cap_alloc](#) *ca)
Allocate a capability slot and wrap it in a [Shared_del_cap](#).
- template<typename T>
[Unique_cap](#)< T > [make_unique_cap](#) ([L4Re::Cap_alloc](#) *ca)
Allocate a capability slot and wrap it in an [Unique_cap](#).
- template<typename T>
[Unique_del_cap](#)< T > [make_unique_del_cap](#) ([L4Re::Cap_alloc](#) *ca)
Allocate a capability slot and wrap it in an [Unique_del_cap](#).

15.9.1 Detailed Description

[L4Re](#) C++ Interfaces.

[L4](#) Runtime Environment.

15.9.2 Typedef Documentation

15.9.2.1 Shared_cap

```
template<typename T>
using L4Re::Shared\_cap = L4::Detail::Shared_cap_impl<T, Smart\_count\_cap<L4\_FP\_ALL\_SPACES>>
```

Shared capability that implements automatic free and unmap of the capability selector.

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

This shared capability implements a counted reference to a capability selector. The capability shall be unmapped and freed when the reference count in the allocator goes to zero.

Note

This type is intended for users who implement a custom capability allocator; otherwise use [L4Re::Util::Shared_cap](#).

Definition at line 33 of file [shared_cap](#).

15.9.2.2 shared_cap

```
template<typename T>
using L4Re::shared_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_ALL_SPACES>>
```

Shared capability that implements automatic free and unmap of the capability selector.

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

This shared capability implements a counted reference to a capability selector. The capability shall be unmapped and freed when the reference count in the allocator goes to zero.

Note

This type is intended for users who implement a custom capability allocator; otherwise use [L4Re::Util::Shared_cap](#).

Definition at line 36 of file [shared_cap](#).

15.9.2.3 Shared_del_cap

```
template<typename T>
using L4Re::Shared_del_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_DELETE_OBJ>>
```

Shared capability that implements automatic free and unmap+delete of the capability selector.

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

This shared capability implements a counted reference to a capability selector. The capability shall be unmapped and freed when the reference count in the allocator goes to zero. The main difference to [Shared_cap](#) is that the unmap is done with the deletion flag enabled and this leads to the deletion of the object if the current task holds appropriate deletion rights.

Note

This type is intended for users who implement a custom capability allocator; otherwise use [L4Re::Util::Shared_del_cap](#).

Definition at line 69 of file [shared_cap](#).

15.9.2.4 shared_del_cap

```
template<typename T>  
using L4Re::shared_del_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_DELETE_OBJ>>
```

Shared capability that implements automatic free and unmap+delete of the capability selector.

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

This shared capability implements a counted reference to a capability selector. The capability shall be unmapped and freed when the reference count in the allocator goes to zero. The main difference to [Shared_cap](#) is that the unmap is done with the deletion flag enabled and this leads to the deletion of the object if the current task holds appropriate deletion rights.

Note

This type is intended for users who implement a custom capability allocator; otherwise use [L4Re::Util::Shared_del_cap](#).

Definition at line 72 of file [shared_cap](#).

15.9.2.5 Unique_cap

```
template<typename T>
using L4Re::Unique_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_ALL_SPACES>>
```

Unique capability that implements automatic free and unmap of the capability selector.

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

The ownership of the capability is managed in the same way as `unique_ptr`.

Note

This type is intended for users who implement a custom capability allocator; otherwise use [L4Re::Util::Unique_cap](#).

Definition at line 31 of file [unique_cap](#).

15.9.2.6 unique_cap

```
template<typename T>
using L4Re::unique_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_ALL_SPACES>>
```

Unique capability that implements automatic free and unmap of the capability selector.

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

The ownership of the capability is managed in the same way as `unique_ptr`.

Note

This type is intended for users who implement a custom capability allocator; otherwise use [L4Re::Util::Unique_cap](#).

Definition at line 34 of file [unique_cap](#).

15.9.2.7 Unique_del_cap

```
template<typename T>  
using L4Re::Unique_del_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_DELETE_OBJ>>
```

Unique capability that implements automatic free and unmap+delete of the capability selector.

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

The main difference to [Unique_cap](#) is that the unmap is done with the deletion flag enabled and this leads to the deletion of the object if the current task holds appropriate deletion rights.

Note

This type is intended for users who implement a custom capability allocator; otherwise use [L4Re::Util::Unique_del_cap](#).

Definition at line 64 of file [unique_cap](#).

15.9.2.8 unique_del_cap

```
template<typename T>
using L4Re::unique_del_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_DELETE_OBJ>>
```

Unique capability that implements automatic free and unmap+delete of the capability selector.

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

The main difference to [Unique_cap](#) is that the unmap is done with the deletion flag enabled and this leads to the deletion of the object if the current task holds appropriate deletion rights.

Note

This type is intended for users who implement a custom capability allocator; otherwise use [L4Re::Util::Unique_del_cap](#).

Definition at line 67 of file [unique_cap](#).

15.9.3 Function Documentation

15.9.3.1 chkcap()

```
template<typename T>
T L4Re::chkcap (
    T && cap,
    char const * extra = "",
    long err = -L4_ENOMEM) [inline]
```

Check for valid capability or raise C++ exception.

Template Parameters

<i>T</i>	Type of object to check, must be capability-like (L4::Cap , L4Re::Util::Unique_cap etc.)
----------	---

Parameters

<i>cap</i>	Capability value to check.
<i>extra</i>	Optional text for exception.
<i>err</i>	Error value for exception or 0 if the error code stored in the invalid capability should be used.

This function checks whether the capability is valid. If the capability is invalid, a C++ exception is generated, using *err* if *err* is not zero, otherwise the capability value is used. A valid capability will just be returned.

Definition at line 149 of file [error_helper](#).

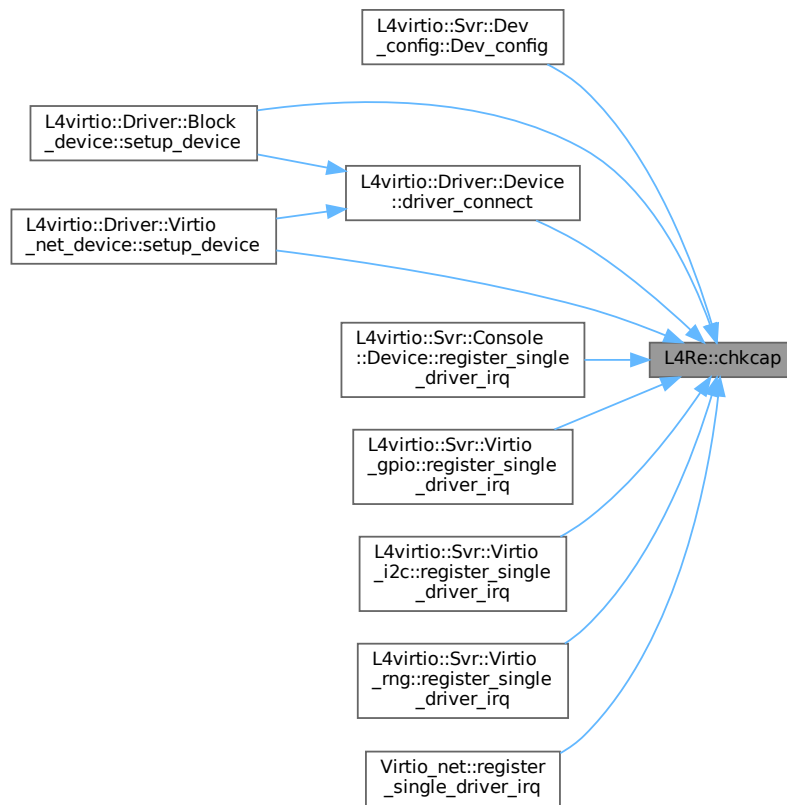
References [L4_ENOMEM](#), [L4_UNLIKELY](#), and [throw_error\(\)](#).

Referenced by [L4virtio::Svr::Dev_config::Dev_config\(\)](#), [L4virtio::Driver::Device::driver_connect\(\)](#), [L4virtio::Svr::Console::Device::register_single_driver_irq\(\)](#), [L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::register_single_driver_irq\(\)](#), [L4virtio::Svr::Virtio_i2c< Request_handler, Epiface >::register_single_driver_irq\(\)](#), [L4virtio::Svr::Virtio_rng< Rnd_state, Epiface >::register_single_driver_irq\(\)](#), [Virtio_net::register_single_driver_irq\(\)](#), [L4virtio::Driver::Block_device::setup_device\(\)](#), and [L4virtio::Driver::Virtio_net_device::setup_device\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.9.3.2 chkipc()

```

l4_msgtag_t L4Re::chkipc (
    l4_msgtag_t tag,
    char const * extra = "",
    l4_utcb_t * utcb = l4_utcb()) [inline]

```

Test a message tag for IPC errors.

Parameters

<i>tag</i>	Message tag returned by the IPC.
<i>extra</i>	Exception message in case of error.
<i>utcb</i>	The UTCB used in the IPC operation.

Returns

On IPC error an exception is thrown, otherwise `tag` is returned.

Exceptions

<code>L4::Runtime_exception</code>	with the translated IPC error code
------------------------------------	------------------------------------

This function does not check the message tag's label value.

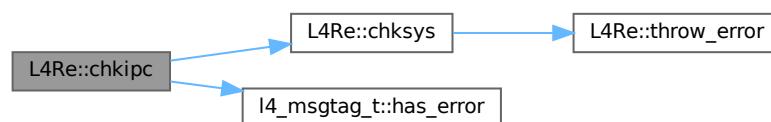
Note

This must be called on a message tag before the UTCB is changed.

Definition at line 180 of file [error_helper](#).

References [chksys\(\)](#), [l4_msgtag_t::has_error\(\)](#), and [L4_UNLIKELY](#).

Here is the call graph for this function:



15.9.3.3 chksys() [1/3]

```

long L4Re::chksys (
    l4_msgtag_t const & t,
    char const * extra = "",
    l4_utcb_t * utcb = l4_utcb(),
    long ret = 0) [inline]

```

Generate C++ exception on error.

Parameters

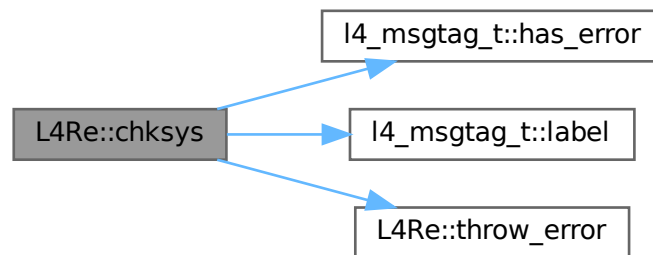
<i>t</i>	Message tag.
<i>extra</i>	Optional text for exception (default "")
<i>utcb</i>	Option UTCB
<i>ret</i>	Optional value for exception, default is error value (err)

This function throws an exception if the message tag contains an error or the label in the message tag is negative. Otherwise the label in the message tag is returned.

Definition at line 93 of file [error_helper](#).

References [l4_msgtag_t::has_error\(\)](#), [L4_UNLIKELY](#), [l4_msgtag_t::label\(\)](#), and [throw_error\(\)](#).

Here is the call graph for this function:



15.9.3.4 chksys() [2/3]

```

long L4Re::chksys (
    l4_msgtag_t const & t,
    l4_utcb_t * utcb,
    char const * extra = "") [inline]
  
```

Generate C++ exception on error.

Parameters

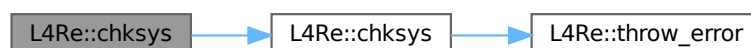
<i>t</i>	Message tag.
<i>utcb</i>	UTCB.
<i>extra</i>	Optional text for exception (default "")

This function throws an exception if the message tag contains an error or the label in the message tag is negative. Otherwise the label in the message tag is returned.

Definition at line 116 of file [error_helper](#).

References [chksys\(\)](#).

Here is the call graph for this function:



15.9.3.5 chksys() [3/3]

```
long L4Re::chksys (
    long err,
    char const * extra = "",
    long ret = 0) [inline]
```

Generate C++ exception on error.

Parameters

<i>err</i>	Error value, if negative exception will be thrown
<i>extra</i>	Optional text for exception (default "")
<i>ret</i>	Optional value for exception, default is error value (err)

This function throws an exception if the *err* is negative and otherwise returns *err*.

Examples

[examples/libs/l4re/c++/shared_ds/ds_srv.cc](#).

Definition at line 72 of file [error_helper](#).

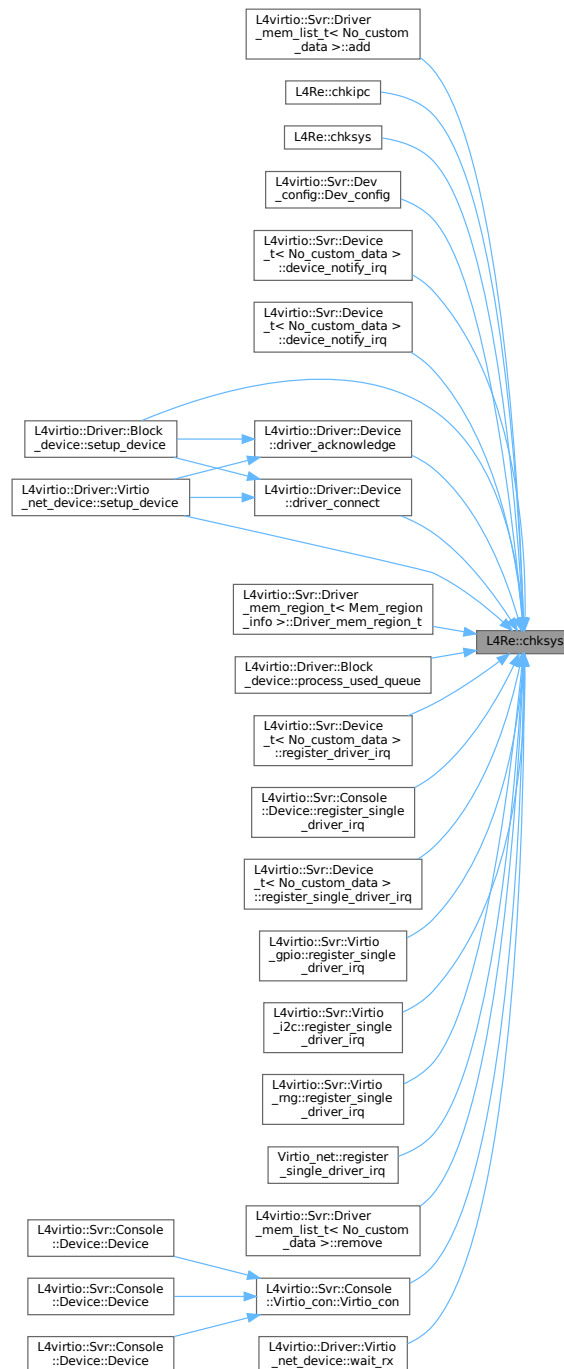
References [L4_UNLIKELY](#), and [throw_error\(\)](#).

Referenced by [L4virtio::Svr::Driver_mem_list_t< No_custom_data >::add\(\)](#), [chkipc\(\)](#), [chksys\(\)](#), [L4virtio::Svr::Dev_config::Dev_config\(\)](#), [L4virtio::Svr::Device_t< No_custom_data >::device_notify_irq\(\)](#), [L4virtio::Svr::Device_t< No_custom_data >::device_notify_irq\(\)](#), [L4virtio::Driver::Device::driver_acknowledge\(\)](#), [L4virtio::Driver::Device::driver_connect\(\)](#), [L4virtio::Svr::Driver_mem_region_t< Mem_region >::register_driver_irq\(\)](#), [L4virtio::Driver::Block_device::process_used_queue\(\)](#), [L4virtio::Svr::Device_t< No_custom_data >::register_driver_irq\(\)](#), [L4virtio::Svr::Console::Device::register_single_driver_irq\(\)](#), [L4virtio::Svr::Device_t< No_custom_data >::register_single_driver_irq\(\)](#), [L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::register_single_driver_irq\(\)](#), [L4virtio::Svr::Virtio_i2c< Request_handler, Epiface >::register_single_driver_irq\(\)](#), [L4virtio::Svr::Virtio_rng< Rnd_state, Epiface >::register_single_driver_irq\(\)](#), [Virtio_net::register_single_driver_irq\(\)](#), [L4virtio::Svr::Driver_mem_list_t< No_custom_data >::remove\(\)](#), [L4virtio::Driver::Block_device::setup_device\(\)](#), [L4virtio::Driver::Virtio_net_device::setup_device\(\)](#), [L4virtio::Svr::Console::Virtio_con::Virtio_con\(\)](#), and [L4virtio::Driver::Virtio_net_device::Virtio_net_device\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.9.3.6 make_shared_cap()

```

template<typename T>
Shared_cap< T > L4Re::make_shared_cap (
    L4Re::Cap_alloc * ca)
  
```

Allocate a capability slot and wrap it in a [Shared_cap](#).

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

Parameters

<i>ca</i>	Capability allocator to use.
-----------	------------------------------

Note

This function is intended for users who implement a custom capability allocator; otherwise use [L4Re::Util::make_shared_cap<T>\(\)](#).

Definition at line 49 of file [shared_cap](#).

References [L4Re::Cap_alloc::alloc\(\)](#).

Here is the call graph for this function:



15.9.3.7 make_shared_del_cap()

```

template<typename T>
Shared_del_cap< T > L4Re::make_shared_del_cap (
    L4Re::Cap_alloc * ca)
  
```

Allocate a capability slot and wrap it in a [Shared_del_cap](#).

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

Parameters

<i>ca</i>	Capability allocator to use.
-----------	------------------------------

Note

This function is intended for users who implement a custom capability allocator; otherwise use [L4Re::Util::make_shared_del_cap<T>\(\)](#).

Definition at line 85 of file [shared_cap](#).

References [L4Re::Cap_alloc::alloc\(\)](#).

Here is the call graph for this function:

**15.9.3.8 make_unique_cap()**

```

template<typename T>
Unique_cap< T > L4Re::make_unique_cap (
    L4Re::Cap_alloc * ca)
  
```

Allocate a capability slot and wrap it in an [Unique_cap](#).

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

Parameters

<i>ca</i>	Capability allocator to use.
-----------	------------------------------

Note

This function is intended for users who implement a custom capability allocator; otherwise use [L4Re::Util::make_unique_cap<T>\(\)](#).

Definition at line 47 of file [unique_cap](#).

References [L4Re::Cap_alloc::alloc\(\)](#).

Here is the call graph for this function:



15.9.3.9 make_unique_del_cap()

```
template<typename T>
Unique_del_cap< T > L4Re::make_unique_del_cap (
    L4Re::Cap_alloc * ca)
```

Allocate a capability slot and wrap it in an [Unique_del_cap](#).

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

Parameters

<i>ca</i>	Capability allocator to use.
-----------	------------------------------

Note

This function is intended for users who implement a custom capability allocator; otherwise use [L4Re::Util::make_unique_del_cap<T>\(\)](#).

Definition at line 80 of file [unique_cap](#).

References [L4Re::Cap_alloc::alloc\(\)](#).

Here is the call graph for this function:



15.9.3.10 throw_error()

```
void L4Re::throw_error (
    long err,
    char const * extra = "") [inline]
```

Generate C++ exception.

Parameters

<i>err</i>	Error value
------------	-------------

<i>extra</i>	Optional text for exception (default "")
--------------	--

This function throws an [L4](#) exception. The exact exception type depends on the error value (err). This function does never return.

Examples

[examples/libs/l4re/c++/shared_ds/ds_srv.cc](#).

Definition at line [37](#) of file [error_helper](#).

References [L4_EEXIST](#), [L4_ENOENT](#), [L4_ENOMEM](#), and [L4_ERANGE](#).

Referenced by [chkcapp\(\)](#), [chksys\(\)](#), and [chksys\(\)](#).

Data Structures

- class [Cap_alloc_base](#)
Capability allocator.
- class [Br_manager](#)
Buffer-register (BR) manager for [L4::Server](#).
- struct [Br_manager_hooks](#)
Predefined server-loop hooks for a server loop using the [Br_manager](#).
- struct [Br_manager_timeout_hooks](#)
Predefined server-loop hooks for a server with using the [Br_manager](#) and a timeout queue.
- class [Smart_cap_auto](#)
Helper for [Unique_cap](#) and [Unique_del_cap](#).
- class [Smart_count_cap](#)
Helper for [Ref_cap](#) and [Ref_del_cap](#).
- struct [Ref_cap](#)
Automatic capability that implements automatic free and unmap of the capability selector.
- struct [Ref_del_cap](#)
Automatic capability that implements automatic free and unmap+delete of the capability selector.
- class [_Cap_alloc](#)
Adapter to expose the cap allocator implementation as [L4Re::Cap_alloc](#) compatible class.
- struct [Counter](#)
Counter for [Counting_cap_alloc](#) with variable data width.
- struct [Counter_atomic](#)
Thread safe version of counter for [Counting_cap_alloc](#).
- class [Counting_cap_alloc](#)
Internal reference-counting cap allocator.
- class [Dataspace_svr](#)
Dataspace server class.
- class [Event_t](#)
Convenience wrapper for getting access to an event object.
- class [Event_buffer_t](#)
Event_buffer utility class.
- class [Event_buffer_consumer_t](#)
An event buffer consumer.
- class [Event_svr](#)
Convenience wrapper for implementing an event server.
- class [Item_alloc_base](#)
Item allocator.
- class [Object_registry](#)
A registry that manages server objects and their attached IPC gates for a single server loop for a specific thread.
- class [Registry_server](#)
A server loop object which has a [Object_registry](#) included.
- class [Vcon_svr](#)
Console server template class.
- class [Bitmap_base](#)
Basic bitmap abstraction.
- class [Bitmap](#)
A static bitmap.

Typedefs

- `template<typename T>`
`using Shared_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_ALL_SPACES>>`
Shared capability that implements automatic free and unmap of the capability selector.
- `template<typename T>`
`using shared_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_ALL_SPACES>>`
Shared capability that implements automatic free and unmap of the capability selector.
- `template<typename T>`
`using Shared_del_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_DELETE_OBJ>>`
Shared capability that implements automatic free and unmap+delete of the capability selector.
- `template<typename T>`
`using shared_del_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_DELETE_OBJ>>`
Shared capability that implements automatic free and unmap+delete of the capability selector.
- `template<typename T>`
`using Unique_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_ALL_SPACES>>`
Unique capability that implements automatic free and unmap of the capability selector.
- `template<typename T>`
`using unique_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_ALL_SPACES>>`
Unique capability that implements automatic free and unmap of the capability selector.
- `template<typename T>`
`using Unique_del_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_DELETE_OBJ>>`
Unique capability that implements automatic free and unmap+delete of the capability selector.
- `template<typename T>`
`using unique_del_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_DELETE_OBJ>>`
Unique capability that implements automatic free and unmap+delete of the capability selector.

Functions

- `template<typename T>`
`Ref_cap< T >::Cap make_ref_cap ()`
Allocate a capability slot and wrap it in a [Ref_cap](#).
- `template<typename T>`
`Ref_del_cap< T >::Cap make_ref_del_cap ()`
Allocate a capability slot and wrap it in a [Ref_del_cap](#).
- `int kumem_alloc (l4_addr_t *mem, unsigned pages_order, L4::Cap< L4::Task > task=L4Re::Env::env() ->task(), L4::Cap< L4Re::Rm > rm=L4Re::Env::env() ->rm()) noexcept`
Allocate state area.
- `template<typename T>`
`Shared_cap< T > make_shared_cap ()`
Allocate a capability slot and wrap it in a [Shared_cap](#).
- `template<typename T>`
`Shared_del_cap< T > make_shared_del_cap ()`
Allocate a capability slot and wrap it in a [Shared_del_cap](#).
- `template<typename T>`
`Unique_cap< T > make_unique_cap ()`
Allocate a capability slot and wrap it in an [Unique_cap](#).
- `template<typename T>`
`Unique_del_cap< T > make_unique_del_cap ()`
Allocate a capability slot and wrap it in an [Unique_del_cap](#).

Variables

- [_Cap_alloc](#) & [cap_alloc](#)
Capability allocator.

15.10.1 Detailed Description

Documentation of the [L4](#) Runtime Environment utility functionality in C++.

15.10.2 Typedef Documentation

15.10.2.1 Shared_cap

```
template<typename T>
using L4Re::Util::Shared\_cap = L4::Detail::Shared_cap_impl<T, Smart\_count\_cap<L4\_FP\_ALL\_SPACES>>
```

Shared capability that implements automatic free and unmap of the capability selector.

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

This shared capability implements a counted reference to a capability selector. The capability shall be unmapped and freed when the reference count in the allocator goes to zero.

Usage:

```
L4Re::Util::Shared_cap<L4Re::Dataspace> global_ds_cap;

{
    L4Re::Util::Shared_cap<L4Re::Dataspace>
        ds_cap = make_shared_cap<L4Re::Dataspace>();
    // reference count for the allocated cap selector is now 1

    // use the dataspace cap
    L4Re::chksys(mem_alloc->alloc(4096, ds_cap.get()));

    global_ds_cap = ds_cap;
    // reference count is now 2
    ...
}
// reference count dropped to 1 (ds_cap is no longer existing).
```

Definition at line [48](#) of file [shared_cap](#).

15.10.2.2 shared_cap

```
template<typename T>
using L4Re::Util::shared\_cap = L4::Detail::Shared_cap_impl<T, Smart\_count\_cap<L4\_FP\_ALL\_SPACES>>
```

Shared capability that implements automatic free and unmap of the capability selector.

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

This shared capability implements a counted reference to a capability selector. The capability shall be unmapped and freed when the reference count in the allocator goes to zero.

Usage:

```
L4Re::Util::Shared_cap<L4Re::Dataspace> global_ds_cap;

{
    L4Re::Util::Shared_cap<L4Re::Dataspace>
        ds_cap = make_shared_cap<L4Re::Dataspace>();
    // reference count for the allocated cap selector is now 1

    // use the dataspace cap
    L4Re::chksys(mem_alloc->alloc(4096, ds_cap.get()));

    global_ds_cap = ds_cap;
    // reference count is now 2
    ...
}
// reference count dropped to 1 (ds_cap is no longer existing).
```

Definition at line 51 of file [shared_cap](#).

15.10.2.3 Shared_del_cap

```
template<typename T>
using L4Re::Util::Shared_del_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_DELETE_OBJ>>
```

Shared capability that implements automatic free and unmap+delete of the capability selector.

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

This shared capability implements a counted reference to a capability selector. The capability shall be unmapped and freed when the reference count in the allocator goes to zero. The main difference to [Shared_cap](#) is that the unmap is done with the deletion flag enabled and this leads to the deletion of the object if the current task holds appropriate deletion rights.

Usage:

```
L4Re::Util::Shared_del_cap<L4Re::Dataspace> global_ds_cap;

{
    L4Re::Util::Shared_del_cap<L4Re::Dataspace>
        ds_cap = make_shared_del_cap<L4Re::Dataspace>();
    // reference count for the allocated cap selector is now 1

    // use the dataspace cap
    L4Re::chksys(mem_alloc->alloc(4096, ds_cap.get()));

    global_ds_cap = ds_cap;
    // reference count is now 2
    ...
}
// reference count dropped to 1 (ds_cap is no longer existing).
...
global_ds_cap = L4_INVALID_CAP;
// reference count dropped to 0 (data space shall be deleted).
```

Definition at line 98 of file [shared_cap](#).

15.10.2.4 shared_del_cap

```
template<typename T>
using L4Re::Util::shared_del_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_DELETE_OBJ>>
```

Shared capability that implements automatic free and unmap+delete of the capability selector.

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

This shared capability implements a counted reference to a capability selector. The capability shall be unmapped and freed when the reference count in the allocator goes to zero. The main difference to [Shared_cap](#) is that the unmap is done with the deletion flag enabled and this leads to the deletion of the object if the current task holds appropriate deletion rights.

Usage:

```
L4Re::Util::Shared_del_cap<L4Re::Dataspace> global_ds_cap;

{
    L4Re::Util::Shared_del_cap<L4Re::Dataspace>
    ds_cap = make_shared_del_cap<L4Re::Dataspace>();
    // reference count for the allocated cap selector is now 1

    // use the dataspace cap
    L4Re::chksys(mem_alloc->alloc(4096, ds_cap.get()));

    global_ds_cap = ds_cap;
    // reference count is now 2
    ...
}
// reference count dropped to 1 (ds_cap is no longer existing).
...
global_ds_cap = L4_INVALID_CAP;
// reference count dropped to 0 (data space shall be deleted).
```

Definition at line 101 of file [shared_cap](#).

15.10.2.5 Unique_cap

```
template<typename T>
using L4Re::Util::Unique_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_ALL_SPACES>>
```

Unique capability that implements automatic free and unmap of the capability selector.

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

The ownership of the capability is managed in the same way as `unique_ptr`.

Usage:

```

{
    L4Re::Util::Unique_cap<L4Re::Dataspace>
        ds_cap = L4Re::Util::make_unique_cap<L4Re::Dataspace>();

    // use the dataspace cap
    L4Re::chksys(mem_alloc->alloc(L4_PAGESIZE, ds_cap.get()));

    ...

    // At the end of the scope ds_cap is unmapped and the capability
    // selector is freed.
}

```

Definition at line 43 of file [unique_cap](#).

15.10.2.6 unique_cap

```

template<typename T>
using L4Re::Util::unique_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_ALL_SPACES>>

```

Unique capability that implements automatic free and unmap of the capability selector.

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

The ownership of the capability is managed in the same way as `unique_ptr`.

Usage:

```

{
    L4Re::Util::Unique_cap<L4Re::Dataspace>
        ds_cap = L4Re::Util::make_unique_cap<L4Re::Dataspace>();

    // use the dataspace cap
    L4Re::chksys(mem_alloc->alloc(L4_PAGESIZE, ds_cap.get()));

    ...

    // At the end of the scope ds_cap is unmapped and the capability
    // selector is freed.
}

```

Definition at line 46 of file [unique_cap](#).

15.10.2.7 Unique_del_cap

```

template<typename T>
using L4Re::Util::Unique_del_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_DELETE_OBJ>>

```

Unique capability that implements automatic free and unmap+delete of the capability selector.

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

The main difference to [Unique_cap](#) is that the unmap is done with the deletion flag enabled and this leads to the deletion of the object if the current task holds appropriate deletion rights.

Usage:

```
{
    L4Re::Util::Unique_del_cap<L4Re::Dataspace>
        ds_cap = make_unique_del_cap<L4Re::Dataspace>();

    // use the dataspace cap
    L4Re::chksys(mem_alloc->alloc(L4_PAGESIZE, ds_cap.get()));

    ...

    // At the end of the scope ds_cap is unmapped and the capability
    // selector is freed. Because the deletion flag is set the data space
    // shall also be deleted (even if there are other references to this
    // data space).
}
```

Definition at line 86 of file [unique_cap](#).

15.10.2.8 unique_del_cap

```
template<typename T>
using L4Re::Util::unique_del_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_DELETE_OBJ>>
```

Unique capability that implements automatic free and unmap+delete of the capability selector.

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

The main difference to [Unique_cap](#) is that the unmap is done with the deletion flag enabled and this leads to the deletion of the object if the current task holds appropriate deletion rights.

Usage:

```
{
    L4Re::Util::Unique_del_cap<L4Re::Dataspace>
        ds_cap = make_unique_del_cap<L4Re::Dataspace>();

    // use the dataspace cap
    L4Re::chksys(mem_alloc->alloc(L4_PAGESIZE, ds_cap.get()));

    ...

    // At the end of the scope ds_cap is unmapped and the capability
    // selector is freed. Because the deletion flag is set the data space
    // shall also be deleted (even if there are other references to this
    // data space).
}
```

Definition at line 89 of file [unique_cap](#).

15.10.3 Function Documentation

15.10.3.1 `make_shared_cap()`

```
template<typename T>
Shared_cap< T > L4Re::Util::make_shared_cap ()
```

Allocate a capability slot and wrap it in a [Shared_cap](#).

Template Parameters

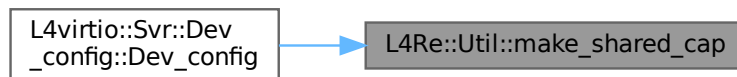
<i>T</i>	Type of the object the capability refers to.
----------	--

Definition at line 60 of file [shared_cap](#).

References [cap_alloc](#).

Referenced by [L4virtio::Svr::Dev_config::Dev_config\(\)](#).

Here is the caller graph for this function:



15.10.3.2 `make_shared_del_cap()`

```
template<typename T>
Shared_del_cap< T > L4Re::Util::make_shared_del_cap ()
```

Allocate a capability slot and wrap it in a [Shared_del_cap](#).

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

Definition at line 110 of file [shared_cap](#).

References [cap_alloc](#).

15.10.3.3 `make_unique_cap()`

```
template<typename T>
Unique_cap< T > L4Re::Util::make_unique_cap ()
```

Allocate a capability slot and wrap it in an [Unique_cap](#).

Template Parameters

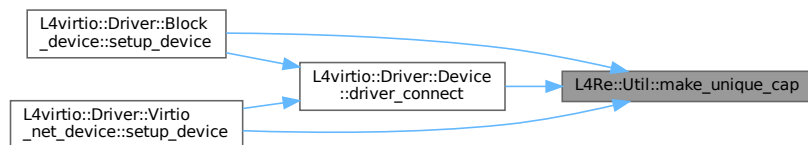
<i>T</i>	Type of the object the capability refers to.
----------	--

Definition at line 55 of file [unique_cap](#).

References [cap_alloc](#).

Referenced by [L4virtio::Driver::Device::driver_connect\(\)](#), [L4virtio::Driver::Block_device::setup_device\(\)](#), and [L4virtio::Driver::Virtio_net_device::setup_device\(\)](#).

Here is the caller graph for this function:



15.10.3.4 make_unique_del_cap()

```
template<typename T>
Unique_del_cap< T > L4Re::Util::make_unique_del_cap ()
```

Allocate a capability slot and wrap it in an [Unique_del_cap](#).

Template Parameters

<i>T</i>	Type of the object the capability refers to.
----------	--

Definition at line 98 of file [unique_cap](#).

References [cap_alloc](#).

15.11 L4Re::Vfs Namespace Reference

Virtual file system for interfaces in POSIX libc.

Data Structures

- class [Be_file](#)
Boiler plate class for implementing an open file for [L4Re::Vfs](#).
- class [Be_file_system](#)
Boilerplate class for implementing a [L4Re::Vfs::File_system](#).
- class [Generic_file](#)
The common interface for an open POSIX file.
- class [Directory](#)
Interface for a POSIX file that is a directory.
- class [Regular_file](#)
Interface for a POSIX file that provides regular file semantics.
- class [Special_file](#)
Interface for a POSIX file that provides special file semantics.
- class [File](#)
The basic interface for an open POSIX file.
- class [Mman](#)
Interface for POSIX memory management.
- class [File_system](#)
Basic interface for an [L4Re::Vfs](#) file system.
- class [Fs](#)
POSIX File-system related functionality.
- class [Ops](#)
Interface for the POSIX backends of an application.

Functions

- [L4Re::Vfs::Ops](#) *vfs_ops **asm** ("l4re_env_posix_vfs_ops")
Reference to the applications [L4Re::Vfs::Ops](#) singleton.

15.11.1 Detailed Description

Virtual file system for interfaces in POSIX libc.

15.12 L4vbus Namespace Reference

C++ interface of the [Vbus](#) API.

Data Structures

- class [Pm](#)
Power-management API mixin.
- class [Device](#)
Device on a [L4vbus::Vbus](#).
- class [Icu](#)
[Vbus](#) Interrupt controller API.
- class [Vbus](#)
The virtual bus ([Vbus](#)) interface.
- class [Gpio_pin](#)
A GPIO pin.
- class [Gpio_module](#)
A [Gpio_module](#) groups multiple GPIO pins together.
- class [Pci_host_bridge](#)
A Pci host bridge.
- class [Pci_dev](#)
A PCI device.

15.12.1 Detailed Description

C++ interface of the [Vbus](#) API.

The virtual bus ([Vbus](#)) is a hierarchical (tree) structure of device nodes where each device has a set of resources attached to it. Each virtual bus provides an [Icu](#) ([Interrupt controller](#)) for interrupt handling.

The [Vbus](#) interface allows a client to find and query devices present on his virtual bus. After obtaining a device handle for a specific device the client can enumerate its resources.

Refer to [L4 Vbus functions](#) for the C API.

Include File

```
#include <l4/vbus/vbus>
```

Include File

```
#include <l4/vbus/vbus_gpio>
```

Include File

```
#include <l4/vbus/vbus_pci>
```

15.13 L4virtio Namespace Reference

L4-VIRTIO Transport C++ API.

Data Structures

- class [Device](#)
IPC interface for virtio over [L4](#) IPC.
- class [Ptr](#)
Pointer used in virtio descriptors.
- class [Virtqueue](#)
Low-level [Virtqueue](#).

15.13.1 Detailed Description

L4-VIRTIO Transport C++ API.

Chapter 16

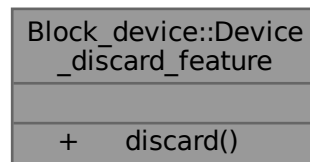
Data Structure Documentation

16.1 Block_device::Device_discard_feature Struct Reference

Partial interface for devices that offer discard functionality.

```
#include <device.h>
```

Collaboration diagram for Block_device::Device_discard_feature:



Public Member Functions

- virtual int **discard** ([l4_uint64_t](#) offset, [Block_device::Inout_block](#) const &blocks, [Block_device::Inout_callback](#) const &cb, bool discard)=0
Issues one or more WRITE_ZEROES or DISCARD commands.

16.1.1 Detailed Description

Partial interface for devices that offer discard functionality.

Definition at line [119](#) of file [device.h](#).

The documentation for this struct was generated from the following file:

- [l4/libblock-device/device.h](#)

16.2 Block_device::Device_mgr< DEV, FACTORY, SCHEDULER > Class Template Reference

Basic class that scans devices and handles client connections.

```
#include <block_device_mgr.h>
```

Collaboration diagram for Block_device::Device_mgr< DEV, FACTORY, SCHEDULER >:

Block_device::Device_mgr< DEV, FACTORY, SCHEDULER >	
+	check_clients()
+	shutdown_event()
+	parse_device_name()

Public Member Functions

- void **check_clients** ()
Remove clients where the client IPC gate is no longer valid.
- void **shutdown_event** (Shutdown_type type)
Process a shutdown event on all connections.

Static Public Member Functions

- static int **parse_device_name** (std::string const ¶m, std::string &device)
Parse and verify a device string parameter.

16.2.1 Detailed Description

```
template<typename DEV, typename FACTORY = Simple_factory<DEV>, typename SCHEDULER = Rr_↔
scheduler<typename FACTORY::Device_type>>
class Block_device::Device_mgr< DEV, FACTORY, SCHEDULER >
```

Basic class that scans devices and handles client connections.

Template Parameters

<i>DEV</i>	Base class for all devices.
------------	-----------------------------

<i>FACTORY</i>	Class that creates clients and partitions. See Simple_factory for an example of the required interface.
<i>SCHEDULER</i>	Class that schedules VIRTIO block requests from all clients.

Definition at line 79 of file [block_device_mgr.h](#).

16.2.2 Member Function Documentation

16.2.2.1 parse_device_name()

```
template<typename DEV, typename FACTORY = Simple_factory<DEV>, typename SCHEDULER = Rr_↵
scheduler<typename FACTORY::Device_type>>
int Block_device::Device_mgr< DEV, FACTORY, SCHEDULER >::parse_device_name (
    std::string const & param,
    std::string & device) [inline], [static]
```

Parse and verify a device string parameter.

Parameters

in	<i>param</i>	Device string name parameter.
out	<i>device</i>	Device name extracted from parameter.

Returns

[L4](#) error code.

This function tests if 'param' contains one of the following variants of a device name and extracts it into 'device':

- "partlabel:<label>": 'device' contains "<label>" without conversion.
- "partuuid:<uuid>": Check if "<uuid>" is a valid UUID and return with error if not. In case of success, 'device' contains "<uuid>" with all characters converted to upper case.
- "<string>": Check if "<string>" is a valid UUID. If so, 'device' contains "<string>" with all characters converted to upper case. Otherwise, 'device' contains the unmodified "<string>".

Definition at line 383 of file [block_device_mgr.h](#).

References [L4_EINVAL](#), and [L4_EOK](#).

The documentation for this class was generated from the following file:

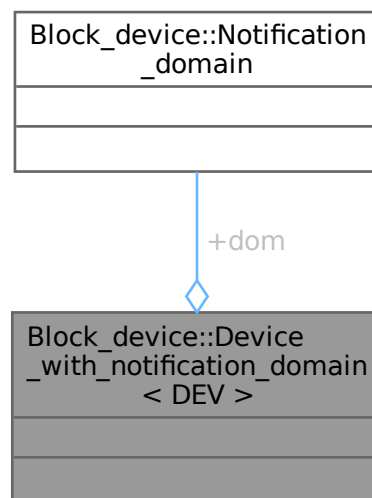
- l4/libblock-device/block_device_mgr.h

16.3 Block_device::Device_with_notification_domain< DEV > Struct Template Reference

Device with a per-device notification domain.

```
#include <device.h>
```

Collaboration diagram for Block_device::Device_with_notification_domain< DEV >:



16.3.1 Detailed Description

```
template<typename DEV>
struct Block_device::Device_with_notification_domain< DEV >
```

Device with a per-device notification domain.

Definition at line 109 of file [device.h](#).

The documentation for this struct was generated from the following file:

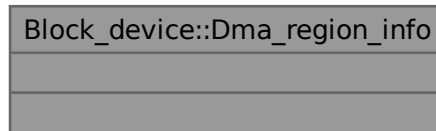
- `I4/libblock-device/device.h`

16.4 Block_device::Dma_region_info Struct Reference

Base class used by the driver implementation to derive its own DMA mapping tracking structure.

```
#include <types.h>
```

Collaboration diagram for Block_device::Dma_region_info:



16.4.1 Detailed Description

Base class used by the driver implementation to derive its own DMA mapping tracking structure.

Definition at line 43 of file [types.h](#).

The documentation for this struct was generated from the following file:

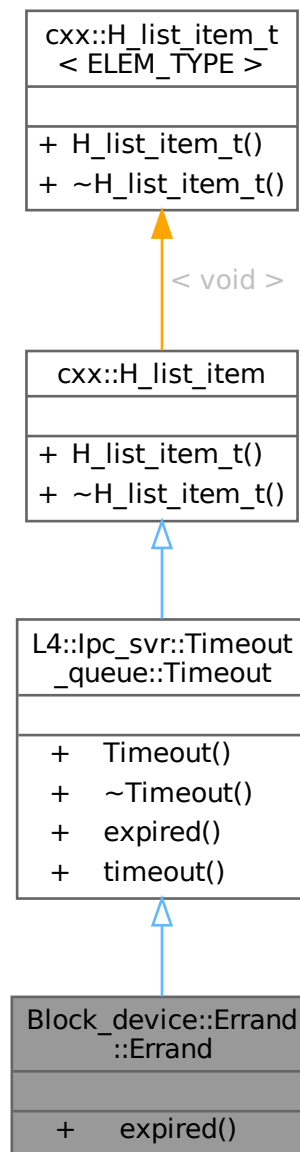
- I4/libblock-device/types.h

16.5 Block_device::Errand::Errand Class Reference

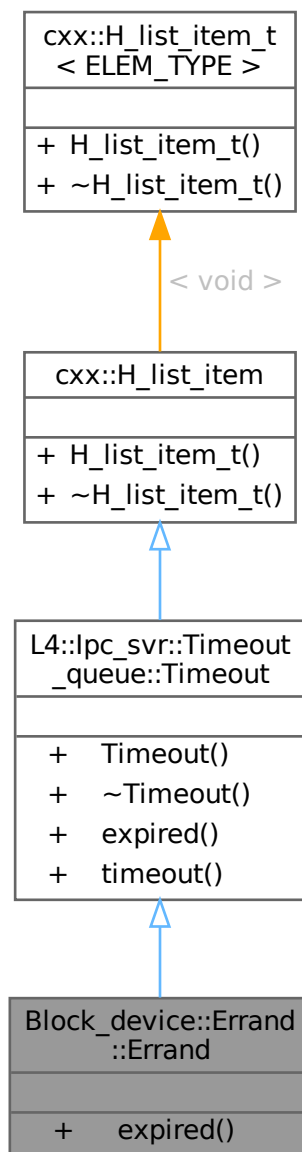
Wrapper for a small task executed asynchronously in the server loop.

```
#include <errand.h>
```

Inheritance diagram for Block_device::Errand::Errand:



Collaboration diagram for Block_device::Errand::Errand:



Public Member Functions

- void `expired()` final
callback function to be called when timeout happened

Public Member Functions inherited from `L4::lpc_svr::Timeout`

- `Timeout()`
Make a timeout.

- virtual `~Timeout ()=0`
Destroy a timeout.
- `l4_kernel_clock_t timeout () const`
return absolute timeout of this callback.

Public Member Functions inherited from `cxx::H_list_item_t< void >`

- `H_list_item_t ()`
Constructor.
- `~H_list_item_t () noexcept`
Destructor.

16.5.1 Detailed Description

Wrapper for a small task executed asynchronously in the server loop.

Errands are implemented as timeout tasks. They might be queued with the current timestamp, so that they are executed as soon as possible on the next iteration of the server loop or they might be scheduled with a timeout, which is particularly useful if the driver has to do a busy wait on the hardware.

Definition at line 98 of file [errand.h](#).

16.5.2 Member Function Documentation

16.5.2.1 `expired()`

```
void Block_device::Errand::Errand::expired () [inline], [final], [virtual]
```

callback function to be called when timeout happened

Note

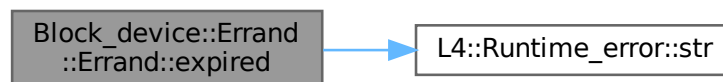
The timeout object is already dequeued when this function is called, this means the timeout may be safely queued again within the `expired()` function.

Implements [L4::lpc_svr::Timeout](#).

Definition at line 103 of file [errand.h](#).

References [L4::Runtime_error::str\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

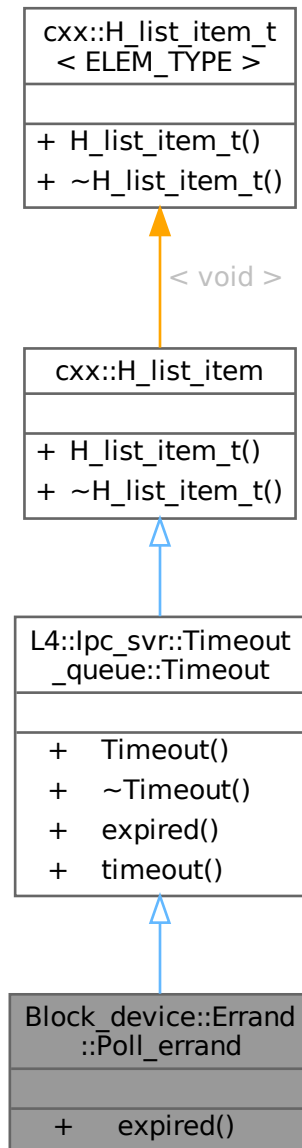
- [l4/libblock-device/errand.h](#)

16.6 Block_device::Errand::Poll_errand Class Reference

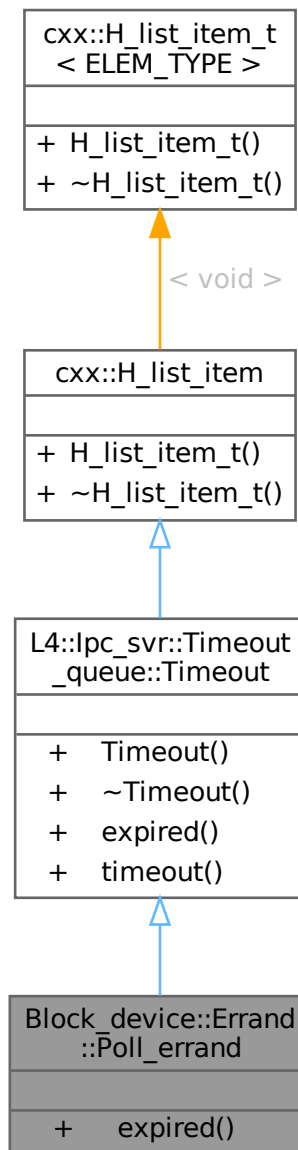
Wrapper for a regularly repeated task.

```
#include <errand.h>
```

Inheritance diagram for Block_device::Errand::Poll_errand:



Collaboration diagram for Block_device::Errand::Poll_errand:



Public Member Functions

- void `expired()` final
callback function to be called when timeout happened

Public Member Functions inherited from `L4::lpc_svr::Timeout`

- `Timeout()`
Make a timeout.

- virtual `~Timeout()` = 0
Destroy a timeout.
- `l4_kernel_clock_t timeout()` const
return absolute timeout of this callback.

Public Member Functions inherited from `cxx::H_list_item_t< void >`

- `H_list_item_t()`
Constructor.
- `~H_list_item_t()` noexcept
Destructor.

16.6.1 Detailed Description

Wrapper for a regularly repeated task.

Definition at line 32 of file [errand.h](#).

16.6.2 Member Function Documentation

16.6.2.1 `expired()`

```
void Block_device::Errand::Poll_errand::expired () [inline], [final], [virtual]
```

callback function to be called when timeout happened

Note

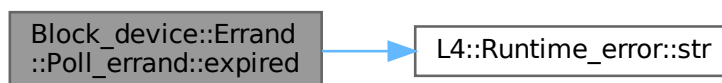
The timeout object is already dequeued when this function is called, this means the timeout may be safely queued again within the `expired()` function.

Implements [L4::lpc_svr::Timeout](#).

Definition at line 37 of file [errand.h](#).

References [L4::Runtime_error::str\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

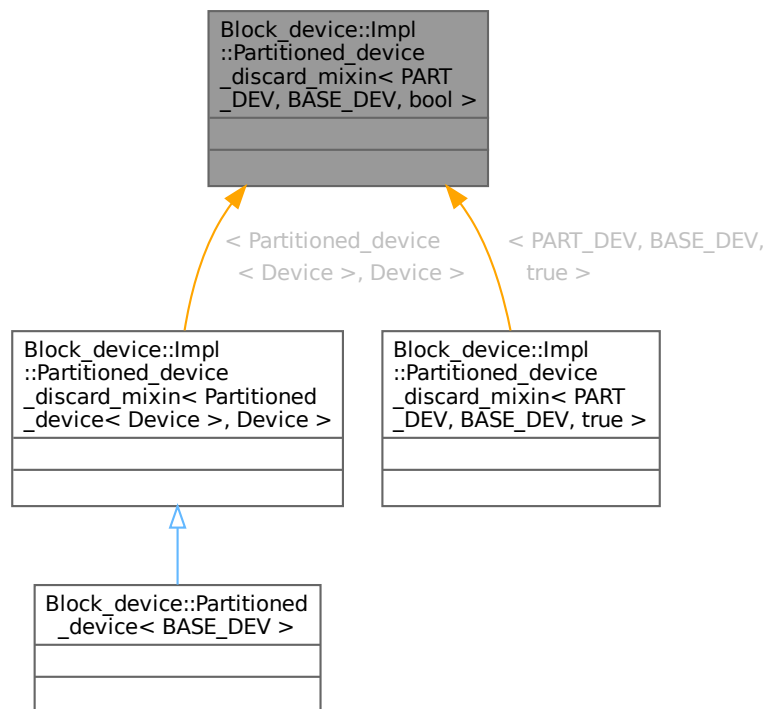
- `l4/libblock-device/errand.h`

16.7 Block_device::Impl::Partitioned_device_discard_mixin< PART_DEV, BASE_DEV, bool > Class Template Reference

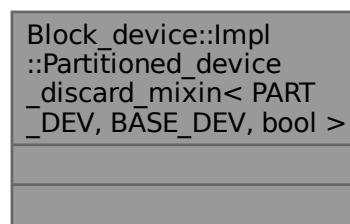
Dummy class used when the device class is not derived from [Device_discard_feature](#).

```
#include <part_device.h>
```

Inheritance diagram for Block_device::Impl::Partitioned_device_discard_mixin< PART_DEV, BASE_DEV, bool >:



Collaboration diagram for Block_device::Impl::Partitioned_device_discard_mixin< PART_DEV, BASE_DEV, bool >:



16.7.1 Detailed Description

```
template<typename PART_DEV, typename BASE_DEV, bool = std::is_base_of<Device_discard_feature,  
BASE_DEV>::value>  
class Block_device::Impl::Partitioned_device_discard_mixin< PART_DEV, BASE_DEV, bool >
```

Dummy class used when the device class is not derived from [Device_discard_feature](#).

Definition at line 28 of file [part_device.h](#).

The documentation for this class was generated from the following file:

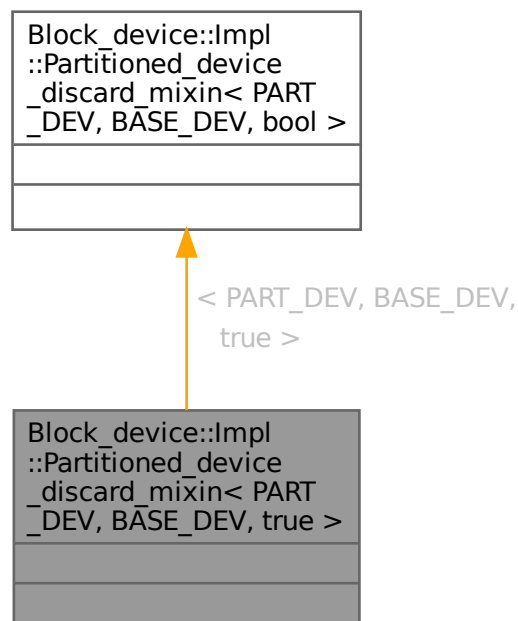
- I4/libblock-device/part_device.h

16.8 Block_device::Impl::Partitioned_device_discard_mixin< PART_DEV, BASE_DEV, true > Class Template Reference

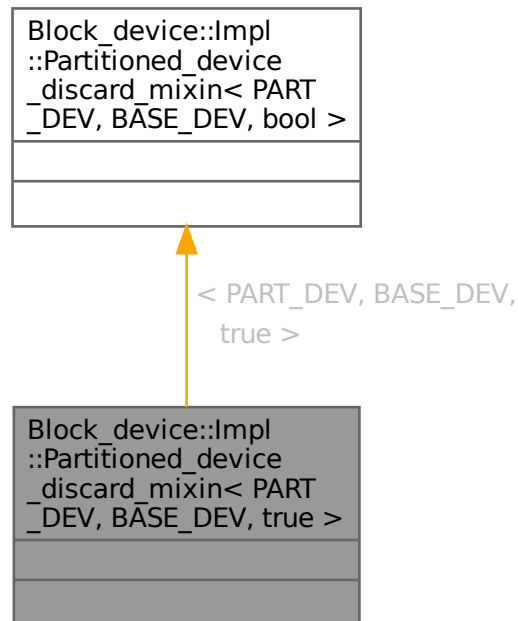
Mixin implementing discard for partition devices.

```
#include <part_device.h>
```

Inheritance diagram for Block_device::Impl::Partitioned_device_discard_mixin< PART_DEV, BASE_DEV, true >:



Collaboration diagram for `Block_device::Impl::Partitioned_device_discard_mixin< PART_DEV, BASE_DEV, true >`:



16.8.1 Detailed Description

```

template<typename PART_DEV, typename BASE_DEV>
class Block_device::Impl::Partitioned_device_discard_mixin< PART_DEV, BASE_DEV, true >

```

Mixin implementing discard for partition devices.

Template Parameters

<i>PART_DEV</i>	Class of the partition device
<i>BASE_DEV</i>	Class implementing the Device interface.

Definition at line 37 of file [part_device.h](#).

The documentation for this class was generated from the following file:

- `I4/libblock-device/part_device.h`

16.9 Block_device::Inout_block Struct Reference

Description of an inout block to be sent to the device.

```
#include <types.h>
```

Collaboration diagram for Block_device::Inout_block:

Block_device::Inout_block	
+	sector
+	flags

Data Fields

- [l4_uint64_t](#) **sector** = 0
Initial sector. Used only by DISCARD / WRITE_ZEROES requests.
- [l4_uint32_t](#) **flags** = 0
Flags from Inout_flags.

16.9.1 Detailed Description

Description of an inout block to be sent to the device.

Block may be scatter gather in which case they are chained via the next pointer.

Definition at line 66 of file [types.h](#).

The documentation for this struct was generated from the following file:

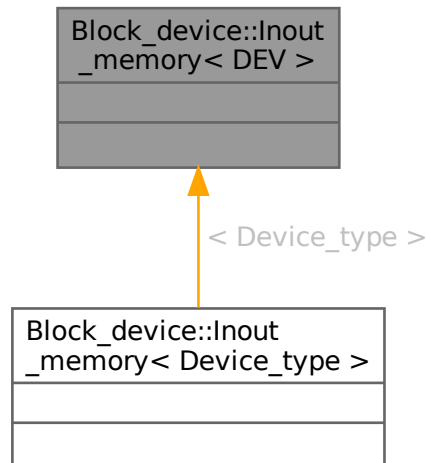
- [l4/libblock-device/types.h](#)

16.10 Block_device::Inout_memory< DEV > Class Template Reference

Helper class that temporarily allocates memory that can be used for in/out operations with the device.

```
#include <inout_memory.h>
```

Inheritance diagram for Block_device::Inout_memory< DEV >:



Collaboration diagram for Block_device::Inout_memory< DEV >:



16.10.1 Detailed Description

```
template<typename DEV>
class Block_device::Inout_memory< DEV >
```

Helper class that temporarily allocates memory that can be used for in/out operations with the device.

Definition at line 25 of file [inout_memory.h](#).

The documentation for this class was generated from the following file:

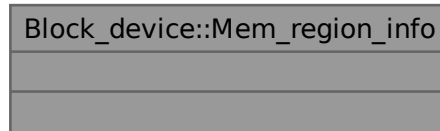
- `I4/libblock-device/inout_memory.h`

16.11 Block_device::Mem_region_info Struct Reference

Additional info stored in each [L4virtio::Svr::Driver_mem_region_t](#) used for tracking dataspace-wide DMA mappings.

```
#include <types.h>
```

Collaboration diagram for Block_device::Mem_region_info:



16.11.1 Detailed Description

Additional info stored in each [L4virtio::Svr::Driver_mem_region_t](#) used for tracking dataspace-wide DMA mappings.

Definition at line 52 of file [types.h](#).

The documentation for this struct was generated from the following file:

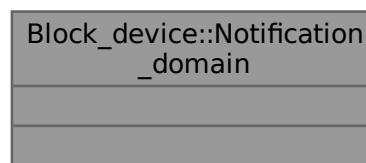
- `l4/libblock-device/types.h`

16.12 Block_device::Notification_domain Struct Reference

Opaque type for representing a notification domain.

```
#include <device.h>
```

Collaboration diagram for Block_device::Notification_domain:



16.12.1 Detailed Description

Opaque type for representing a notification domain.

Notification domains must be assigned to devices such that all devices that require a shared pool of resources to process their requests, also find themselves in the same notification domain. In particular, if two devices access common resources, then they must be in the same domain. An example of this are two partitions sharing the same parent device because processing of requests for one partition might depend on completion of request processing in another partition. On the other hand, independent disk devices will typically not share the same notification domain because their requests are completely independent of each other.

Definition at line 32 of file [device.h](#).

The documentation for this struct was generated from the following file:

- [l4/libblock-device/device.h](#)

16.13 Block_device::Partition_info Struct Reference

Information about a single partition.

```
#include <partition.h>
```

Collaboration diagram for Block_device::Partition_info:

Block_device::Partition_info	
+	guid
+	name
+	first
+	last
+	flags

Data Fields

- char **guid** [37]
ID of the partition.
- std::u16string **name**
UTF16 name of the partition.
- [l4_uint64_t](#) **first**
First valid sector.
- [l4_uint64_t](#) **last**
Last valid sector.
- [l4_uint64_t](#) **flags**
Additional flags, depending on partition type.

16.13.1 Detailed Description

Information about a single partition.

Definition at line 29 of file [partition.h](#).

The documentation for this struct was generated from the following file:

- l4/libblock-device/partition.h

16.14 Block_device::Partition_reader< DEV > Class Template Reference

Partition table reader for block devices.

```
#include <partition.h>
```

Collaboration diagram for Block_device::Partition_reader< DEV >:



16.14.1 Detailed Description

```
template<typename DEV>
class Block_device::Partition_reader< DEV >
```

Partition table reader for block devices.

Definition at line 43 of file [partition.h](#).

The documentation for this class was generated from the following file:

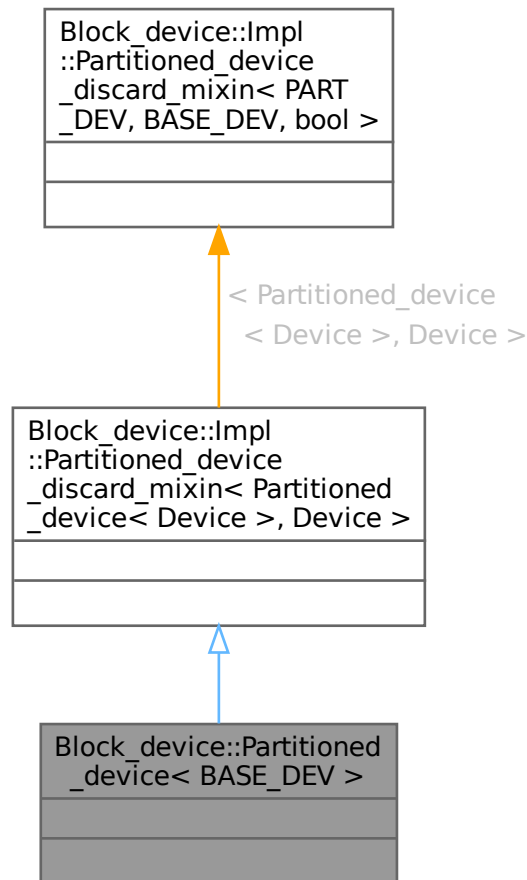
- l4/libblock-device/partition.h

16.15 Block_device::Partitioned_device< BASE_DEV > Class Template Reference

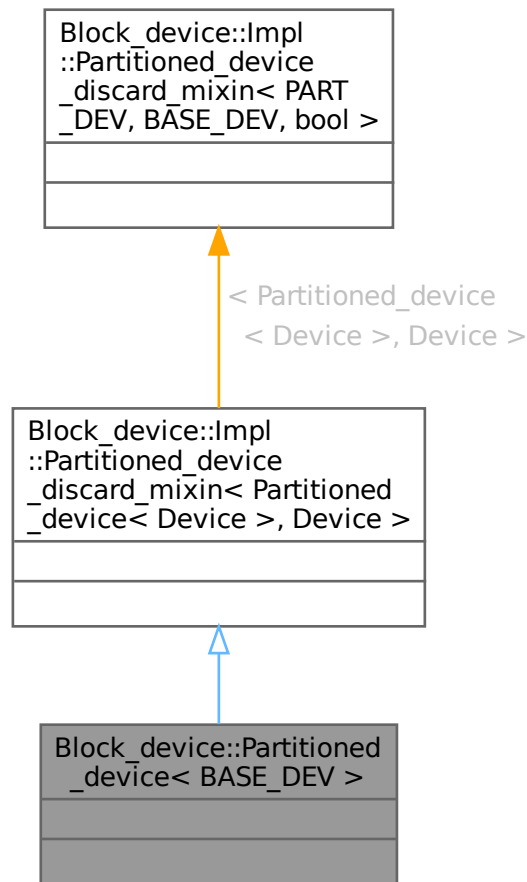
A partition device for the given device interface.

```
#include <part_device.h>
```

Inheritance diagram for Block_device::Partitioned_device< BASE_DEV >:



Collaboration diagram for Block_device::Partitioned_device< BASE_DEV >:



16.15.1 Detailed Description

```
template<typename BASE_DEV = Device>
class Block_device::Partitioned_device< BASE_DEV >
```

A partition device for the given device interface.

Template Parameters

<i>BASE_DEV</i>	Class defining the device interface. Attention: this is not the class implementing the device itself.
-----------------	---

Definition at line 91 of file [part_device.h](#).

The documentation for this class was generated from the following file:

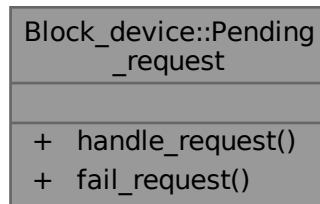
- `l4/libblock-device/part_device.h`

16.16 Block_device::Pending_request Struct Reference

Interface for pending requests.

```
#include <request.h>
```

Collaboration diagram for Block_device::Pending_request:



Public Member Functions

- virtual int [handle_request](#) ()=0
Callback used when the request is ready for processing.
- virtual void [fail_request](#) ()=0
Callback used when a request is dropped from the queue.

16.16.1 Detailed Description

Interface for pending requests.

Definition at line 14 of file [request.h](#).

16.16.2 Member Function Documentation

16.16.2.1 fail_request()

```
virtual void Block_device::Pending_request::fail_request () [pure virtual]
```

Callback used when a request is dropped from the queue.

The function is called for notification only. The request will be destroyed.

16.16.2.2 handle_request()

```
virtual int Block_device::Pending_request::handle_request () [pure virtual]
```

Callback used when the request is ready for processing.

Return values

<i>L4_EOK</i>	Request successfully issued. The callee has taken ownership of the request.
<i>-L4_EBUSY</i>	Device is still busy. The callee must not requeue the request as it will remain in the queue.
<	0 Other fatal error. The caller may dispose of the request.

The documentation for this struct was generated from the following file:

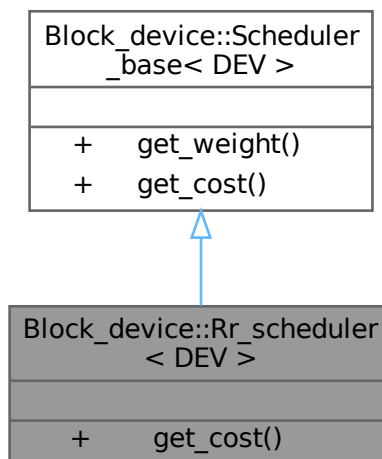
- l4/libblock-device/request.h

16.17 Block_device::Rr_scheduler< DEV > Struct Template Reference

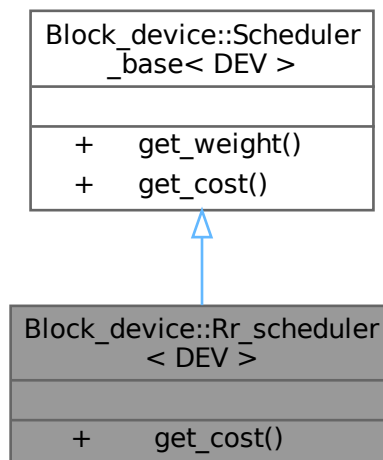
Round Robin scheduler class.

```
#include <scheduler.h>
```

Inheritance diagram for Block_device::Rr_scheduler< DEV >:



Collaboration diagram for `Block_device::Rr_scheduler< DEV >`:



Public Member Functions

- `l4_size_t get_cost (Pending_request const &)` override
Return the cost of the pending request.

Public Member Functions inherited from `Block_device::Scheduler_base< DEV >`

- virtual `l4_size_t get_weight (Client_type const *)=0`
Return the weight of the client.

16.17.1 Detailed Description

```
template<typename DEV>
struct Block_device::Rr_scheduler< DEV >
```

Round Robin scheduler class.

All clients have fixed weight of 1 and all requests have fixed cost of 1, giving thus each client one scheduling chance per scheduling round.

Definition at line 340 of file [scheduler.h](#).

The documentation for this struct was generated from the following file:

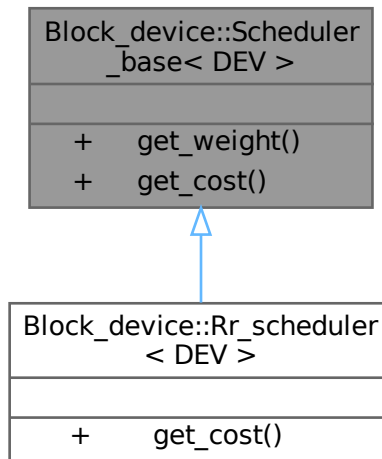
- `l4/libblock-device/scheduler.h`

16.18 Block_device::Scheduler_base< DEV > Class Template Reference

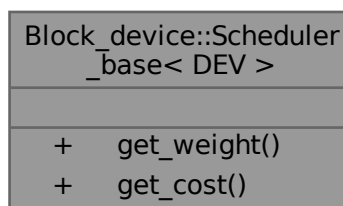
Scheduler base class.

```
#include <scheduler.h>
```

Inheritance diagram for Block_device::Scheduler_base< DEV >:



Collaboration diagram for Block_device::Scheduler_base< DEV >:



Public Member Functions

- virtual [l4_size_t](#) **get_weight** (Client_type const *)=0
Return the weight of the client.
- virtual [l4_size_t](#) **get_cost** ([Pending_request](#) const &)=0
Return the cost of the pending request.

16.18.1 Detailed Description

```
template<typename DEV>
class Block_device::Scheduler_base< DEV >
```

Scheduler base class.

Derive from this class and override `get_weight()` and `get_cost()` to implement the desired scheduling algorithm.

The interpretation of the weight function depends on the definition of the cost function. For example, if the cost of each request is fixed to be 1, the weight then says how many requests per scheduling round the client can process. If the weight of each client is also fixed to be 1, it will result in the Round Robin scheduler. If the request cost derives from the size of data the request operates on, the weight determines a data limit.

Definition at line 35 of file [scheduler.h](#).

The documentation for this class was generated from the following file:

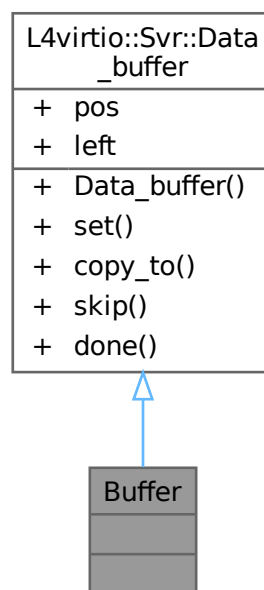
- [l4/libblock-device/scheduler.h](#)

16.19 Buffer Struct Reference

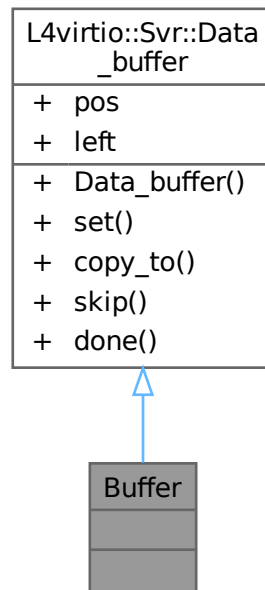
Data buffer used to transfer packets.

```
#include <virtio_net_buffer.h>
```

Inheritance diagram for Buffer:



Collaboration diagram for Buffer:



Additional Inherited Members

Public Member Functions inherited from `L4virtio::Svr::Data_buffer`

- `template<typename T>`
`Data_buffer` (`T *p`)
Create buffer for object p.
- `template<typename T>`
`void set` (`T *p`)
Set buffer for object p.
- `l4_uint32_t copy_to` (`Data_buffer *dst`, `l4_uint32_t max=UINT_MAX`)
Copy contents from this buffer to the destination buffer.
- `l4_uint32_t skip` (`l4_uint32_t bytes`)
Skip given number of bytes in this buffer.
- `bool done` () const
Check if there are no more bytes left in the buffer.

Data Fields inherited from `L4virtio::Svr::Data_buffer`

- `char * pos`
Current buffer position.
- `l4_uint32_t left`
Bytes left in buffer.

16.19.1 Detailed Description

Data buffer used to transfer packets.

Definition at line 19 of file [virtio_net_buffer.h](#).

The documentation for this struct was generated from the following file:

- pkg/virtio-net-switch/server/switch/virtio_net_buffer.h

16.20 `cxx::arith::Ld< V >` Struct Template Reference

Computes the binary logarithm of the given number at compile time.

```
#include <arith>
```

Collaboration diagram for `cxx::arith::Ld< V >`:



16.20.1 Detailed Description

```
template<unsigned long V>
struct cxx::arith::Ld< V >
```

Computes the binary logarithm of the given number at compile time.

Parameters

<i>val</i>	Number whose logarithm to compute, must be greater than zero.
------------	---

Returns

The binary logarithm of *val*.

Definition at line 48 of file [arith](#).

The documentation for this struct was generated from the following file:

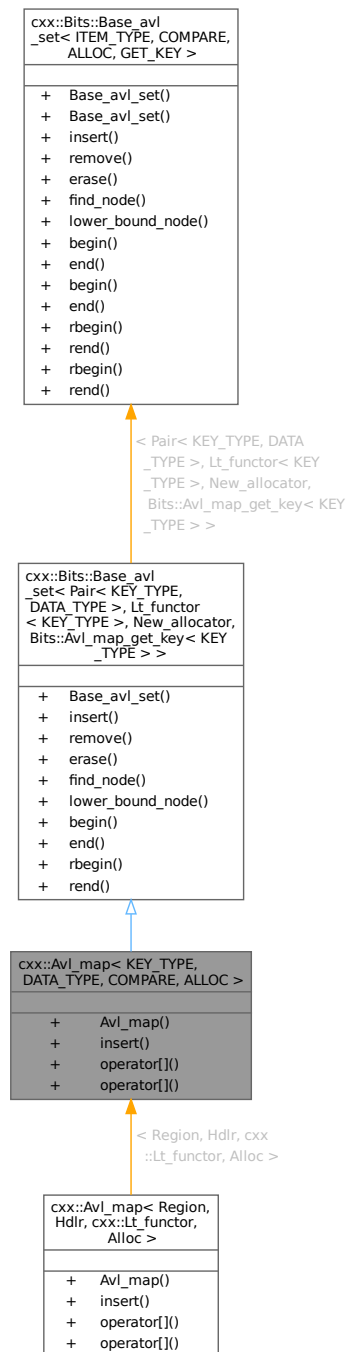
- l4/cxx/arith

16.21 cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC > Class Template Reference

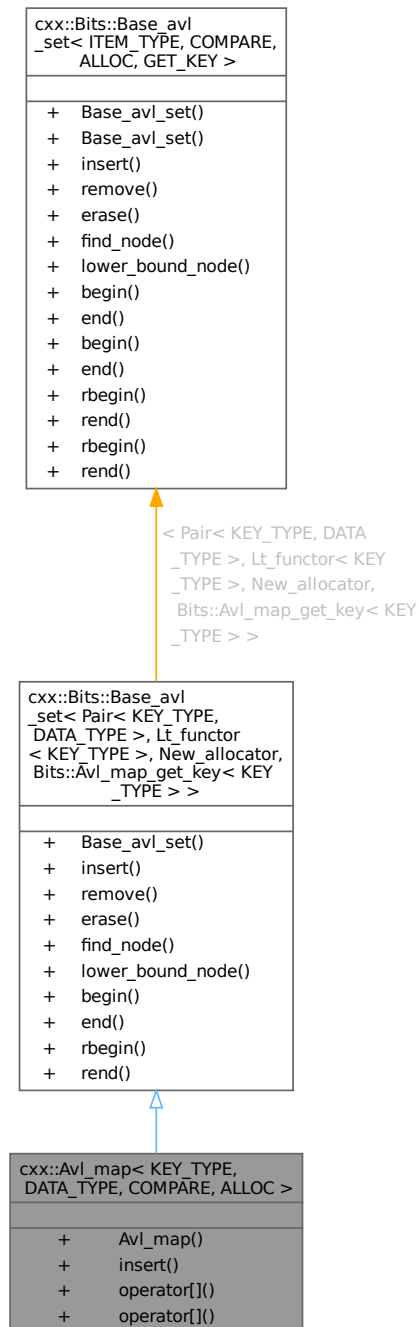
AVL tree based associative container.

```
#include <avl_map>
```

Inheritance diagram for cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC >:



Collaboration diagram for `cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC >`:



Public Types

- `typedef COMPARE< KEY_TYPE > Key_compare`
Type of the comparison functor.
- `typedef KEY_TYPE Key_type`
Type of the key values.
- `typedef DATA_TYPE Data_type`

Type of the data values.

- typedef `Base_type::Node` **Node**

Return type for find.

- typedef `Base_type::Node_allocator` **Node_allocator**

Type of the allocator.

Public Types inherited from

`cxx::Bits::Base_avl_set< Pair< KEY_TYPE, DATA_TYPE >, Lt_functor< KEY_TYPE >, New_allocator,`

- enum

Return status constants.

- typedef `Pair< KEY_TYPE, DATA_TYPE >` **Item_type**

Type for the items store in the set.

- typedef `Bits::Avl_map_get_key< KEY_TYPE >` **Get_key**

Key-getter type to derive the sort key of an internal node.

- typedef `Bits::Avl_map_get_key< KEY_TYPE >::Key_type` **Key_type**

Type of the sort key used for the items.

- typedef `Type_traits< Item_type >::Const_type` **Const_item_type**

Type used for const items within the set.

- typedef `Lt_functor< KEY_TYPE >` **Item_compare**

Type for the comparison functor.

- typedef `New_allocator< _Node >` **Node_allocator**

Type for the node allocator.

- typedef `Avl_set_iter< _Node, Item_type, Fwd >` **Iterator**

Forward iterator for the set.

- typedef `Avl_set_iter< _Node, Const_item_type, Fwd >` **Const_iterator**

Constant forward iterator for the set.

- typedef `Avl_set_iter< _Node, Item_type, Rev >` **Rev_iterator**

Backward iterator for the set.

- typedef `Avl_set_iter< _Node, Const_item_type, Rev >` **Const_rev_iterator**

Constant backward iterator for the set.

Public Member Functions

- `Avl_map` (`Node_allocator` const &alloc=`Node_allocator`())

Create an empty AVL-tree based map.

- `cxx::Pair< Iterator, int >` `insert` (`Key_type` const &key, `Data_type` const &data)

Insert a <key, data> pair into the map.

- `Data_type` const & `operator[]` (`Key_type` const &key) const

Get the data for the given key.

- `Data_type` & `operator[]` (`Key_type` const &key)

Get or insert data for the given key.

Public Member Functions inherited from

cxx::Bits::Base_avl_set< **Pair**< **KEY_TYPE**, **DATA_TYPE** >, **Lt_func**< **KEY_TYPE** >, **New_allocator**,

- **Base_avl_set** (**Node_allocator** const &alloc=**Node_allocator**())
Create a AVL-tree based set.
- **cxx::Pair**< **Iterator**, int > **insert** (**Item_type** const &item)
Insert an item into the set.
- int **remove** (**Key_type** const &item)
Remove an item from the set.
- int **erase** (**Key_type** const &item)
Erase the item with the given key.
- **Node** **find_node** (**Key_type** const &item) const
Lookup a node equal to item.
- **Node** **lower_bound_node** (**Key_type** const &key) const
Find the first node greater or equal to key.
- **Const_iterator** **begin** () const
Get the constant forward iterator for the first element in the set.
- **Const_iterator** **end** () const
Get the end marker for the constant forward iterator.
- **Const_rev_iterator** **rbegin** () const
Get the constant backward iterator for the last element in the set.
- **Const_rev_iterator** **rend** () const
Get the end marker for the constant backward iterator.

16.21.1 Detailed Description

```
template<typename KEY_TYPE, typename DATA_TYPE, template< typename A > class COMPARE = Lt_↔
func, template< typename B > class ALLOC = New_allocator>
class cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC >
```

AVL tree based associative container.

Template Parameters

<i>KEY_TYPE</i>	Type of the key values.
<i>DATA_TYPE</i>	Type of the data values.
<i>COMPARE</i>	Type comparison functor for the key values.
<i>ALLOC</i>	Type of the allocator used for the nodes.

Definition at line 45 of file [avl_map](#).

16.21.2 Constructor & Destructor Documentation

16.21.2.1 Avl_map()

```
template<typename KEY_TYPE, typename DATA_TYPE, template< typename A > class COMPARE = Lt_↔
func, template< typename B > class ALLOC = New_allocator>
cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC >::Avl_map (
    Node_allocator const & alloc = Node_allocator()) [inline]
```

Create an empty AVL-tree based map.

Parameters

<code>alloc</code>	The node allocator.
--------------------	---------------------

Definition at line 80 of file `avl_map`.

16.21.3 Member Function Documentation

16.21.3.1 `insert()`

```
template<typename KEY_TYPE, typename DATA_TYPE, template< typename A > class COMPARE = Lt_↔
functor, template< typename B > class ALLOC = New_allocator>
cxx::Pair< Iterator, int > cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC >::insert (
    Key_type const & key,
    Data_type const & data) [inline]
```

Insert a `<key, data>` pair into the map.

Parameters

<code>key</code>	The key value.
<code>data</code>	The data value to insert.

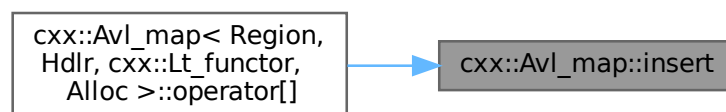
Returns

A pair of iterator (`first`) and return value (`second`). `second` will be 0 if the element was inserted into the set and `-#E_exist` if the key was already in the set and the set was therefore not updated. In both cases, `first` contains an iterator that points to the element. `second` may also be `-#E_nomem` when memory for the new node could not be allocated. `first` is then invalid.

Definition at line 99 of file `avl_map`.

Referenced by `cxx::Avl_map< Region, Hdlr, cxx::Lt_functor, Alloc >::operator[]()`.

Here is the caller graph for this function:



16.21.3.2 `operator[]()` [1/2]

```
template<typename KEY_TYPE, typename DATA_TYPE, template< typename A > class COMPARE = Lt_↔
functor, template< typename B > class ALLOC = New_allocator>
Data_type & cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC >::operator[] (
    Key_type const & key) [inline]
```

Get or insert data for the given key.

Parameters

<i>key</i>	The key value to use for lookup.
------------	----------------------------------

Returns

If the item already exists, a reference to the data item. Otherwise a new data item is default-constructed and inserted under the given key before a reference is returned.

Definition at line 123 of file [avl_map](#).

16.21.3.3 operator[]() [2/2]

```
template<typename KEY_TYPE, typename DATA_TYPE, template< typename A > class COMPARE = Lt_↔
functor, template< typename B > class ALLOC = New_allocator>
Data_type const & cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC >::operator[] (
    Key_type const & key) const [inline]
```

Get the data for the given key.

Parameters

<i>key</i>	The key value to use for lookup.
------------	----------------------------------

Precondition

A <key, data> pair for the given key value must exist.

Definition at line 111 of file [avl_map](#).

The documentation for this class was generated from the following file:

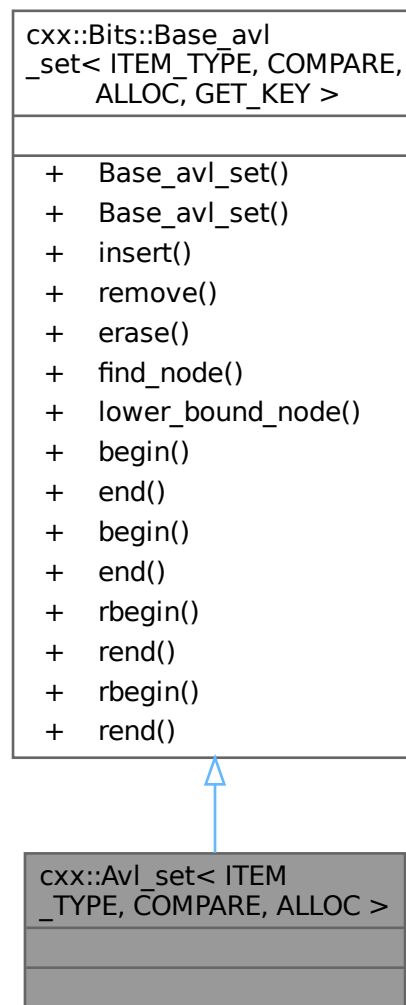
- I4/cxx/[avl_map](#)

16.22 cxx::Avl_set< ITEM_TYPE, COMPARE, ALLOC > Class Template Reference

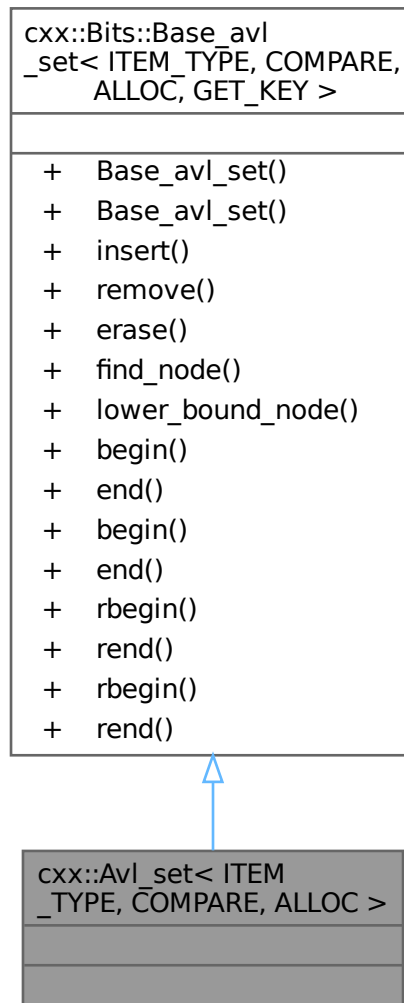
AVL set for simple comparable items.

```
#include <avl_set>
```


Inheritance diagram for `cxx::Avl_set< ITEM_TYPE, COMPARE, ALLOC >`:



Collaboration diagram for `cxx::Avl_set< ITEM_TYPE, COMPARE, ALLOC >`:



Additional Inherited Members

Public Types inherited from

`cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >`

- enum { `E_noent` = 2 , `E_exist` = 17 , `E_nomem` = 12 , `E_inval` = 22 }

Return status constants.

- typedef `ITEM_TYPE` **Item_type**

Type for the items store in the set.

- typedef `GET_KEY` **Get_key**

Key-getter type to derive the sort key of an internal node.

- typedef `GET_KEY::Key_type` **Key_type**

Type of the sort key used for the items.

- `typedef Type_traits< Item_type >::Const_type Const_item_type`
Type used for const items within the set.
- `typedef COMPARE Item_compare`
Type for the comparison functor.
- `typedef ALLOC< _Node > Node_allocator`
Type for the node allocator.
- `typedef Avl_set_iter< _Node, Item_type, Fwd > Iterator`
Forward iterator for the set.
- `typedef Avl_set_iter< _Node, Const_item_type, Fwd > Const_iterator`
Constant forward iterator for the set.
- `typedef Avl_set_iter< _Node, Item_type, Rev > Rev_iterator`
Backward iterator for the set.
- `typedef Avl_set_iter< _Node, Const_item_type, Rev > Const_rev_iterator`
Constant backward iterator for the set.

Public Member Functions inherited from

`cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >`

- `Base_avl_set (Node_allocator const &alloc=Node_allocator())`
Create a AVL-tree based set.
- `Base_avl_set (Base_avl_set const &o)`
Create a copy of an AVL-tree based set.
- `cxx::Pair< Iterator, int > insert (Item_type const &item)`
Insert an item into the set.
- `int remove (Key_type const &item)`
Remove an item from the set.
- `int erase (Key_type const &item)`
Erase the item with the given key.
- `Node find_node (Key_type const &item) const`
*Lookup a node equal to *item*.*
- `Node lower_bound_node (Key_type const &key) const`
*Find the first node greater or equal to *key*.*
- `Const_iterator begin () const`
Get the constant forward iterator for the first element in the set.
- `Const_iterator end () const`
Get the end marker for the constant forward iterator.
- `Iterator begin ()`
Get the mutable forward iterator for the first element of the set.
- `Iterator end ()`
Get the end marker for the mutable forward iterator.
- `Const_rev_iterator rbegin () const`
Get the constant backward iterator for the last element in the set.
- `Const_rev_iterator rend () const`
Get the end marker for the constant backward iterator.
- `Rev_iterator rbegin ()`
Get the mutable backward iterator for the last element of the set.
- `Rev_iterator rend ()`
Get the end marker for the mutable backward iterator.

16.22.1 Detailed Description

```
template<typename ITEM_TYPE, class COMPARE = Lt_functor<ITEM_TYPE>, template< typename A >
class ALLOC = New_allocator>
class cxx::Avl_set< ITEM_TYPE, COMPARE, ALLOC >
```

AVL set for simple comparable items.

The AVL set can store any kind of items where a partial order is defined. The default relation is defined by the '<' operator.

Template Parameters

<i>ITEM_TYPE</i>	The type of the items to be stored in the set.
<i>COMPARE</i>	The relation to define the partial order, default is to use operator '<'.
<i>ALLOC</i>	The allocator to use for the nodes of the AVL set.

Definition at line [465](#) of file [avl_set](#).

The documentation for this class was generated from the following file:

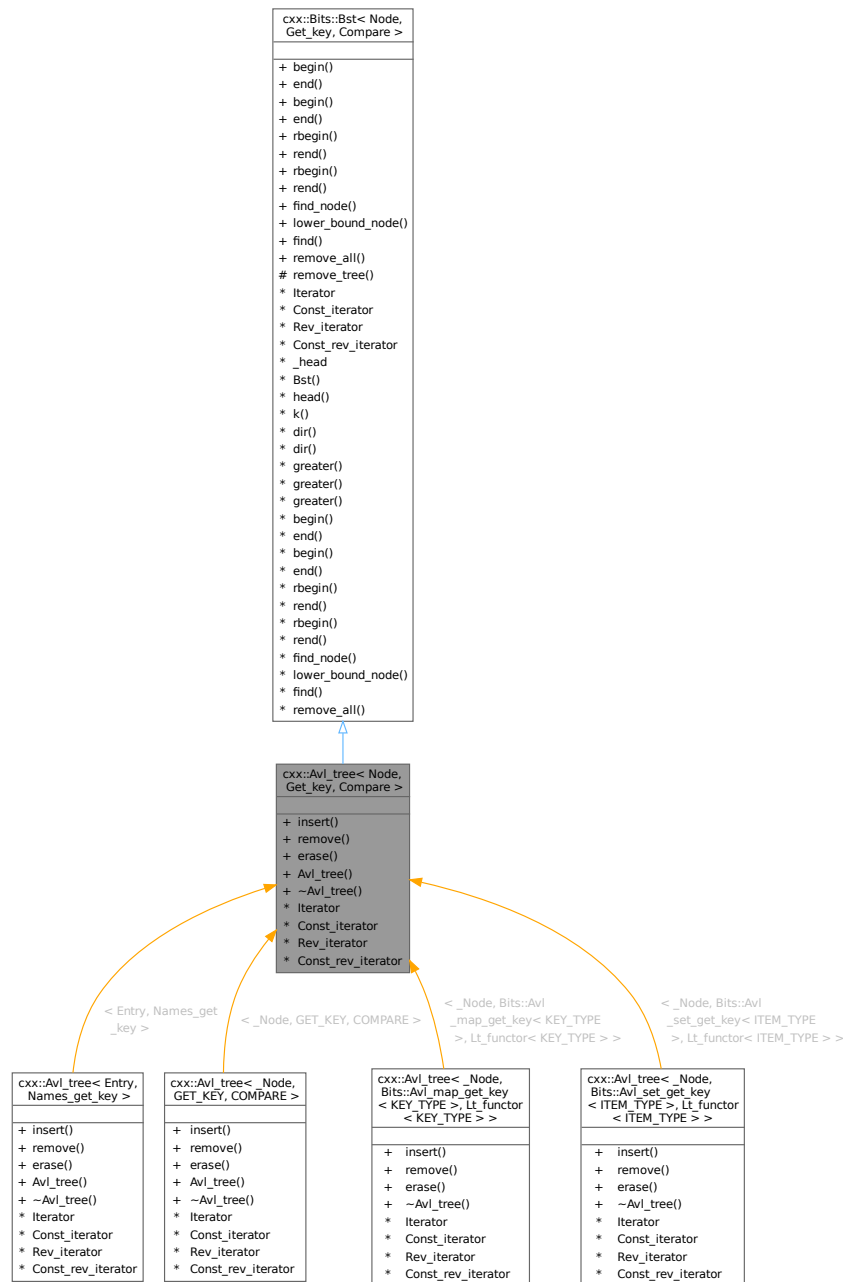
- [l4/cxx/avl_set](#)

16.23 cxx::Avl_tree< Node, Get_key, Compare > Class Template Reference

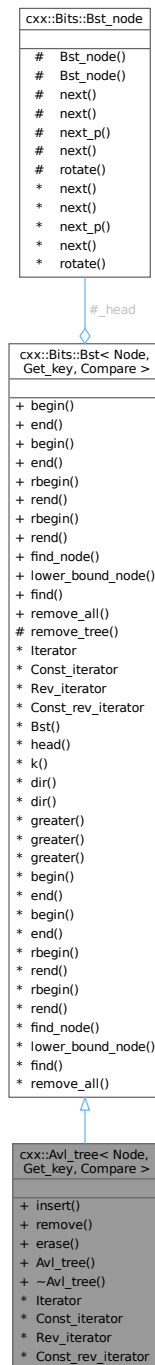
A generic AVL tree.

```
#include <avl_tree>
```

Inheritance diagram for cxx::Avl_tree< Node, Get_key, Compare >:



Collaboration diagram for `cxx::Avl_tree< Node, Get_key, Compare >`:



Public Types

- typedef `Bst::Iterator` `Iterator`

- typedef [Bst::Const_iterator](#) **Const_iterator**
Constant forward iterator for the tree.
- typedef [Bst::Rev_iterator](#) **Rev_iterator**
Backward iterator for the tree.
- typedef [Bst::Const_rev_iterator](#) **Const_rev_iterator**
Constant backward iterator for the tree.

Public Types inherited from [cxx::Bits::Bst< Node, Get_key, Compare >](#)

- typedef [Get_key::Key_type](#) **Key_type**
The type of key values used to generate the total order of the elements.
- typedef [Type_traits< \[Key_type\]\(#\) >::Param_type](#) **Key_param_type**
The type for key parameters.
- typedef [Fwd](#) **Fwd_iter_ops**
Helper for building forward iterators for different wrapper classes.
- typedef [Rev](#) **Rev_iter_ops**
Helper for building reverse iterators for different wrapper classes.
- typedef [__Bst_iter< Node, Node, Fwd >](#) **Iterator**
Forward iterator.
- typedef [__Bst_iter< Node, Node const, Fwd >](#) **Const_iterator**
Constant forward iterator.
- typedef [__Bst_iter< Node, Node, Rev >](#) **Rev_iterator**
Backward iterator.
- typedef [__Bst_iter< Node, Node const, Rev >](#) **Const_rev_iterator**
Constant backward.

Public Member Functions

- [Pair< Node *, bool >](#) [insert](#) (Node *new_node)
Insert a new node into this AVL tree.
- Node * [remove](#) (Key_param_type key)
Remove the node with key from the tree.
- Node * [erase](#) (Key_param_type key)
An alias for [remove\(\)](#).
- [Avl_tree](#) ()=default
Create an empty AVL tree.
- [~Avl_tree](#) () noexcept
Destroy the tree.

Public Member Functions inherited from `cxx::Bits::Bst< Node, Get_key, Compare >`

- `Const_iterator begin ()` const
Get the constant forward iterator for the first element in the set.
- `Const_iterator end ()` const
Get the end marker for the constant forward iterator.
- `Iterator begin ()`
Get the mutable forward iterator for the first element of the set.
- `Iterator end ()`
Get the end marker for the mutable forward iterator.
- `Const_rev_iterator rbegin ()` const
Get the constant backward iterator for the last element in the set.
- `Const_rev_iterator rend ()` const
Get the end marker for the constant backward iterator.
- `Rev_iterator rbegin ()`
Get the mutable backward iterator for the last element of the set.
- `Rev_iterator rend ()`
Get the end marker for the mutable backward iterator.

- `Node * find_node (Key_param_type key)` const
find the node with the given key.
- `Node * lower_bound_node (Key_param_type key)` const
Find the first node with a key not less than the given `key`.
- `Const_iterator find (Key_param_type key)` const
find the node with the given key.
- `template<typename FUNC>`
`void remove_all (FUNC &&callback)`
Clear the tree.

Additional Inherited Members

- `Bst ()`
Create an empty tree.

- `Node * head ()` const
Access the head node as object of type `Node`.

Static Protected Member Functions inherited from `cxx::Bits::Bst< Node, Get_key, Compare >`

- `template<typename FUNC>`
`static void remove_tree (Bst_node *head, FUNC &&callback)`
Remove all elements in the subtree of head.

- `static Key_type k (Bst_node const *n)`
Get the key value of `n`.

- `static Dir dir (Key_param_type l, Key_param_type r)`

Get the direction to go from l to search for r .

- static `Dir dir (Key_param_type l, Bst_node const *r)`

Get the direction to go from l to search for r .

- static bool **greater** (Key_param_type l, Key_param_type r)

Is l greater than r .

- static bool **greater** (Key_param_type l, Bst_node const *r)

Is l greater than r .

- static bool **greater** (Bst_node const *l, Bst_node const *r)

Is l greater than r .

- `Bst_node * _head`

The head pointer of the tree.

16.23.1 Detailed Description

```
template<typename Node, typename Get_key, typename Compare = Lt_functor<typename Get_key::Key↵
_type>>
class cxx::Avl_tree< Node, Get_key, Compare >
```

A generic AVL tree.

Template Parameters

<i>Node</i>	The data type of the nodes (must inherit from Avl_tree_node).
<i>Get_key</i>	The meta function to get the key value from a node. The implementation uses <code>Get_key::key↵_of(ptr_to_node)</code> . The type of the key values must be defined in <code>Get_key::Key_type</code> .
<i>Compare</i>	Binary relation to establish a total order for the nodes of the tree. <code>Compare() (l, r)</code> must return true if the key l is smaller than the key r .

This implementation does not provide any memory management. It is the responsibility of the caller to allocate nodes before inserting them and to free them when they are removed or when the tree is destroyed. Conversely, the caller must also ensure that nodes are removed from the tree before they are destroyed.

Examples

[tmpfs/lib/src/fs.cc](#).

Definition at line 100 of file [avl_tree](#).

16.23.2 Member Typedef Documentation

16.23.2.1 Iterator

```
template<typename Node, typename Get_key, typename Compare = It_functor<typename Get_key::←
Key_type>>
typedef Bst::Iterator cxx::Avl_tree< Node, Get_key, Compare >::Iterator
```

Forward iterator for the tree.

Definition at line 130 of file [avl_tree](#).

16.23.3 Member Function Documentation

16.23.3.1 insert()

```
template<typename Node, typename Get_key, class Compare>
Pair< Node *, bool > cxx::Avl_tree< Node, Get_key, Compare >::insert (
    Node * new_node)
```

Insert a new node into this AVL tree.

Parameters

<i>new_node</i>	A pointer to the new node.
-----------------	----------------------------

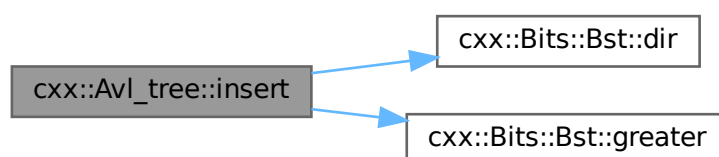
Returns

A pair, with *second* set to `true` and *first* pointing to *new_node*, on success. If there is already a node with the same key then *first* points to this node and *second* is 'false'.

Definition at line 220 of file [avl_tree](#).

References [cxx::Bits::Bst< Node, Get_key, Compare >::dir\(\)](#), [cxx::Bits::Bst< Node, Get_key, Compare >::greater\(\)](#), and [cxx::Bits::Direction::N](#).

Here is the call graph for this function:



16.23.3.2 remove()

```
template<typename Node, typename Get_key, class Compare>
Node * cxx::Avl_tree< Node, Get_key, Compare >::remove (
    Key_param_type key) [inline]
```

Remove the node with *key* from the tree.

Parameters

<i>key</i>	The key to the node to remove.
------------	--------------------------------

Returns

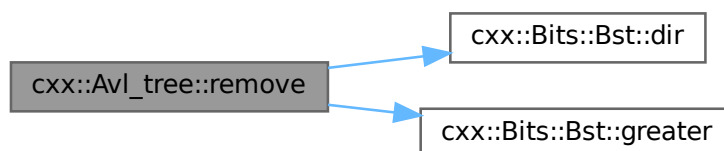
The pointer to the removed node on success, or 0 if no node with the *key* exists.

Definition at line 282 of file [avl_tree](#).

References [cxx::Bits::Bst< Node, Get_key, Compare >::dir\(\)](#), [cxx::Bits::Bst< Node, Get_key, Compare >::greater\(\)](#), [cxx::Bits::Direction::L](#), and [cxx::Bits::Direction::N](#).

Referenced by [cxx::Avl_tree< Entry, Names_get_key >::erase\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

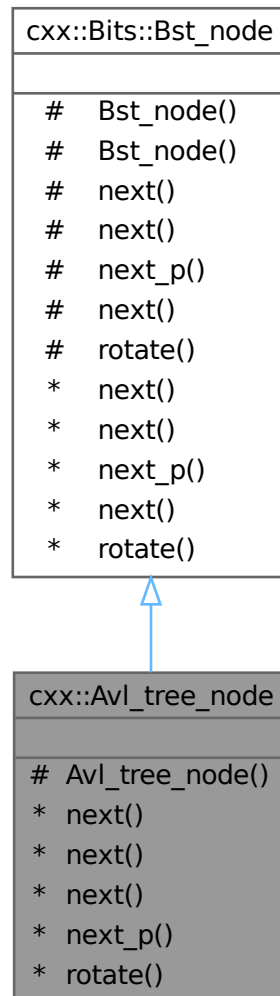
- [I4/cxx/avl_tree](#)

16.24 cxx::Avl_tree_node Class Reference

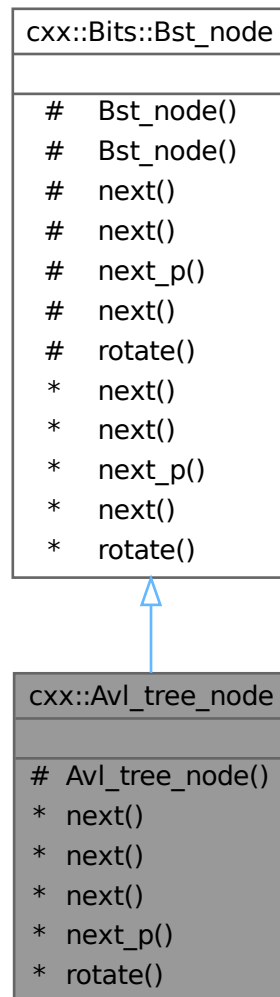
Node of an AVL tree.

```
#include <avl_tree>
```

Inheritance diagram for cxx::Avl_tree_node:



Collaboration diagram for `cxx::Avl_tree_node`:



Protected Member Functions

- **`Avl_tree_node()`**=default

Create an uninitialized node, this is what you should do.

Protected Member Functions inherited from `cxx::Bits::Bst_node`

- **`Bst_node()`**

Create uninitialized node.

- **`Bst_node(bool)`**

Create initialized node.

Additional Inherited Members

Static Protected Member Functions inherited from `cxx::Bits::Bst_node`

- static `Bst_node * next` (`Bst_node` const *p, `Direction` d)
Get next node in direction d.
- static void `next` (`Bst_node` *p, `Direction` d, `Bst_node` *n)
Set next node of p in direction d to n.
- static `Bst_node ** next_p` (`Bst_node` *p, `Direction` d)
Get pointer to link in direction d.
- template<typename Node>
static Node * `next` (`Bst_node` const *p, `Direction` d)
Get next node in direction d as type Node.
- static void `rotate` (`Bst_node` **t, `Direction` idir)
Rotate subtree t in the opposite direction of idir.

16.24.1 Detailed Description

Node of an AVL tree.

Examples

[tmpfs/lib/src/fs.cc](#).

Definition at line 29 of file [avl_tree](#).

The documentation for this class was generated from the following file:

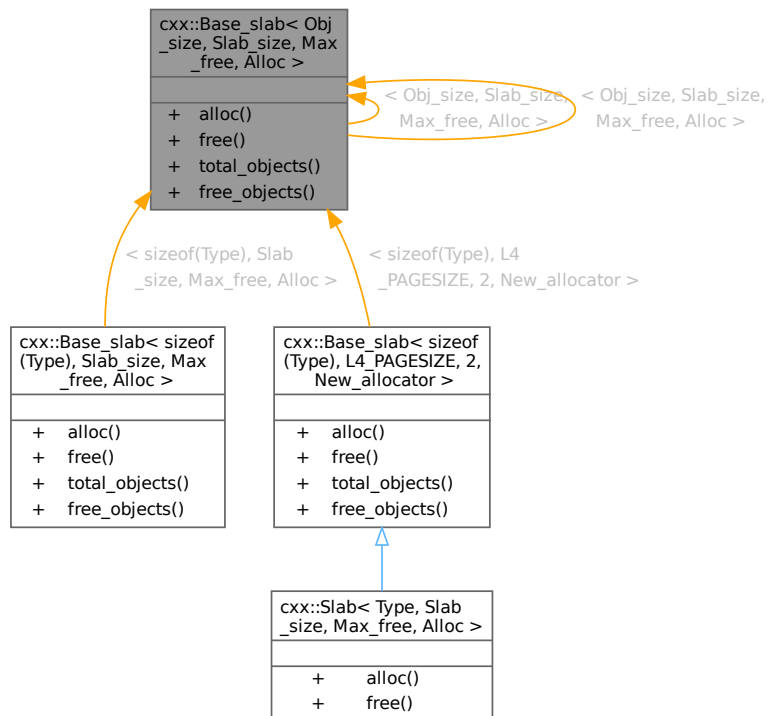
- [l4/cxx/avl_tree](#)

16.25 `cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >` Class Template Reference

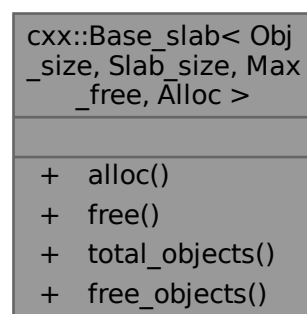
Basic slab allocator.

```
#include <slab_alloc>
```

Inheritance diagram for `cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >`:



Collaboration diagram for `cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >`:



Data Structures

- struct [Slab_i](#)
Type of a slab.

Public Types

- enum { `object_size` = `Obj_size` , `slab_size` = `Slab_size` , `objects_per_slab` = (`Slab_size` - `sizeof(Slab_head)`) / `object_size` , `max_free_slabs` = `Max_free` }
- typedef `Alloc< Slab_i >` **`Slab_alloc`**
Type of the backend allocator.

Public Member Functions

- void * `alloc` () noexcept
Allocate a new object.
- void `free` (void * `_o`) noexcept
Free the given object (`_o`).
- unsigned `total_objects` () const noexcept
Get the total number of objects managed by the slab allocator.
- unsigned `free_objects` () const noexcept
Get the number of objects which can be allocated before a new empty slab needs to be added to the slab allocator.

16.25.1 Detailed Description

```
template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A > class Alloc
= New_allocator>
class cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >
```

Basic slab allocator.

Template Parameters

<i>Obj_size</i>	The size of the objects managed by the allocator (in bytes).
<i>Slab_size</i>	The size of a slab (in bytes).
<i>Max_free</i>	The maximum number of free slabs. When this limit is reached slabs are freed, provided that the backend allocator supports allocated memory to be freed.
<i>Alloc</i>	The backend allocator used to allocate slabs.

Definition at line 31 of file `slab_alloc`.

16.25.2 Member Enumeration Documentation

16.25.2.1 anonymous enum

```
template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A >
class Alloc = New_allocator>
anonymous enum
```

Enumerator

object_size	Size of an object.
slab_size	Size of a slab.
objects_per_slab	Objects per slab.
max_free_slabs	Maximum number of free slabs.

Definition at line 65 of file [slab_alloc](#).

16.25.3 Member Function Documentation

16.25.3.1 alloc()

```
template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A >
class Alloc = New_allocator>
void * cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >::alloc () [inline], [noexcept]
```

Allocate a new object.

Returns

A pointer to the new object if the allocation succeeds, or 0 on failure to acquire memory from the backend allocator when the slab cache memory is already exhausted.

Note

The user is responsible for initializing the object.

Definition at line 207 of file [slab_alloc](#).

16.25.3.2 free()

```
template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A >
class Alloc = New_allocator>
void cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >::free (
    void * _o) [inline], [noexcept]
```

Free the given object (_o).

Precondition

The object must have been allocated with this allocator.

Definition at line 246 of file [slab_alloc](#).

16.25.3.3 `free_objects()`

```
template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A >
class Alloc = New_allocator>
unsigned cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >::free_objects () const [inline],
[noexcept]
```

Get the number of objects which can be allocated before a new empty slab needs to be added to the slab allocator.

Returns

The number of free objects in the slab allocator.

Definition at line 308 of file [slab_alloc](#).

16.25.3.4 `total_objects()`

```
template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A >
class Alloc = New_allocator>
unsigned cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >::total_objects () const [inline],
[noexcept]
```

Get the total number of objects managed by the slab allocator.

Returns

The number of objects managed by the allocator (including the free objects).

Definition at line 299 of file [slab_alloc](#).

The documentation for this class was generated from the following file:

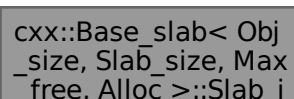
- `l4/cxx/slab_alloc`

16.26 `cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >::Slab_i` Struct Reference

Type of a slab.

```
#include <slab_alloc>
```

Collaboration diagram for `cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >::Slab_i`:



```
graph TD
    S["cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >::Slab_i"]
    S --- B1[" "]
    S --- B2[" "]
    style B1 fill:none,stroke:none
    style B2 fill:none,stroke:none
```

16.26.1 Detailed Description

```
template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A > class Alloc
= New_allocator>
struct cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >::Slab_i
```

Type of a slab.

Definition at line 86 of file [slab_alloc](#).

The documentation for this struct was generated from the following file:

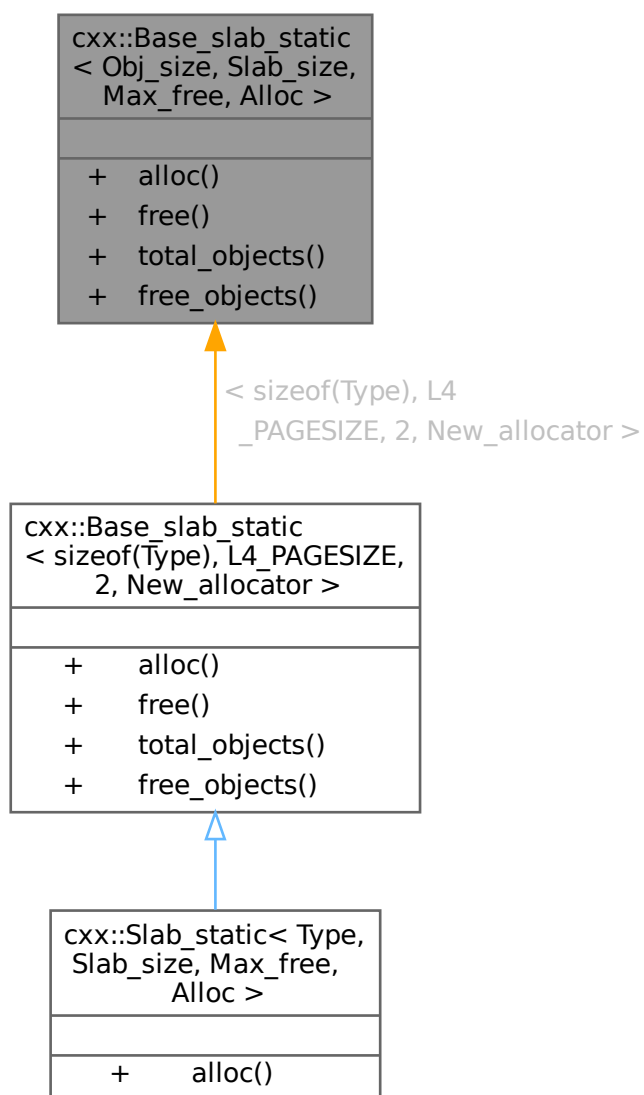
- l4/cxx/slab_alloc

16.27 cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc > Class Template Reference

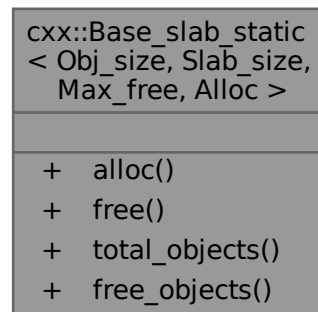
Merged slab allocator (allocators for objects of the same size are merged together).

```
#include <slab_alloc>
```

Inheritance diagram for cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >:



Collaboration diagram for `cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >`:



Public Types

- enum { `object_size` = `Obj_size` , `slab_size` = `Slab_size` , `objects_per_slab` = `_A::objects_per_slab` , `max_free_slabs` = `Max_free` }

Public Member Functions

- void * `alloc` () noexcept
Allocate an object.
- void `free` (void *p) noexcept
Free the given object (p).
- unsigned `total_objects` () const noexcept
Get the total number of objects managed by the slab allocator.
- unsigned `free_objects` () const noexcept
Get the number of free objects in the slab allocator.

16.27.1 Detailed Description

`template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A > class Alloc = New_allocator>`

`class cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >`

Merged slab allocator (allocators for objects of the same size are merged together).

Template Parameters

<i>Obj_size</i>	The size of an object managed by the slab allocator.
<i>Slab_size</i>	The size of a slab.
<i>Max_free</i>	The maximum number of free slabs.

<i>Alloc</i>	The allocator for the slabs.
--------------	------------------------------

This slab allocator class is useful for merging slab allocators with the same parameters (equal `Obj_size`, `Slab_size`, `Max_free`, and `Alloc` parameters) together and share the overhead for the slab caches among all equal-sized objects.

Definition at line 388 of file [slab_alloc](#).

16.27.2 Member Enumeration Documentation

16.27.2.1 anonymous enum

```
template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A >
class Alloc = New_allocator>
anonymous enum
```

Enumerator

<code>object_size</code>	Size of an object.
<code>slab_size</code>	Size of a slab.
<code>objects_per_slab</code>	Number of objects per slab.
<code>max_free_slabs</code>	Maximum number of free slabs.

Definition at line 395 of file [slab_alloc](#).

16.27.3 Member Function Documentation

16.27.3.1 `alloc()`

```
template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A >
class Alloc = New_allocator>
void * cxx::Base\_slab\_static< Obj_size, Slab_size, Max_free, Alloc >::alloc () [inline],
[noexcept]
```

Allocate an object.

Note

The user is responsible for initializing the object.

Definition at line 412 of file [slab_alloc](#).

16.27.3.2 `free()`

```
template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A >
class Alloc = New_allocator>
void cxx::Base\_slab\_static< Obj_size, Slab_size, Max_free, Alloc >::free (
    void * p) [inline], [noexcept]
```

Free the given object (`p`).

Parameters

<i>p</i>	The pointer to the object to free.
----------	------------------------------------

Precondition

p must be a pointer to an object allocated by this allocator.

Definition at line 420 of file [slab_alloc](#).

16.27.3.3 free_objects()

```
template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A >
class Alloc = New_allocator>
unsigned cxx::Base\_slab\_static< Obj_size, Slab_size, Max_free, Alloc >::free_objects () const
[inline], [noexcept]
```

Get the number of free objects in the slab allocator.

Returns

The number of free objects in all free and partially used slabs managed by this allocator.

Note

The value is the merged value for all equal parameterized [Base_slab_static](#) instances.

Definition at line 440 of file [slab_alloc](#).

16.27.3.4 total_objects()

```
template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A >
class Alloc = New_allocator>
unsigned cxx::Base\_slab\_static< Obj_size, Slab_size, Max_free, Alloc >::total_objects () const
[inline], [noexcept]
```

Get the total number of objects managed by the slab allocator.

Returns

The number of objects managed by the allocator (including the free objects).

Note

The value is the merged value for all equal parameterized [Base_slab_static](#) instances.

Definition at line 430 of file [slab_alloc](#).

The documentation for this class was generated from the following file:

- I4/cxx/slab_alloc

16.28 `cxx::Bitfield< T, LSB, MSB >` Class Template Reference

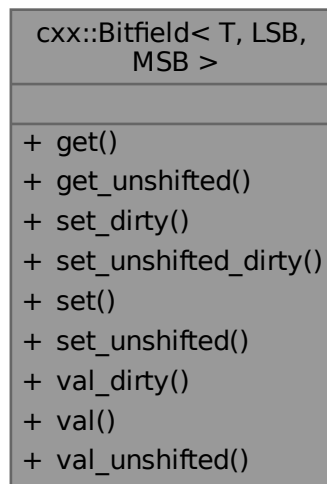
Definition for a member (part) of a bit field.

```
#include <bitfield>
```

Inheritance diagram for `cxx::Bitfield< T, LSB, MSB >`:



Collaboration diagram for `cxx::Bitfield< T, LSB, MSB >`:



Data Structures

- class [Value_base](#)
Internal helper type.
- class [Value](#)
Internal helper type.
- class [Value_unshifted](#)
Internal helper type.

Public Types

- enum { **Bits** = MSB + 1 - LSB , **Lsb** = LSB , **Msb** = MSB }
- enum **Masks** : Base_type { **Low_mask** = static_cast<Base_type>(~0ULL) >> (sizeof(Base_type)*8 - Bits) , **Mask** = Low_mask << Lsb }
- *Masks for bitwise operation on internal parts of a bitfield.*
- typedef Best_type< **Bits** >::Type **Bits_type**
Type to hold at least **Bits** bits.
- typedef Best_type< **Bits**+**Lsb** >::Type **Shift_type**
Type to hold at least **Bits** + **Lsb** bits.
- typedef **Value**< Base_type & > **Ref**
Reference type to access the bits inside a raw bit field.
- typedef **Value**< Base_type volatile & > **Ref_volatile**
Volatile reference type to access the bits inside a raw bit field.
- typedef **Value**< Base_type const > **Val**
Value type to access the bits inside a raw bit field.
- typedef **Value_unshifted**< Base_type & > **Ref_unshifted**
Reference type to access the bits inside a raw bit field (in place).
- typedef **Value_unshifted**< Base_type volatile & > **Ref_unshifted_volatile**
Volatile reference type to access the bits inside a raw bit field (in place).
- typedef **Value_unshifted**< Base_type const > **Val_unshifted**
Value type to access the bits inside a raw bit field (in place).

Static Public Member Functions

- static constexpr **Bits_type** get (**Shift_type** val)
Get the bits out of *val*.
- static constexpr Base_type **get_unshifted** (**Shift_type** val)
Get the bits in place out of *val*.
- static constexpr Base_type **set_dirty** (Base_type dest, **Shift_type** val)
Set the bits corresponding to *val*.
- static constexpr Base_type **set_unshifted_dirty** (Base_type dest, **Shift_type** val)
Set the bits corresponding to *val*.
- static Base_type **set** (Base_type dest, **Bits_type** val)
Set the bits corresponding to *val*.
- static Base_type **set_unshifted** (Base_type dest, **Shift_type** val)
Set the bits corresponding to *val*.
- static constexpr Base_type **val_dirty** (**Shift_type** val)
Get the shifted bits for *val*.
- static constexpr Base_type **val** (**Bits_type** val)
Get the shifted bits for *val*.
- static constexpr Base_type **val_unshifted** (**Shift_type** val)
Get the shifted bits for *val*.

16.28.1 Detailed Description

```
template<typename T, unsigned LSB, unsigned MSB>
class cxx::Bitfield< T, LSB, MSB >
```

Definition for a member (part) of a bit field.

Parameters

<i>T</i>	The underlying type of the bit field.
<i>LSB</i>	The least significant bit of our bits.
<i>MSB</i>	The most significant bit of our bits.

Definition at line 24 of file [bitfield](#).

16.28.2 Member Typedef Documentation

16.28.2.1 Bits_type

```
template<typename T, unsigned LSB, unsigned MSB>
typedef Best_type<Bits>::Type cxx::Bitfield< T, LSB, MSB >::Bits_type
```

Type to hold at least [Bits](#) bits.

This type can handle all values that can be stored in this part of the bit field.

Definition at line 74 of file [bitfield](#).

16.28.2.2 Shift_type

```
template<typename T, unsigned LSB, unsigned MSB>
typedef Best_type<Bits+Lsb>::Type cxx::Bitfield< T, LSB, MSB >::Shift_type
```

Type to hold at least [Bits](#) + [Lsb](#) bits.

This type can handle all values that can be stored in this part of the bit field when they are at the target location ([Lsb](#) bits shifted to the left).

Definition at line 82 of file [bitfield](#).

16.28.3 Member Enumeration Documentation

16.28.3.1 anonymous enum

```
template<typename T, unsigned LSB, unsigned MSB>
anonymous enum
```

Enumerator

Bits	Number of bits.
Lsb	index of the LSB
Msb	index of the MSB

Definition at line 52 of file [bitfield](#).

16.28.3.2 Masks

```
template<typename T, unsigned LSB, unsigned MSB>
enum cxx::Bitfield::Masks : Base_type
```

[Masks](#) for bitwise operation on internal parts of a bitfield.

Enumerator

Low_mask	Mask value to get Bits bits.
Mask	Mask value to the bits out of a T.

Definition at line 60 of file [bitfield](#).

16.28.4 Member Function Documentation

16.28.4.1 `get()`

```
template<typename T, unsigned LSB, unsigned MSB>
constexpr Bits\_type cxx::Bitfield< T, LSB, MSB >::get (
    Shift\_type val) [inline], [static], [constexpr]
```

Get the bits out of [val](#).

Parameters

val	The raw value of the whole bit field.
---------------------	---------------------------------------

Returns

The bits form [Lsb](#) to [Msb](#) shifted to the right.

Definition at line 99 of file [bitfield](#).

16.28.4.2 `get_unshifted()`

```
template<typename T, unsigned LSB, unsigned MSB>
constexpr Base_type cxx::Bitfield< T, LSB, MSB >::get_unshifted (
    Shift\_type val) [inline], [static], [constexpr]
```

Get the bits in place out of [val](#).

Parameters

val	The raw value of the whole bit field.
---------------------	---------------------------------------

Returns

The bits from [Lsb](#) to [Msb](#) (unshifted).

This means other bits are masked out, however the result is not shifted to the right.

Definition at line 112 of file [bitfield](#).

16.28.4.3 `set()`

```
template<typename T, unsigned LSB, unsigned MSB>
Base_type cxx::Bitfield< T, LSB, MSB >::set (
    Base_type dest,
    Bits\_type val) [inline], [static]
```

Set the bits corresponding to [val](#).

Parameters

<i>dest</i>	The current value of the whole bit field.
<i>val</i>	The value to set into the bits.

Returns

The new value of the whole bit field.

Definition at line 161 of file `bitfield`.

16.28.4.4 `set_dirty()`

```
template<typename T, unsigned LSB, unsigned MSB>
constexpr Base_type cxx::Bitfield< T, LSB, MSB >::set_dirty (
    Base_type dest,
    Shift_type val) [inline], [static], [constexpr]
```

Set the bits corresponding to `val`.

Parameters

<i>dest</i>	The current value of the whole bit field.
<i>val</i>	The value to set into the bits.

Returns

The new value of the whole bit field.

Precondition

`val` must not contain more than `Bits` bits.

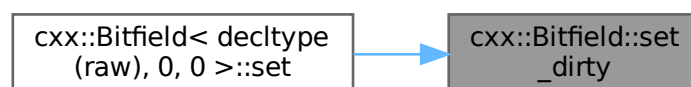
Note

This function does not mask `val` to the right number of bits.

Definition at line 127 of file `bitfield`.

Referenced by `cxx::Bitfield< decltype(raw), 0, 0 >::set()`.

Here is the caller graph for this function:



16.28.4.5 set_unshifted()

```
template<typename T, unsigned LSB, unsigned MSB>
Base_type cxx::Bitfield< T, LSB, MSB >::set_unshifted (
    Base_type dest,
    Shift_type val) [inline], [static]
```

Set the bits corresponding to `val`.

Parameters

<i>dest</i>	The current value of the whole bit field.
<i>val</i>	The value shifted <code>Lsb</code> bits to the left that shall be set into the bit field.

Returns

the new value of the whole bit field.

Definition at line 173 of file `bitfield`.

16.28.4.6 set_unshifted_dirty()

```
template<typename T, unsigned LSB, unsigned MSB>
constexpr Base_type cxx::Bitfield< T, LSB, MSB >::set_unshifted_dirty (
    Base_type dest,
    Shift_type val) [inline], [static], [constexpr]
```

Set the bits corresponding to `val`.

Parameters

<i>dest</i>	The current value of the whole bit field.
<i>val</i>	The value shifted <code>Lsb</code> bits to the left that shall be set into the bits.

Returns

The new value of the whole bit field.

Precondition

`val` must not contain more than `Bits` bits shifted `Lsb` bits to the left.

Note

This function does not mask `val` to the right number of bits.

Definition at line 147 of file `bitfield`.

Referenced by `cxx::Bitfield< decltype(raw), 0, 0 >::set_unshifted()`.

Here is the caller graph for this function:

**16.28.4.7 val()**

```

template<typename T, unsigned LSB, unsigned MSB>
constexpr Base_type cxx::Bitfield< T, LSB, MSB >::val (
    Bits_type val) [inline], [static], [constexpr]
  
```

Get the shifted bits for `val`.

Parameters

<code>val</code>	The value to set into the bits.
------------------	---------------------------------

Returns

The raw bit field value.

Definition at line 196 of file `bitfield`.

16.28.4.8 val_dirty()

```

template<typename T, unsigned LSB, unsigned MSB>
constexpr Base_type cxx::Bitfield< T, LSB, MSB >::val_dirty (
    Shift_type val) [inline], [static], [constexpr]
  
```

Get the shifted bits for `val`.

Parameters

<i>val</i>	The value to set into the bits.
------------	---------------------------------

Returns

The raw bit field value.

Precondition

val must not contain more than *Bits* bits.

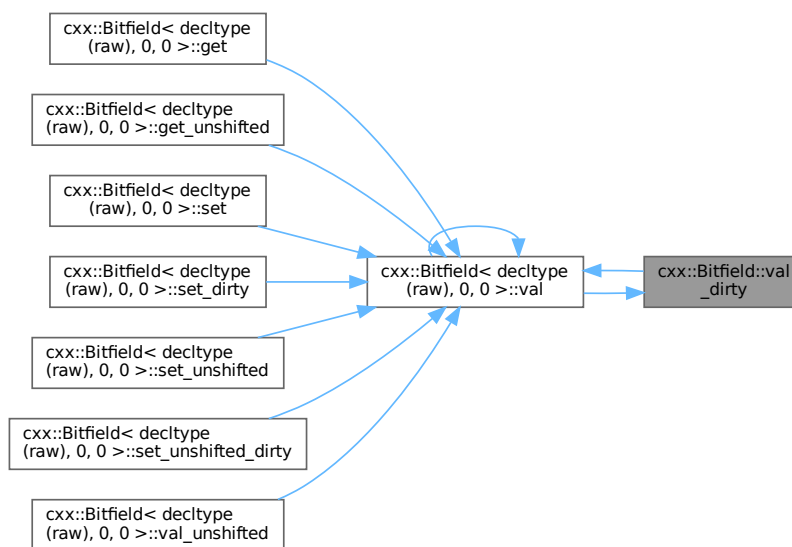
Note

This function does not mask *val* to the right number of bits.

Definition at line 187 of file [bitfield](#).

Referenced by [cxx::Bitfield< decltype\(raw\), 0, 0 >::val\(\)](#).

Here is the caller graph for this function:

**16.28.4.9 val_unshifted()**

```

template<typename T, unsigned LSB, unsigned MSB>
constexpr Base_type cxx::Bitfield< T, LSB, MSB >::val_unshifted (
    Shift_type val) [inline], [static], [constexpr]

```

Get the shifted bits for *val*.

Parameters

<i>val</i>	The value shifted <code>Lsb</code> bits to the left that shall be set into the bits.
------------	--

Returns

The raw bit field value.

Definition at line 206 of file [bitfield](#).

The documentation for this class was generated from the following file:

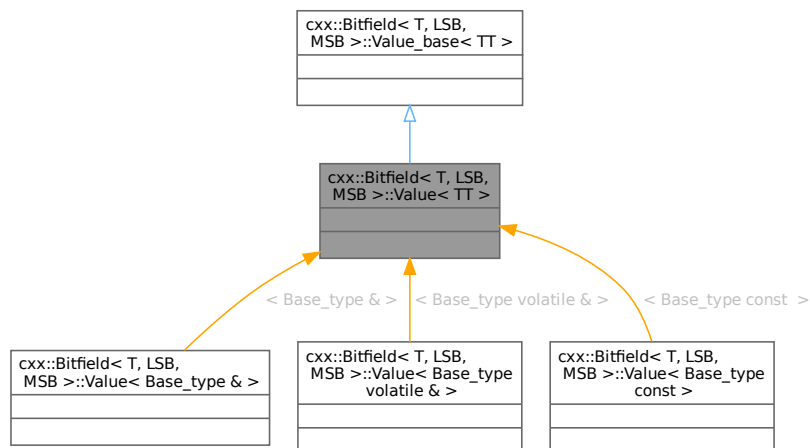
- `I4/cxx/bitfield`

16.29 `cxx::Bitfield< T, LSB, MSB >::Value< TT >` Class Template Reference

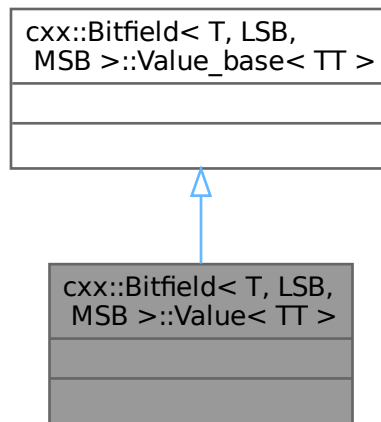
Internal helper type.

```
#include <bitfield>
```

Inheritance diagram for `cxx::Bitfield< T, LSB, MSB >::Value< TT >`:



Collaboration diagram for `cxx::Bitfield< T, LSB, MSB >::Value< TT >`:



16.29.1 Detailed Description

```

template<typename T, unsigned LSB, unsigned MSB>
template<typename TT>
class cxx::Bitfield< T, LSB, MSB >::Value< TT >
  
```

Internal helper type.

Definition at line 228 of file [bitfield](#).

The documentation for this class was generated from the following file:

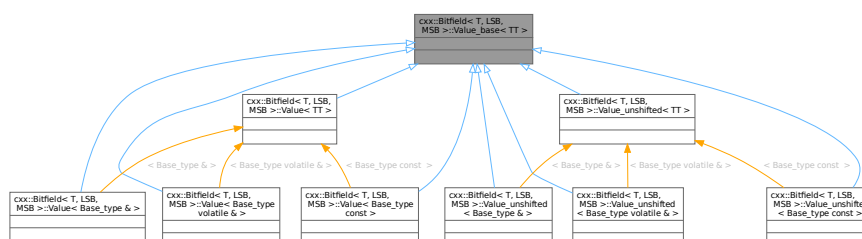
- `I4/cxx/bitfield`

16.30 `cxx::Bitfield< T, LSB, MSB >::Value_base< TT >` Class Template Reference

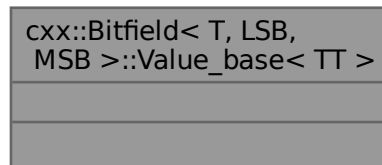
Internal helper type.

```
#include <bitfield>
```

Inheritance diagram for `cxx::Bitfield< T, LSB, MSB >::Value_base< TT >`:



Collaboration diagram for `cxx::Bitfield< T, LSB, MSB >::Value_base< TT >`:



16.30.1 Detailed Description

```

template<typename T, unsigned LSB, unsigned MSB>
template<typename TT>
class cxx::Bitfield< T, LSB, MSB >::Value_base< TT >

```

Internal helper type.

Definition at line 210 of file [bitfield](#).

The documentation for this class was generated from the following file:

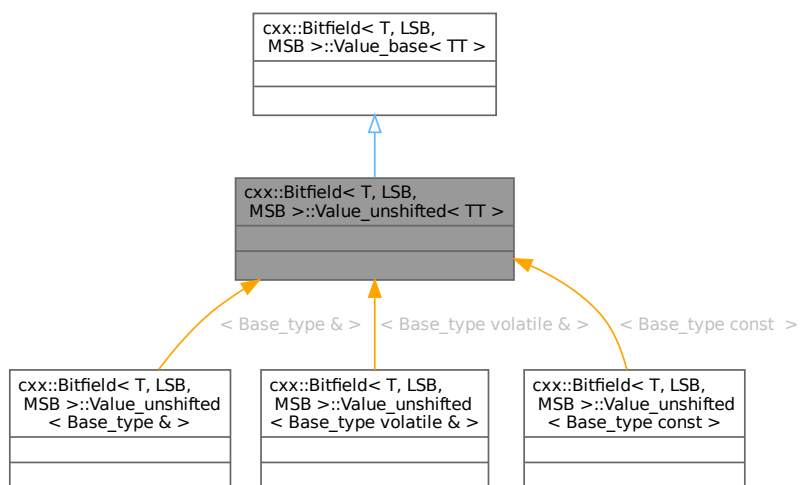
- `I4/cxx/bitfield`

16.31 `cxx::Bitfield< T, LSB, MSB >::Value_unshifted< TT >` Class Template Reference

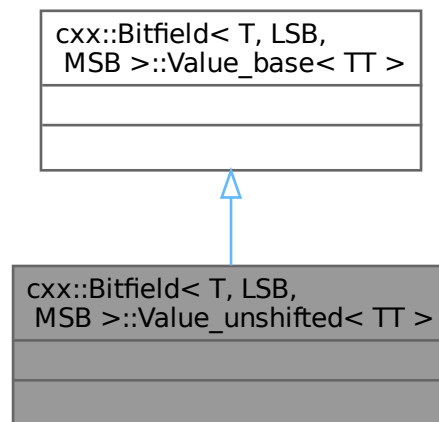
Internal helper type.

```
#include <bitfield>
```

Inheritance diagram for `cxx::Bitfield< T, LSB, MSB >::Value_unshifted< TT >`:



Collaboration diagram for `cxx::Bitfield< T, LSB, MSB >::Value_unshifted< TT >`:



16.31.1 Detailed Description

```

template<typename T, unsigned LSB, unsigned MSB>
template<typename TT>
class cxx::Bitfield< T, LSB, MSB >::Value_unshifted< TT >

```

Internal helper type.

Definition at line 241 of file [bitfield](#).

The documentation for this class was generated from the following file:

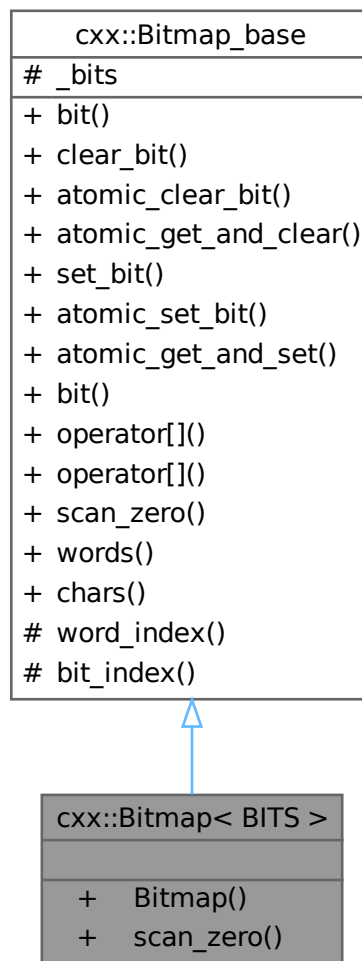
- `I4/cxx/bitfield`

16.32 `cxx::Bitmap< BITS >` Class Template Reference

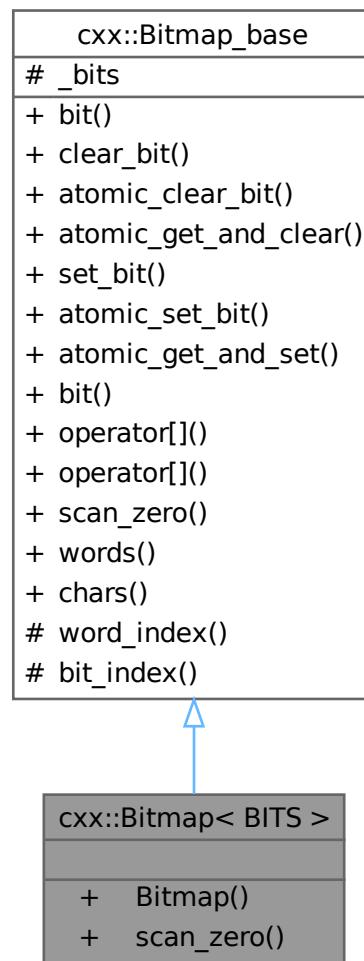
A static bitmap.

```
#include <bitmap>
```

Inheritance diagram for cxx::Bitmap< BITS >:



Collaboration diagram for `cxx::Bitmap< BITS >`:



Public Member Functions

- **Bitmap** () noexcept
Create a bitmap with `BITS` bits.
- long **scan_zero** (long start_bit=0) const noexcept
Scan for the first zero bit.

Public Member Functions inherited from `cxx::Bitmap_base`

- void **bit** (long bit, bool on) noexcept
Set the value of bit `bit` to on.
- void **clear_bit** (long bit) noexcept
Clear bit `bit`.
- void **atomic_clear_bit** (long bit) noexcept

- Clear bit `bit` atomically.*
- `word_type atomic_get_and_clear` (long `bit`) noexcept
Clear bit `bit` atomically and return old state.
- void `set_bit` (long `bit`) noexcept
Set bit `bit`.
- void `atomic_set_bit` (long `bit`) noexcept
Set bit `bit` atomically.
- `word_type atomic_get_and_set` (long `bit`) noexcept
Set bit `bit` atomically and return old state.
- `word_type bit` (long `bit`) const noexcept
Get the truth value of a bit.
- `word_type operator[]` (long `bit`) const noexcept
Get the bit at index `bit`.
- `Bit operator[]` (long `bit`) noexcept
Get the lvalue for the bit at index `bit`.
- long `scan_zero` (long `max_bit`, long `start_bit`=0) const noexcept
Scan for the first zero bit.

Additional Inherited Members

Static Public Member Functions inherited from `cx::Bitmap_base`

- static long `words` (long bits) noexcept
Get the number of `Words` that are used for the bitmap.
- static long `chars` (long bits) noexcept
Get the number of `chars` that are used for the bitmap.

Protected Types inherited from `cx::Bitmap_base`

- enum { `W_bits` = sizeof(`word_type`) * 8 , `C_bits` = 8 }
- typedef unsigned long `word_type`
Data type for each element of the bit buffer.

Static Protected Member Functions inherited from `cx::Bitmap_base`

- static unsigned `word_index` (unsigned `bit`)
Get the word index for the given bit.
- static unsigned `bit_index` (unsigned `bit`)
Get the bit index within `word_type` for the given bit.

Protected Attributes inherited from `cx::Bitmap_base`

- `word_type * _bits`
Pointer to the buffer storing the bits.

16.32.1 Detailed Description

```
template<int BITS>
class cx::Bitmap< BITS >
```

A static bitmap.

Template Parameters

<i>BITS</i>	The number of bits that shall be in the bitmap.
-------------	---

Definition at line 220 of file [bitmap](#).

16.32.2 Member Function Documentation

16.32.2.1 `scan_zero()`

```
template<int BITS>
long cxx::Bitmap< BITS >::scan_zero (
    long start_bit = 0) const [inline], [noexcept]
```

Scan for the first zero bit.

Parameters

<i>start_bit</i>	Hint at the number of the first bit to look at. Zero bits below <code>start_bit</code> may or may not be taken into account by the implementation.
------------------	--

Return values

<code>>=</code>	0 Number of first zero bit found.
<code>-1</code>	All bits at <code>start_bit</code> or higher are set.

Compared to [Bitmap_base::scan_zero\(\)](#), the upper bound is set to BITS.

Definition at line 365 of file [bitmap](#).

References [cxx::Bitmap_base::scan_zero\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

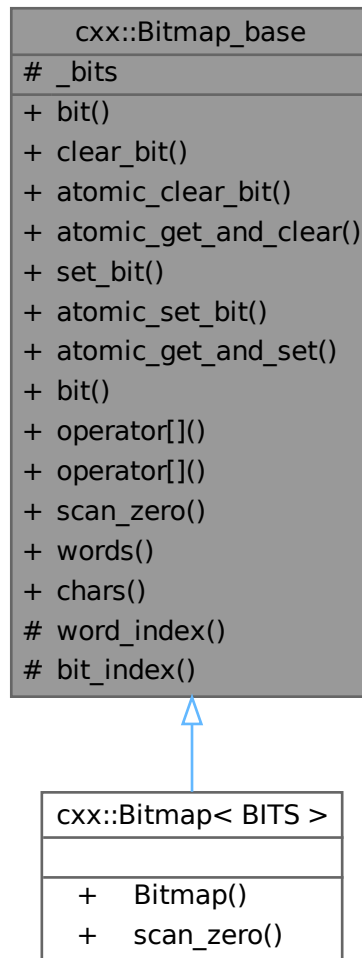
- `I4/cxx/bitmap`

16.33 cxx::Bitmap_base Class Reference

Basic bitmap abstraction.

```
#include <bitmap>
```

Inheritance diagram for cxx::Bitmap_base:



Collaboration diagram for `cxx::Bitmap_base`:

<code>cxx::Bitmap_base</code>
<code># _bits</code>
<code>+ bit()</code>
<code>+ clear_bit()</code>
<code>+ atomic_clear_bit()</code>
<code>+ atomic_get_and_clear()</code>
<code>+ set_bit()</code>
<code>+ atomic_set_bit()</code>
<code>+ atomic_get_and_set()</code>
<code>+ bit()</code>
<code>+ operator[]()</code>
<code>+ operator[]()</code>
<code>+ scan_zero()</code>
<code>+ words()</code>
<code>+ chars()</code>
<code># word_index()</code>
<code># bit_index()</code>

Data Structures

- class [Bit](#)
A writable bit in a bitmap.
- class [Word](#)
Helper abstraction for a word contained in the bitmap.
- class [Char](#)
Helper abstraction for a byte contained in the bitmap.

Public Member Functions

- void [bit](#) (long [bit](#), bool on) noexcept
Set the value of bit [bit](#) to on.
- void [clear_bit](#) (long [bit](#)) noexcept
Clear bit [bit](#).
- void [atomic_clear_bit](#) (long [bit](#)) noexcept
Clear bit [bit](#) atomically.
- [word_type](#) [atomic_get_and_clear](#) (long [bit](#)) noexcept
Clear bit [bit](#) atomically and return old state.
- void [set_bit](#) (long [bit](#)) noexcept
Set bit [bit](#).

- void `atomic_set_bit` (long `bit`) noexcept
Set bit `bit` atomically.
- `word_type` `atomic_get_and_set` (long `bit`) noexcept
Set bit `bit` atomically and return old state.
- `word_type` `bit` (long `bit`) const noexcept
Get the truth value of a bit.
- `word_type` `operator[]` (long `bit`) const noexcept
Get the bit at index `bit`.
- `Bit` `operator[]` (long `bit`) noexcept
Get the lvalue for the bit at index `bit`.
- long `scan_zero` (long `max_bit`, long `start_bit=0`) const noexcept
Scan for the first zero bit.

Static Public Member Functions

- static long `words` (long `bits`) noexcept
Get the number of `Words` that are used for the bitmap.
- static long `chars` (long `bits`) noexcept
Get the number of chars that are used for the bitmap.

Protected Types

- enum { `W_bits` = sizeof(`word_type`) * 8 , `C_bits` = 8 }
- typedef unsigned long `word_type`
Data type for each element of the bit buffer.

Static Protected Member Functions

- static unsigned `word_index` (unsigned `bit`)
Get the word index for the given bit.
- static unsigned `bit_index` (unsigned `bit`)
Get the bit index within `word_type` for the given bit.

Protected Attributes

- `word_type` * `_bits`
Pointer to the buffer storing the bits.

16.33.1 Detailed Description

Basic bitmap abstraction.

This abstraction keeps a pointer to a memory area that is used as bitmap.

Definition at line 18 of file `bitmap`.

16.33.2 Member Enumeration Documentation

16.33.2.1 anonymous enum

```
anonymous enum [protected]
```

Enumerator

W_bits	number of bits in word_type
C_bits	number of bits in char

Definition at line 26 of file [bitmap](#).

16.33.3 Member Function Documentation

16.33.3.1 `atomic_clear_bit()`

```
void cxx::Bitmap_base::atomic_clear_bit (
    long bit) [inline], [noexcept]
```

Clear bit [bit](#) atomically.

Use this function for multi-threaded access to the bitmap.

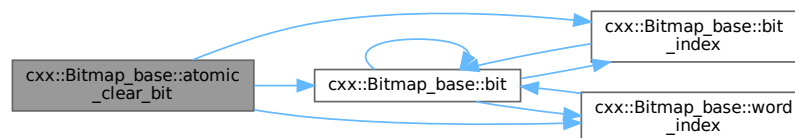
Parameters

<i>bit</i>	The number of the bit to clear.
------------	---------------------------------

Definition at line 269 of file [bitmap](#).

References [_bits](#), [bit\(\)](#), [bit_index\(\)](#), and [word_index\(\)](#).

Here is the call graph for this function:



16.33.3.2 `atomic_get_and_clear()`

```
Bitmap_base::word_type cxx::Bitmap_base::atomic_get_and_clear (
    long bit) [inline], [noexcept]
```

Clear bit [bit](#) atomically and return old state.

Use this function for multi-threaded access to the bitmap.

Parameters

<i>bit</i>	The number of the bit to clear.
------------	---------------------------------

Definition at line 279 of file [bitmap](#).

References [_bits](#), [bit\(\)](#), [bit_index\(\)](#), and [word_index\(\)](#).

Here is the call graph for this function:



16.33.3.3 atomic_get_and_set()

```

Bitmap_base::word_type cxx::Bitmap_base::atomic_get_and_set (
    long bit) [inline], [noexcept]

```

Set bit [bit](#) atomically and return old state.

Use this function for multi-threaded access to the bitmap.

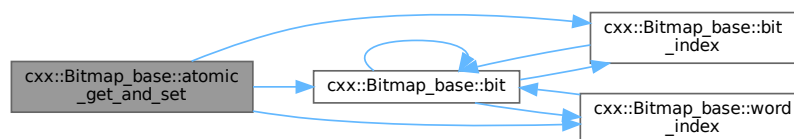
Parameters

<i>bit</i>	The number of the bit to set.
------------	-------------------------------

Definition at line 308 of file [bitmap](#).

References [_bits](#), [bit\(\)](#), [bit_index\(\)](#), and [word_index\(\)](#).

Here is the call graph for this function:



16.33.3.4 atomic_set_bit()

```

void cxx::Bitmap_base::atomic_set_bit (
    long bit) [inline], [noexcept]

```

Set bit [bit](#) atomically.

Use this function for multi-threaded access to the bitmap.

Parameters

<i>bit</i>	The number of the bit to set.
------------	-------------------------------

Definition at line 298 of file [bitmap](#).

References [_bits](#), [bit\(\)](#), [bit_index\(\)](#), and [word_index\(\)](#).

Here is the call graph for this function:



16.33.3.5 bit() [1/2]

```

Bitmap_base::word_type cxx::Bitmap_base::bit (
    long bit) const [inline], [noexcept]

```

Get the truth value of a bit.

Parameters

<i>bit</i>	The number of the bit to read.
------------	--------------------------------

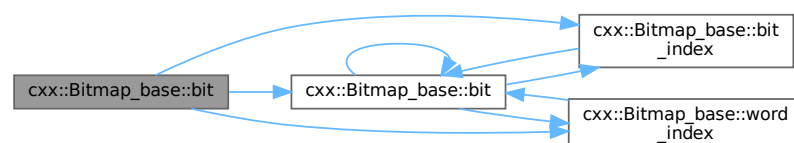
Return values

0	Bit is not set.
!= 0	Bit is set.

Definition at line 318 of file [bitmap](#).

References [_bits](#), [bit\(\)](#), [bit_index\(\)](#), and [word_index\(\)](#).

Here is the call graph for this function:



16.33.3.6 bit() [2/2]

```
void cxx::Bitmap_base::bit (
    long bit,
    bool on) [inline], [noexcept]
```

Set the value of bit `bit` to on.

Parameters

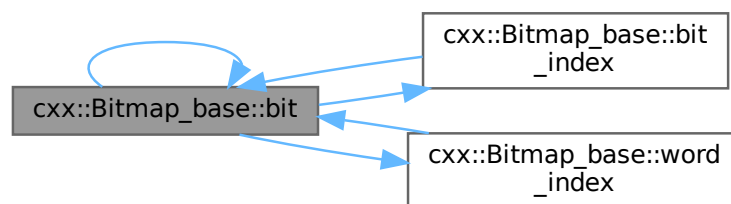
<i>bit</i>	The number of the bit.
<i>on</i>	The boolean value that shall be assigned to the bit.

Definition at line 251 of file [bitmap](#).

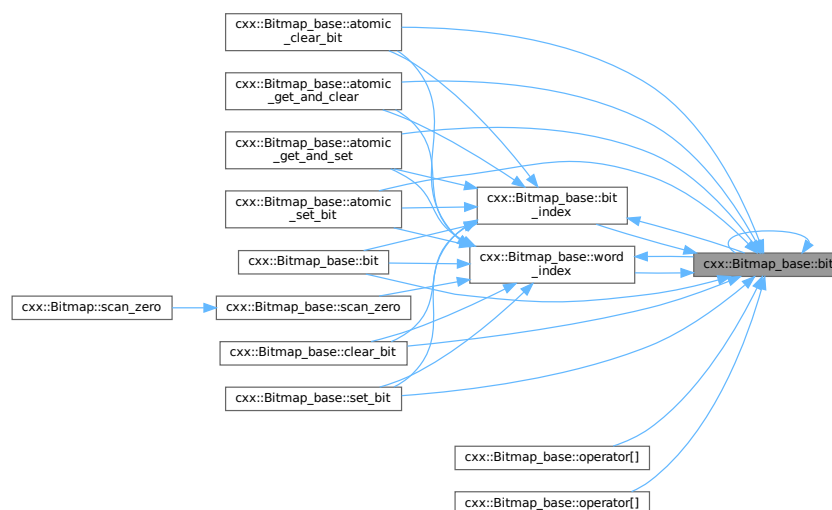
References [_bits](#), [bit\(\)](#), [bit_index\(\)](#), and [word_index\(\)](#).

Referenced by [atomic_clear_bit\(\)](#), [atomic_get_and_clear\(\)](#), [atomic_get_and_set\(\)](#), [atomic_set_bit\(\)](#), [bit\(\)](#), [bit\(\)](#), [bit_index\(\)](#), [clear_bit\(\)](#), [operator\[\]\(\)](#), [operator\[\]\(\)](#), [set_bit\(\)](#), and [word_index\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.33.3.7 bit_index()

```
unsigned cxx::Bitmap_base::bit_index (
    unsigned bit) [inline], [static], [protected]
```

Get the bit index within [word_type](#) for the given bit.

Parameters

<i>bit</i>	The bit index in the bitmap.
------------	------------------------------

Returns

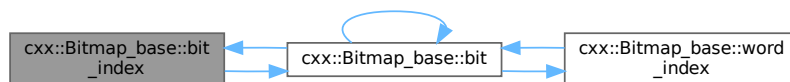
the bit index within [word_type](#) (bit % W_bits).

Definition at line 53 of file [bitmap](#).

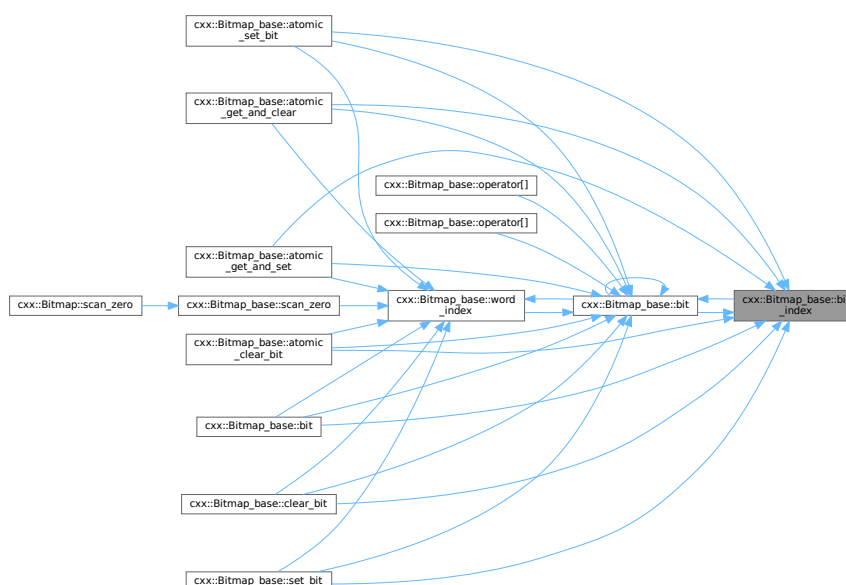
References [bit\(\)](#), and [W_bits](#).

Referenced by [atomic_clear_bit\(\)](#), [atomic_get_and_clear\(\)](#), [atomic_get_and_set\(\)](#), [atomic_set_bit\(\)](#), [bit\(\)](#), [bit\(\)](#), [clear_bit\(\)](#), and [set_bit\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.33.3.8 clear_bit()

```
void cxx::Bitmap_base::clear_bit (
    long bit) [inline], [noexcept]
```

Clear bit [bit](#).

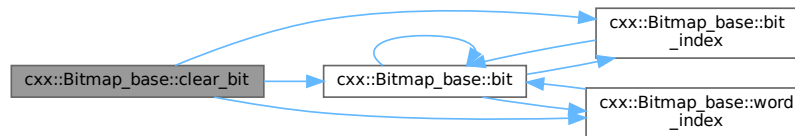
Parameters

<i>bit</i>	The number of the bit to clear.
------------	---------------------------------

Definition at line 260 of file [bitmap](#).

References [_bits](#), [bit\(\)](#), [bit_index\(\)](#), and [word_index\(\)](#).

Here is the call graph for this function:



16.33.3.9 operator[]() [1/2]

```
word_type cxx::Bitmap_base::operator[] (
    long bit) const [inline], [noexcept]
```

Get the bit at index [bit](#).

Parameters

<i>bit</i>	The number of the bit to read.
------------	--------------------------------

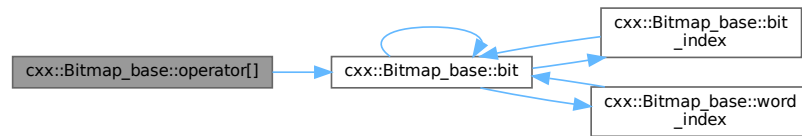
Return values

0	Bit is not set.
!= 0	Bit is set.

Definition at line 181 of file [bitmap](#).

References [bit\(\)](#).

Here is the call graph for this function:



16.33.3.10 operator[]() [2/2]

```

Bit cxx::Bitmap_base::operator[] (
    long bit) [inline], [noexcept]
  
```

Get the lvalue for the bit at index `bit`.

Parameters

<i>bit</i>	The number.
------------	-------------

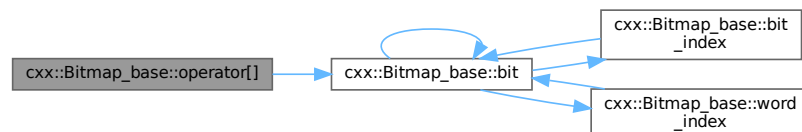
Returns

lvalue for `bit`

Definition at line 191 of file `bitmap`.

References `bit()`.

Here is the call graph for this function:



16.33.3.11 scan_zero()

```

long cxx::Bitmap_base::scan_zero (
    long max_bit,
    long start_bit = 0) const [inline], [noexcept]
  
```

Scan for the first zero bit.

Parameters

<i>max_bit</i>	Upper bound (exclusive) for the scanning operation.
<i>start_bit</i>	Hint at the number of the first bit to look at. Zero bits below <i>start_bit</i> may or may not be taken into account by the implementation.

Return values

\geq	0 Number of first zero bit found.
-1	All bits between <i>start_bit</i> and <i>max_bit</i> are set.

Definition at line 339 of file [bitmap](#).

References [_bits](#), [W_bits](#), and [word_index\(\)](#).

Referenced by [cxx::Bitmap< BITS >::scan_zero\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.33.3.12 set_bit()

```
void cxx::Bitmap_base::set_bit (
    long bit) [inline], [noexcept]
```

Set bit [bit](#).

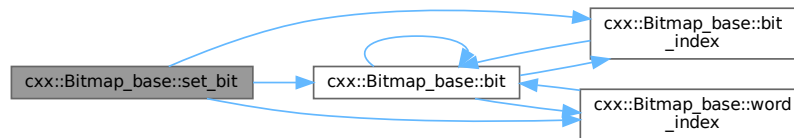
Parameters

<i>bit</i>	The number of the bit to set.
------------	-------------------------------

Definition at line 289 of file [bitmap](#).

References [_bits](#), [bit\(\)](#), [bit_index\(\)](#), and [word_index\(\)](#).

Here is the call graph for this function:



16.33.3.13 word_index()

```

unsigned cxx::Bitmap_base::word_index (
    unsigned bit) [inline], [static], [protected]
  
```

Get the word index for the given bit.

Parameters

<i>bit</i>	The index of the bit in question.
------------	-----------------------------------

Returns

the index in [Bitmap_base::_bits](#) for the given bit (`bit / W_bits`).

Definition at line 44 of file [bitmap](#).

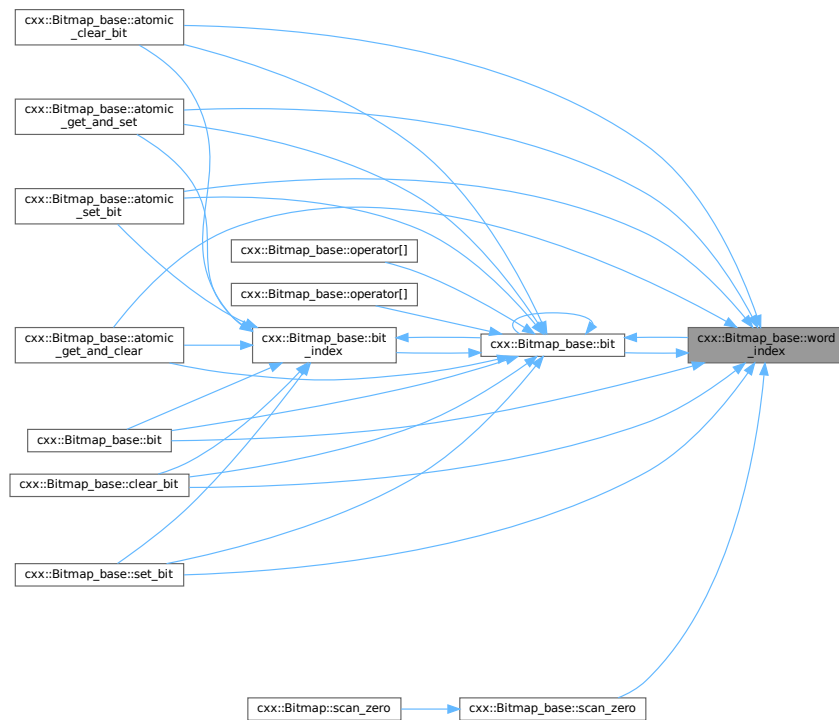
References [bit\(\)](#), and [W_bits](#).

Referenced by [atomic_clear_bit\(\)](#), [atomic_get_and_clear\(\)](#), [atomic_get_and_set\(\)](#), [atomic_set_bit\(\)](#), [bit\(\)](#), [bit\(\)](#), [clear_bit\(\)](#), [scan_zero\(\)](#), and [set_bit\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

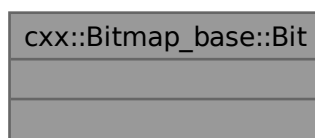
- `I4/cxx/bitmap`

16.34 cxx::Bitmap_base::Bit Class Reference

A writable bit in a bitmap.

```
#include <bitmap>
```

Collaboration diagram for `cxx::Bitmap_base::Bit`:



16.34.1 Detailed Description

A writable bit in a bitmap.

Definition at line 58 of file [bitmap](#).

The documentation for this class was generated from the following file:

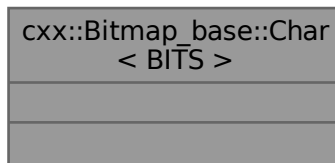
- I4/cxx/bitmap

16.35 cxx::Bitmap_base::Char< BITS > Class Template Reference

Helper abstraction for a byte contained in the bitmap.

```
#include <bitmap>
```

Collaboration diagram for cxx::Bitmap_base::Char< BITS >:



16.35.1 Detailed Description

```
template<long BITS>
class cxx::Bitmap_base::Char< BITS >
```

Helper abstraction for a byte contained in the bitmap.

Definition at line 95 of file [bitmap](#).

The documentation for this class was generated from the following file:

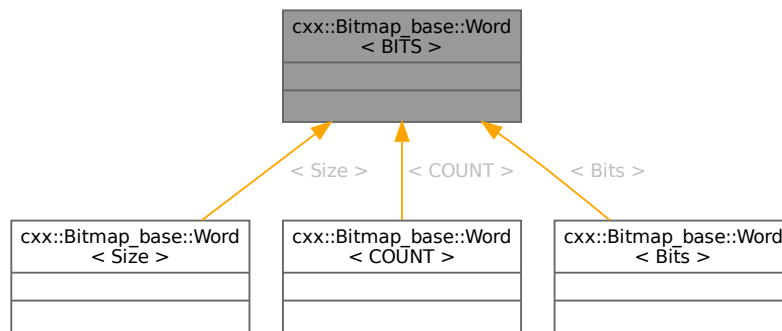
- I4/cxx/bitmap

16.36 cxx::Bitmap_base::Word< BITS > Class Template Reference

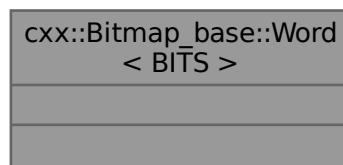
Helper abstraction for a word contained in the bitmap.

```
#include <bitmap>
```

Inheritance diagram for cxx::Bitmap_base::Word< BITS >:



Collaboration diagram for cxx::Bitmap_base::Word< BITS >:



16.36.1 Detailed Description

```
template<long BITS>
class cxx::Bitmap_base::Word< BITS >
```

Helper abstraction for a word contained in the bitmap.

Definition at line 79 of file [bitmap](#).

The documentation for this class was generated from the following file:

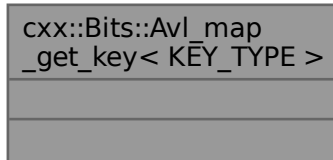
- `I4/cxx/bitmap`

16.37 `cxx::Bits::Avl_map_get_key< KEY_TYPE >` Struct Template Reference

Key-getter for [Avl_map](#).

```
#include <avl_map>
```

Collaboration diagram for `cxx::Bits::Avl_map_get_key< KEY_TYPE >`:



16.37.1 Detailed Description

```
template<typename KEY_TYPE>
struct cxx::Bits::Avl_map_get_key< KEY_TYPE >
```

Key-getter for [Avl_map](#).

Definition at line 25 of file [avl_map](#).

The documentation for this struct was generated from the following file:

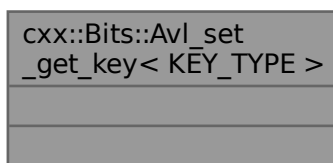
- [l4/cxx/avl_map](#)

16.38 `cxx::Bits::Avl_set_get_key< KEY_TYPE >` Struct Template Reference

Internal, key-getter for [Avl_set](#) nodes.

```
#include <avl_set>
```

Collaboration diagram for `cxx::Bits::Avl_set_get_key< KEY_TYPE >`:



16.38.1 Detailed Description

```
template<typename KEY_TYPE>
struct cxx::Bits::Avl_set_get_key< KEY_TYPE >
```

Internal, key-getter for [Avl_set](#) nodes.

Definition at line 98 of file [avl_set](#).

The documentation for this struct was generated from the following file:

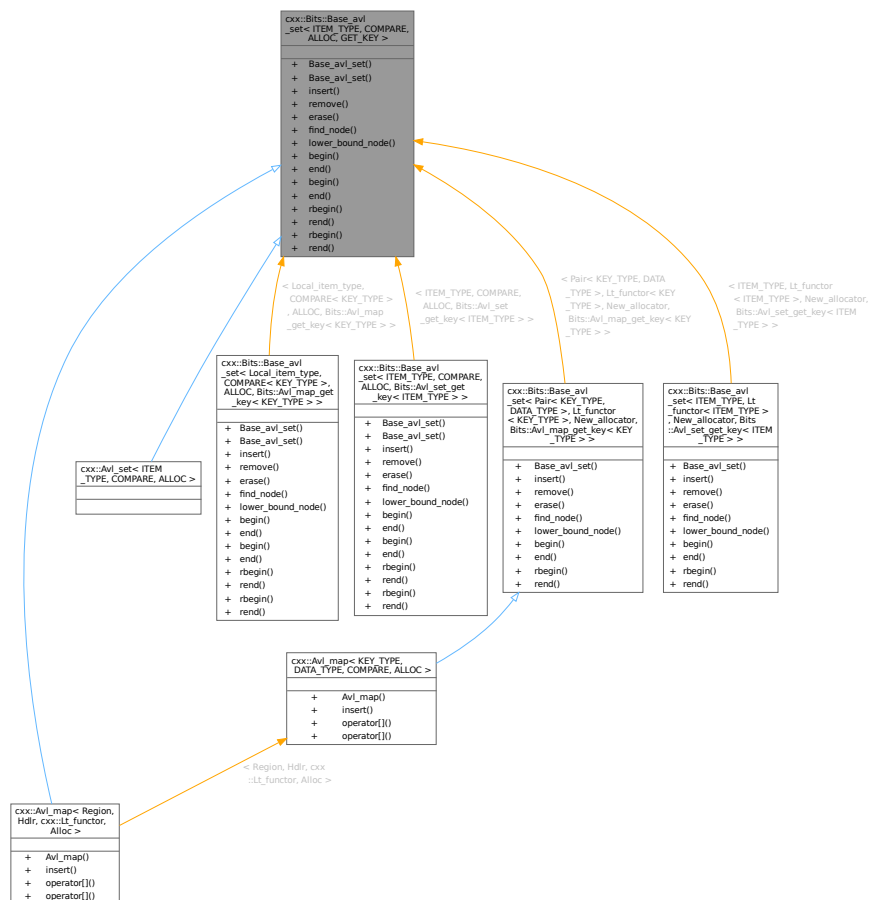
- [l4/cxx/avl_set](#)

16.39 cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY > Class Template Reference

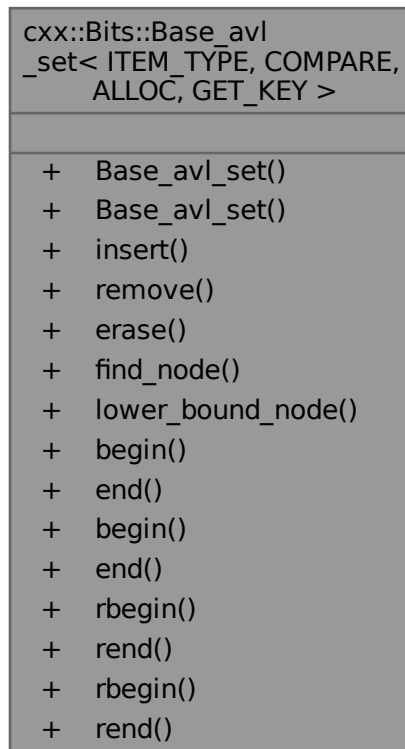
Internal: AVL set with internally managed nodes.

```
#include <avl_set>
```

Inheritance diagram for cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >:



Collaboration diagram for `cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >`:



Data Structures

- class [Node](#)

A smart pointer to a tree item.

Public Types

- enum { [E_noent](#) = 2 , [E_exist](#) = 17 , [E_nomem](#) = 12 , [E_inval](#) = 22 }
- Return status constants.*
- typedef ITEM_TYPE **Item_type**
- Type for the items store in the set.*
- typedef GET_KEY **Get_key**
- Key-getter type to derive the sort key of an internal node.*
- typedef GET_KEY::Key_type **Key_type**
- Type of the sort key used for the items.*
- typedef Type_traits< [Item_type](#) >::Const_type **Const_item_type**
- Type used for const items within the set.*
- typedef COMPARE **Item_compare**
- Type for the comparison functor.*
- typedef ALLOC< _Node > **Node_allocator**

Type for the node allocator.

- typedef Avl_set_iter< _Node, [Item_type](#), Fwd > **Iterator**

Forward iterator for the set.

- typedef Avl_set_iter< _Node, [Const_item_type](#), Fwd > **Const_iterator**

Constant forward iterator for the set.

- typedef Avl_set_iter< _Node, [Item_type](#), Rev > **Rev_iterator**

Backward iterator for the set.

- typedef Avl_set_iter< _Node, [Const_item_type](#), Rev > **Const_rev_iterator**

Constant backward iterator for the set.

Public Member Functions

- [Base_avl_set](#) ([Node_allocator](#) const &alloc=[Node_allocator](#)())

Create a AVL-tree based set.

- [Base_avl_set](#) ([Base_avl_set](#) const &o)

Create a copy of an AVL-tree based set.

- [cxx::Pair](#)< [Iterator](#), int > [insert](#) ([Item_type](#) const &item)

Insert an item into the set.

- int [remove](#) ([Key_type](#) const &item)

Remove an item from the set.

- int [erase](#) ([Key_type](#) const &item)

Erase the item with the given key.

- [Node](#) [find_node](#) ([Key_type](#) const &item) const

*Lookup a node equal to *item*.*

- [Node](#) [lower_bound_node](#) ([Key_type](#) const &key) const

*Find the first node greater or equal to *key*.*

- [Const_iterator](#) [begin](#) () const

Get the constant forward iterator for the first element in the set.

- [Const_iterator](#) [end](#) () const

Get the end marker for the constant forward iterator.

- [Iterator](#) [begin](#) ()

Get the mutable forward iterator for the first element of the set.

- [Iterator](#) [end](#) ()

Get the end marker for the mutable forward iterator.

- [Const_rev_iterator](#) [rbegin](#) () const

Get the constant backward iterator for the last element in the set.

- [Const_rev_iterator](#) [rend](#) () const

Get the end marker for the constant backward iterator.

- [Rev_iterator](#) [rbegin](#) ()

Get the mutable backward iterator for the last element of the set.

- [Rev_iterator](#) [rend](#) ()

Get the end marker for the mutable backward iterator.

16.39.1 Detailed Description

template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY>

class cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >

Internal: AVL set with internally managed nodes.

Use [Avl_set](#), [Avl_map](#), or [Avl_tree](#) in applications.

Template Parameters

<i>ITEM_TYPE</i>	The type of the items to be stored in the set.
<i>COMPARE</i>	The relation to define the partial order, default is to use operator '<'.
<i>ALLOC</i>	The allocator to use for the nodes of the AVL set.
<i>GET_KEY</i>	Sort-key getter (must provide the Key_type and sort-key for an item (of <i>ITEM_TYPE</i>)).

Definition at line [122](#) of file [avl_set](#).

16.39.2 Member Enumeration Documentation

16.39.2.1 anonymous enum

```
template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename
GET_KEY>
anonymous enum
```

Return status constants.

These constants are compatible with the [L4](#) error codes, see [l4_error_code_t](#).

Enumerator

E_noent	Item does not exist.
E_exist	Item exists already.
E_nomem	Memory allocation failed.
E_inval	Internal error.

Definition at line [133](#) of file [avl_set](#).

16.39.3 Constructor & Destructor Documentation

16.39.3.1 Base_avl_set() [1/2]

```
template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename
GET_KEY>
cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Base_avl_set (
    Node_allocator const & alloc = Node_allocator()) [inline], [explicit]
```

Create a AVL-tree based set.

Parameters

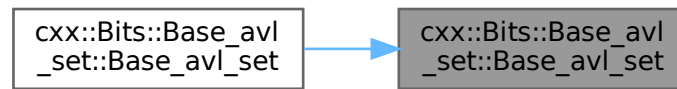
<i>alloc</i>	Node allocator.
--------------	---------------------------------

Create an empty set (AVL-tree based).

Definition at line [243](#) of file [avl_set](#).

Referenced by [Base_avl_set\(\)](#).

Here is the caller graph for this function:



16.39.3.2 Base_avl_set() [2/2]

```

template<typename Item, class Compare, template< typename A > class Alloc, typename KEY_TYPE>
cxx::Bits::Base_avl_set< Item, Compare, Alloc, KEY_TYPE >::Base_avl_set (
    Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY > const & o) [inline]
  
```

Create a copy of an AVL-tree based set.

Parameters

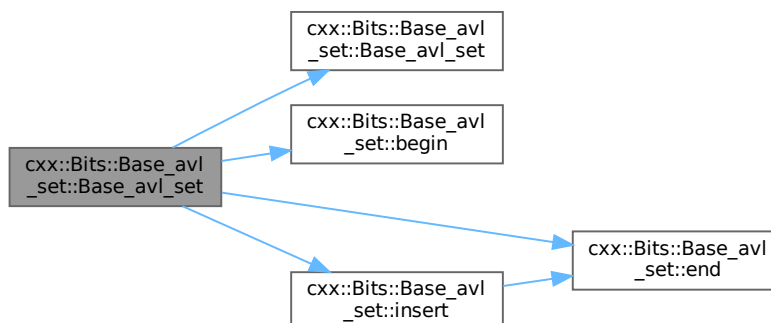
<i>o</i>	The set to copy.
----------	------------------

Creates a deep copy of the set with all its items.

Definition at line 402 of file [avl_set](#).

References [Base_avl_set\(\)](#), [begin\(\)](#), [end\(\)](#), and [insert\(\)](#).

Here is the call graph for this function:



16.39.4 Member Function Documentation

16.39.4.1 `begin()` [1/2]

```
template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename
GET_KEY>
```

```
Iterator cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::begin () [inline]
```

Get the mutable forward iterator for the first element of the set.

Returns

The mutable forward iterator for the first element of the set.

Definition at line 356 of file [avl_set](#).

16.39.4.2 `begin()` [2/2]

```
template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename
GET_KEY>
```

```
Const_iterator cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::begin () const
[inline]
```

Get the constant forward iterator for the first element in the set.

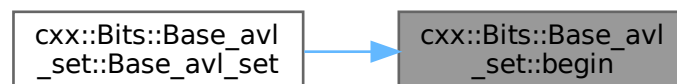
Returns

Constant forward iterator for the first element in the set.

Definition at line 345 of file [avl_set](#).

Referenced by [Base_avl_set\(\)](#).

Here is the caller graph for this function:



16.39.4.3 end() [1/2]

```
template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename
GET_KEY>
Iterator cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::end () [inline]
```

Get the end marker for the mutable forward iterator.

Returns

The end marker for mutable forward iterator.

Definition at line 361 of file [avl_set](#).

16.39.4.4 end() [2/2]

```
template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename
GET_KEY>
Const_iterator cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::end () const
[inline]
```

Get the end marker for the constant forward iterator.

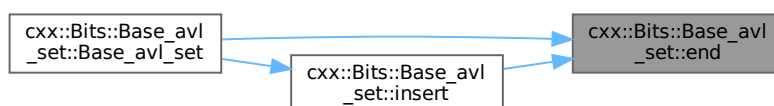
Returns

The end marker for the constant forward iterator.

Definition at line 350 of file [avl_set](#).

Referenced by [Base_avl_set\(\)](#), and [insert\(\)](#).

Here is the caller graph for this function:

**16.39.4.5 erase()**

```
template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename
GET_KEY>
int cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::erase (
    Key_type const & item) [inline]
```

Erase the item with the given key.

Parameters

<i>item</i>	The key of the item to remove.
-------------	--------------------------------

Definition at line 313 of file [avl_set](#).

16.39.4.6 find_node()

```
template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename
GET_KEY>
Node cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::find_node (
    Key_type const & item) const [inline]
```

Lookup a node equal to *item*.

Parameters

<i>item</i>	The value to search for.
-------------	--------------------------

Returns

A smart pointer to the element found. If no element was found the smart pointer will be invalid.

Definition at line 324 of file [avl_set](#).

16.39.4.7 insert()

```
template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename
GET_KEY>
Pair< typename Base_avl_set< Item, Compare, Alloc, KEY_TYPE >::Iterator, int > cxx::Bits::Base_avl_set<
Item, Compare, Alloc, KEY_TYPE >::insert (
    Item_type const & item)
```

Insert an item into the set.

Parameters

<i>item</i>	The item to insert.
-------------	---------------------

Returns

A pair of iterator (*first*) and return value (*second*). *second* will be 0 if the element was inserted into the set and `-#E_exist` if the element was already in the set and the set was therefore not updated. In both cases, *first* contains an iterator that points to the element. *second* may also be `-#E_nomem` when memory for the node could not be allocated. *first* is then invalid.

Insert a new item into the set, each item can only be once in the set.

Definition at line 412 of file [avl_set](#).

References [E_exist](#), [E_nomem](#), [end\(\)](#), [cxx::Pair< First, Second >::first](#), and [cxx::Pair< First, Second >::second](#).

Referenced by [Base_avl_set\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.39.4.8 lower_bound_node()

```
template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY>
```

```
Node cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::lower_bound_node (
    Key_type const & key) const [inline]
```

Find the first node greater or equal to `key`.

Parameters

<i>key</i>	Minimum key to look for.
------------	--------------------------

Returns

Smart pointer to the first node greater or equal to `key`. Will be invalid if no such element was found.

Definition at line 335 of file [avl_set](#).

16.39.4.9 rbegin() [1/2]

```
template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename
GET_KEY>
Rev_iterator cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::rbegin () [inline]
```

Get the mutable backward iterator for the last element of the set.

Returns

The mutable backward iterator for the last element of the set.

Definition at line 378 of file [avl_set](#).

16.39.4.10 rbegin() [2/2]

```
template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename
GET_KEY>
Const_rev_iterator cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::rbegin ()
const [inline]
```

Get the constant backward iterator for the last element in the set.

Returns

The constant backward iterator for the last element in the set.

Definition at line 367 of file [avl_set](#).

16.39.4.11 remove()

```
template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename
GET_KEY>
int cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::remove (
    Key_type const & item) [inline]
```

Remove an item from the set.

Parameters

<i>item</i>	The item to remove.
-------------	---------------------

Return values

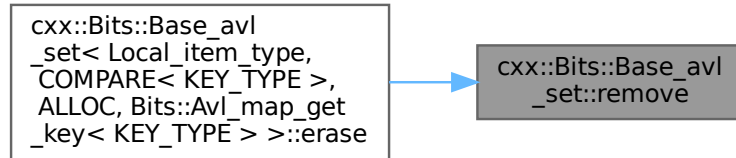
0	Success
-E_noent	Item does not exist

Definition at line 295 of file [avl_set](#).

16.39 cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY > Class Template Reference001

Referenced by [cxx::Bits::Base_avl_set< Local_item_type, COMPARE< KEY_TYPE >, ALLOC, Bits::Avl_map_get_key< KEY_TYP](#)

Here is the caller graph for this function:



16.39.4.12 `rend()` [1/2]

```
template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY>
Rev_iterator cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::rend () [inline]
```

Get the end marker for the mutable backward iterator.

Returns

The end marker for mutable backward iterator.

Definition at line 383 of file [avl_set](#).

16.39.4.13 `rend()` [2/2]

```
template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY>
Const_rev_iterator cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::rend ()
const [inline]
```

Get the end marker for the constant backward iterator.

Returns

The end marker for the constant backward iterator.

Definition at line 372 of file [avl_set](#).

The documentation for this class was generated from the following file:

- [I4/cxx/avl_set](#)

16.40 cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Node Class Reference

A smart pointer to a tree item.

```
#include <avl_set>
```

Collaboration diagram for cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Node:

cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Node	
+	Node()
+	operator*()
+	operator->()
+	valid()
+	operator Item_type const *()

Public Member Functions

- **Node ()**
Default construction for NIL pointer.
- **Item_type const & operator* ()**
Dereference the pointer.
- **Item_type const * operator-> ()**
Dereferenced member access.
- **bool valid () const**
Validity check.
- **operator Item_type const * ()**
Cast to a real item pointer.

16.40.1 Detailed Description

```
template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY>
class cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Node
```

A smart pointer to a tree item.

Definition at line 172 of file [avl_set](#).

16.40.2 Member Function Documentation

16.40.2.1 `operator*()`

```
template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename
GET_KEY>
Item_type const & cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Node::operator*
() [inline]
```

Dereference the pointer.

Precondition

`Node` is valid.

Definition at line 189 of file `avl_set`.

16.40.2.2 `operator->()`

```
template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename
GET_KEY>
Item_type const * cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Node::operator->
() [inline]
```

Dereferenced member access.

Precondition

`Node` is valid.

Definition at line 195 of file `avl_set`.

16.40.2.3 `valid()`

```
template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename
GET_KEY>
bool cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Node::valid () const
[inline]
```

Validity check.

Returns

false if the pointer is NIL, true if valid.

Definition at line 201 of file `avl_set`.

The documentation for this class was generated from the following file:

- `I4/cxx/avl_set`

Public Member Functions

- `bool empty ()` `const`
Check if the list is empty.
- `Value_type front ()` `const`
Return the first element in the list.
- `void clear ()`
Remove all elements from the list.
- `Iterator begin ()`
Return an iterator to the beginning of the list.
- `Const_iterator begin ()` `const`
Return a const iterator to the beginning of the list.
- `Const_iterator end ()` `const`
Return a const iterator to the end of the list.
- `Iterator end ()`
Return an iterator to the end of the list.

Static Public Member Functions

- `static Const_iterator iter (Const_value_type c)`
Return a const iterator that begins at the given element.

Protected Attributes

- `POLICY::Head_type _f`
Pointer to front of the list.

16.41.1 Detailed Description

```
template<typename POLICY>
class cxx::Bits::Basic_list< POLICY >
```

Internal: Common functions for all head-based list implementations.

Definition at line 39 of file [list_basics.h](#).

16.41.2 Member Function Documentation

16.41.2.1 `clear()`

```
template<typename POLICY>
void cxx::Bits::Basic_list< POLICY >::clear () [inline]
```

Remove all elements from the list.

After the operation the state of the elements is undefined.

Definition at line 135 of file [list_basics.h](#).

References [_f](#).

16.41.2.2 iter()

```
template<typename POLICY>
Const_iterator cxx::Bits::Basic_list< POLICY >::iter (
    Const_value_type c) [inline], [static]
```

Return a const iterator that begins at the given element.

Parameters

<code>c</code>	Element where the iterator should start.
----------------	--

Precondition

The element `c` must already be in a list.

Definition at line 148 of file [list_basics.h](#).

The documentation for this class was generated from the following file:

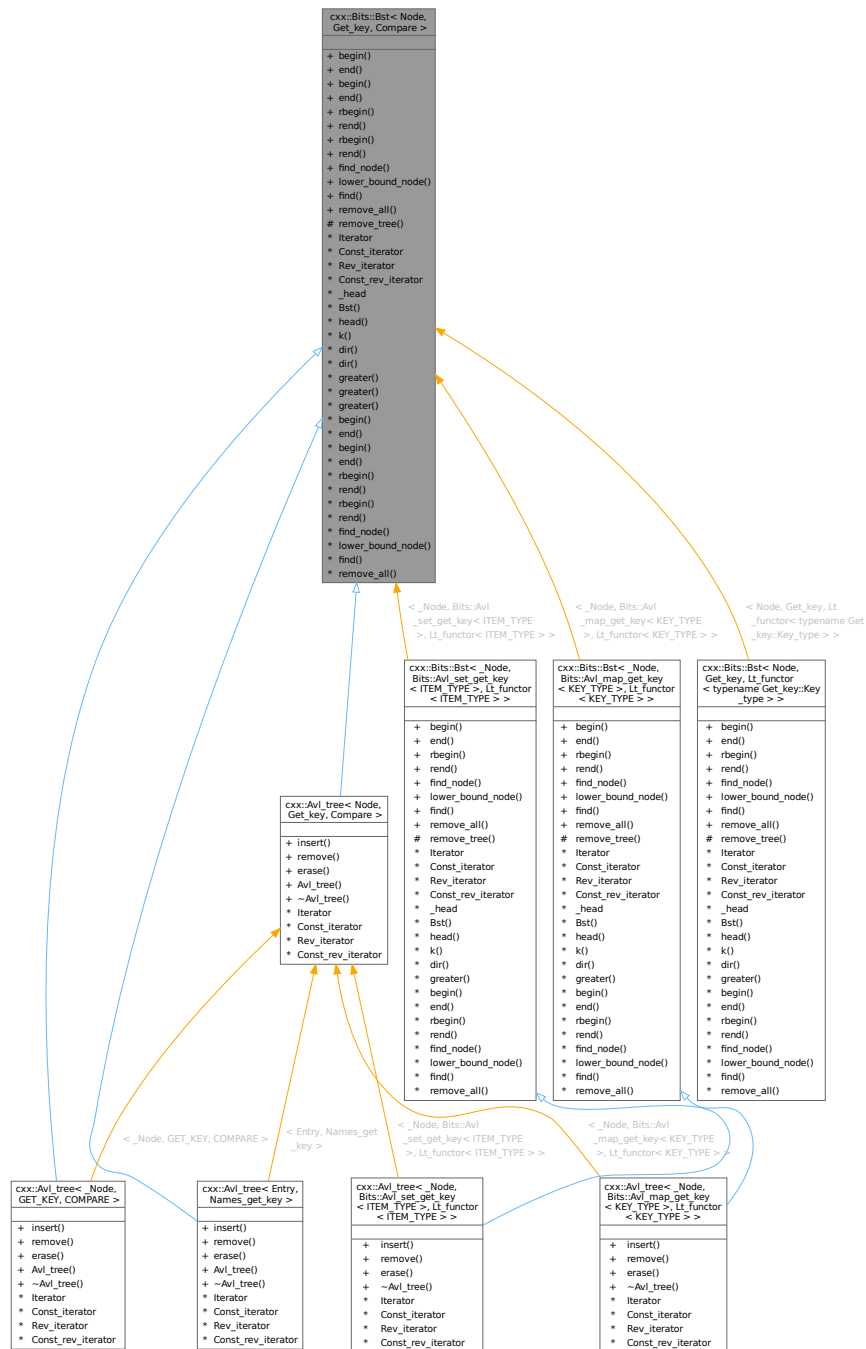
- I4/cxx/bits/list_basics.h

16.42 cxx::Bits::Bst< Node, Get_key, Compare > Class Template Reference

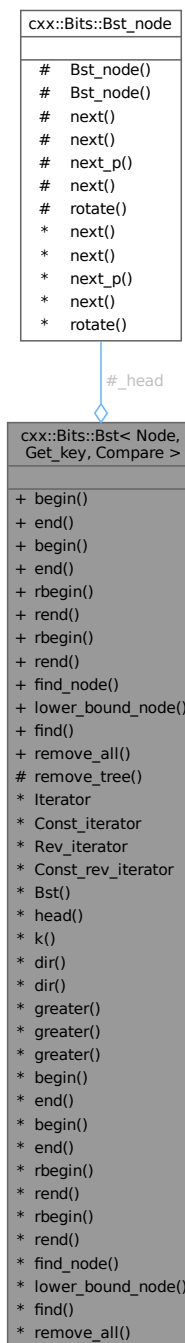
Basic binary search tree (BST).

```
#include <bst.h>
```

Inheritance diagram for `cx::Bits::Bst< Node, Get_key, Compare >`:



Collaboration diagram for cxx::Bits::Bst< Node, Get_key, Compare >:



Public Types

- typedef Get_key::Key_type **Key_type**
The type of key values used to generate the total order of the elements.
- typedef Type_traits< Key_type >::Param_type **Key_param_type**
The type for key parameters.
- typedef Fwd **Fwd_iter_ops**

Helper for building forward iterators for different wrapper classes.

- typedef Rev **Rev_iter_ops**

Helper for building reverse iterators for different wrapper classes.

Iterators

- typedef __Bst_iter< Node, Node, Fwd > **Iterator**
Forward iterator.
- typedef __Bst_iter< Node, Node const, Fwd > **Const_iterator**
Constant forward iterator.
- typedef __Bst_iter< Node, Node, Rev > **Rev_iterator**
Backward iterator.
- typedef __Bst_iter< Node, Node const, Rev > **Const_rev_iterator**
Constant backward.

Public Member Functions

Get default iterators for the ordered tree.

- [Const_iterator begin](#) () const
Get the constant forward iterator for the first element in the set.
- [Const_iterator end](#) () const
Get the end marker for the constant forward iterator.
- [Iterator begin](#) ()
Get the mutable forward iterator for the first element of the set.
- [Iterator end](#) ()
Get the end marker for the mutable forward iterator.
- [Const_rev_iterator rbegin](#) () const
Get the constant backward iterator for the last element in the set.
- [Const_rev_iterator rend](#) () const
Get the end marker for the constant backward iterator.
- [Rev_iterator rbegin](#) ()
Get the mutable backward iterator for the last element of the set.
- [Rev_iterator rend](#) ()
Get the end marker for the mutable backward iterator.

Lookup functions.

- Node * [find_node](#) (Key_param_type key) const
find the node with the given key.
- Node * [lower_bound_node](#) (Key_param_type key) const
Find the first node with a key not less than the given key.
- [Const_iterator find](#) (Key_param_type key) const
find the node with the given key.
- template<typename FUNC>
void [remove_all](#) (FUNC &&callback)
Clear the tree.

Static Protected Member Functions

- template<typename FUNC>
static void [remove_tree](#) (Bst_node *head, FUNC &&callback)
Remove all elements in the subtree of head.

Interior access for descendants.

As this class is an intended base class we provide protected access to our interior, use 'using' to make this private in concrete implementations.

- [Bst_node](#) * **_head**
The head pointer of the tree.
- **Bst** ()
Create an empty tree.
- Node * **head** () const
Access the head node as object of type Node.
- static [Key_type](#) **k** ([Bst_node](#) const *n)
Get the key value of n.
- static [Dir](#) **dir** ([Key_param_type](#) l, [Key_param_type](#) r)
Get the direction to go from l to search for r.
- static [Dir](#) **dir** ([Key_param_type](#) l, [Bst_node](#) const *r)
Get the direction to go from l to search for r.
- static bool **greater** ([Key_param_type](#) l, [Key_param_type](#) r)
Is l greater than r.
- static bool **greater** ([Key_param_type](#) l, [Bst_node](#) const *r)
Is l greater than r.
- static bool **greater** ([Bst_node](#) const *l, [Bst_node](#) const *r)
Is l greater than r.

16.42.1 Detailed Description

```
template<typename Node, typename Get_key, typename Compare>
class cxx::Bits::Bst< Node, Get_key, Compare >
```

Basic binary search tree (BST).

This class is intended as a base class for concrete binary search trees, such as an AVL tree. This class already provides the basic lookup methods and iterator definitions for a BST.

Definition at line 31 of file [bst.h](#).

16.42.2 Member Function Documentation**16.42.2.1 begin() [1/2]**

```
template<typename Node, typename Get_key, typename Compare>
Iterator cxx::Bits::Bst< Node, Get_key, Compare >::begin () [inline]
```

Get the mutable forward iterator for the first element of the set.

Returns

The mutable forward iterator for the first element of the set.

Definition at line 183 of file [bst.h](#).

References [head\(\)](#).

Here is the call graph for this function:

**16.42.2.2 begin() [2/2]**

```
template<typename Node, typename Get_key, typename Compare>
Const_iterator cxx::Bits::Bst< Node, Get_key, Compare >::begin () const [inline]
```

Get the constant forward iterator for the first element in the set.

Returns

Constant forward iterator for the first element in the set.

Definition at line 172 of file [bst.h](#).

References [head\(\)](#).

Here is the call graph for this function:

**16.42.2.3 dir() [1/2]**

```
template<typename Node, typename Get_key, typename Compare>
Dir cxx::Bits::Bst< Node, Get_key, Compare >::dir (
    Key_param_type l,
    Bst_node const * r) [inline], [static], [protected]
```

Get the direction to go from `l` to search for `r`.

Parameters

<i>l</i>	is the key to look for.
<i>r</i>	is the node at the current position.

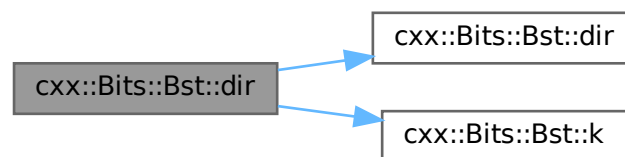
Return values

<i>Direction::L</i>	For left.
<i>Direction::R</i>	For right.
<i>Direction::N</i>	If <i>l</i> is equal to <i>r</i> .

Definition at line 128 of file [bst.h](#).

References [dir\(\)](#), and [k\(\)](#).

Here is the call graph for this function:

**16.42.2.4 dir() [2/2]**

```

template<typename Node, typename Get_key, typename Compare>
Dir cxx::Bits::Bst< Node, Get_key, Compare >::dir (
    Key_param_type l,
    Key_param_type r) [inline], [static], [protected]
  
```

Get the direction to go from *l* to search for *r*.

Parameters

<i>l</i>	is the key to look for.
<i>r</i>	is the key at the current position.

Return values

<i>Direction::L</i>	for left
<i>Direction::R</i>	for right

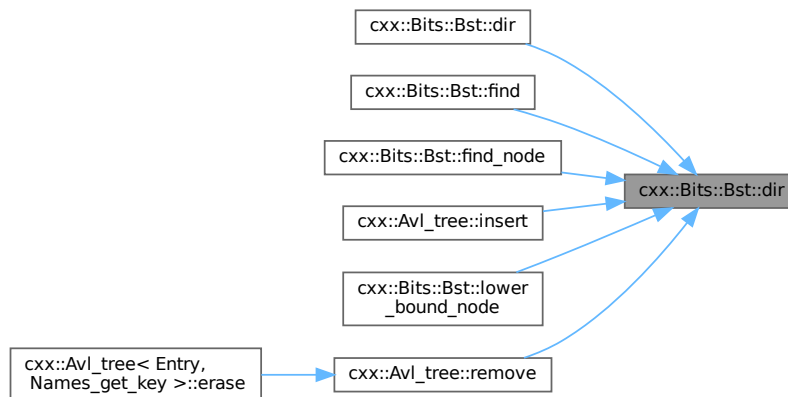
<i>Direction::N</i>	if <i>l</i> is equal to <i>r</i> .
---------------------	------------------------------------

Definition at line 111 of file [bst.h](#).

References [cxx::Bits::Direction::L](#), and [cxx::Bits::Direction::N](#).

Referenced by [dir\(\)](#), [find\(\)](#), [find_node\(\)](#), [cxx::Avl_tree< Node, Get_key, Compare >::insert\(\)](#), [lower_bound_node\(\)](#), and [cxx::Avl_tree< Node, Get_key, Compare >::remove\(\)](#).

Here is the caller graph for this function:



16.42.2.5 `end()` [1/2]

```
template<typename Node, typename Get_key, typename Compare>
Iterator cxx::Bits::Bst< Node, Get_key, Compare >::end () [inline]
```

Get the end marker for the mutable forward iterator.

Returns

The end marker for mutable forward iterator.

Definition at line 188 of file [bst.h](#).

16.42.2.6 `end()` [2/2]

```
template<typename Node, typename Get_key, typename Compare>
Const_iterator cxx::Bits::Bst< Node, Get_key, Compare >::end () const [inline]
```

Get the end marker for the constant forward iterator.

Returns

The end marker for the constant forward iterator.

Definition at line 177 of file [bst.h](#).

16.42.2.7 find()

```
template<typename Node, typename Get_key, class Compare>
Bst< Node, Get_key, Compare >::Const_iterator cxx::Bits::Bst< Node, Get_key, Compare >::find
(
    Key_param_type key) const [inline]
```

find the node with the given *key*.

Parameters

<i>key</i>	The key value of the element to search.
------------	---

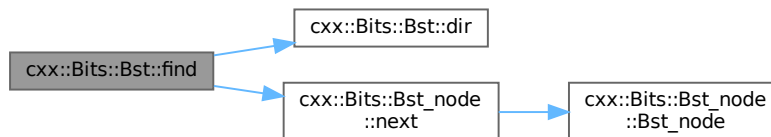
Returns

A valid iterator for the node with the given *key*, or an invalid iterator if *key* was not found.

Definition at line 305 of file [bst.h](#).

References [_head](#), [dir\(\)](#), [cxx::Bits::Direction::L](#), [cxx::Bits::Direction::N](#), and [cxx::Bits::Bst_node::next\(\)](#).

Here is the call graph for this function:



16.42.2.8 find_node()

```
template<typename Node, typename Get_key, class Compare>
Node * cxx::Bits::Bst< Node, Get_key, Compare >::find_node (
    Key_param_type key) const [inline]
```

find the node with the given *key*.

Parameters

<i>key</i>	The key value of the element to search.
------------	---

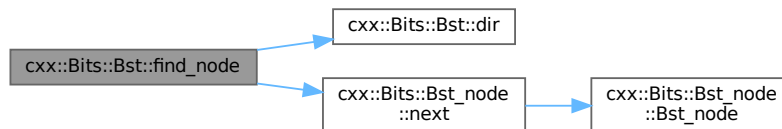
Returns

A pointer to the node with the given *key*, or `NULL` if *key* was not found.

Definition at line 269 of file [bst.h](#).

References [_head](#), [dir\(\)](#), [cxx::Bits::Direction::N](#), and [cxx::Bits::Bst_node::next\(\)](#).

Here is the call graph for this function:

**16.42.2.9 lower_bound_node()**

```
template<typename Node, typename Get_key, class Compare>
Node * cxx::Bits::Bst< Node, Get_key, Compare >::lower_bound_node (
    Key_param_type key) const [inline]
```

Find the first node with a key not less than the given key.

Parameters

<i>key</i>	The key used for the search.
------------	------------------------------

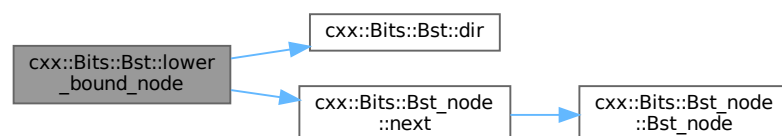
Returns

A pointer to the found node, or `NULL` if no node was found.

Definition at line 285 of file [bst.h](#).

References [_head](#), [dir\(\)](#), [cxx::Bits::Direction::L](#), [cxx::Bits::Direction::N](#), and [cxx::Bits::Bst_node::next\(\)](#).

Here is the call graph for this function:



16.42.2.10 rbegin() [1/2]

```
template<typename Node, typename Get_key, typename Compare>
Rev_iterator cxx::Bits::Bst< Node, Get_key, Compare >::rbegin () [inline]
```

Get the mutable backward iterator for the last element of the set.

Returns

The mutable backward iterator for the last element of the set.

Definition at line 205 of file [bst.h](#).

References [head\(\)](#).

Here is the call graph for this function:

**16.42.2.11 rbegin()** [2/2]

```
template<typename Node, typename Get_key, typename Compare>
Const_rev_iterator cxx::Bits::Bst< Node, Get_key, Compare >::rbegin () const [inline]
```

Get the constant backward iterator for the last element in the set.

Returns

The constant backward iterator for the last element in the set.

Definition at line 194 of file [bst.h](#).

References [head\(\)](#).

Here is the call graph for this function:



16.42.2.12 remove_all()

```
template<typename Node, typename Get_key, typename Compare>
template<typename FUNC>
void cxx::Bits::Bst< Node, Get_key, Compare >::remove_all (
    FUNC && callback) [inline]
```

Clear the tree.

Parameters

<i>callback</i>	Optional function to be called on each removed element.
-----------------	---

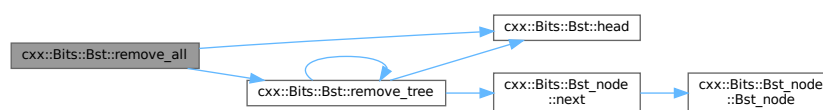
The callback may delete the elements. The function guarantees that the elements are no longer used after the callback has been called.

Definition at line 251 of file [bst.h](#).

References [_head](#), [head\(\)](#), and [remove_tree\(\)](#).

Referenced by [cxx::Avl_tree< Entry, Names_get_key >::~~Avl_tree\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.42.2.13 remove_tree()

```

template<typename Node, typename Get_key, typename Compare>
template<typename FUNC>
void cxx::Bits::Bst< Node, Get_key, Compare >::remove_tree (
    Bst_node * head,
    FUNC && callback) [inline], [static], [protected]

```

Remove all elements in the subtree of head.

Parameters

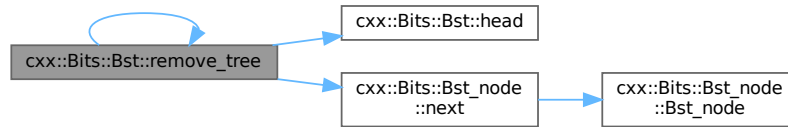
<i>head</i>	Head of the the subtree to remove
<i>callback</i>	Optional function called on each removed element.

Definition at line 151 of file [bst.h](#).

References [head\(\)](#), [cxx::Bits::Direction::L](#), [cxx::Bits::Bst_node::next\(\)](#), [cxx::Bits::Direction::R](#), and [remove_tree\(\)](#).

Referenced by [remove_all\(\)](#), and [remove_tree\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.42.2.14 `rend()` [1/2]

```
template<typename Node, typename Get_key, typename Compare>
Rev_iterator cxx::Bits::Bst< Node, Get_key, Compare >::rend () [inline]
```

Get the end marker for the mutable backward iterator.

Returns

The end marker for mutable backward iterator.

Definition at line 210 of file [bst.h](#).

16.42.2.15 `rend()` [2/2]

```
template<typename Node, typename Get_key, typename Compare>
Const_rev_iterator cxx::Bits::Bst< Node, Get_key, Compare >::rend () const [inline]
```

Get the end marker for the constant backward iterator.

Returns

The end marker for the constant backward iterator.

Definition at line 199 of file [bst.h](#).

The documentation for this class was generated from the following file:

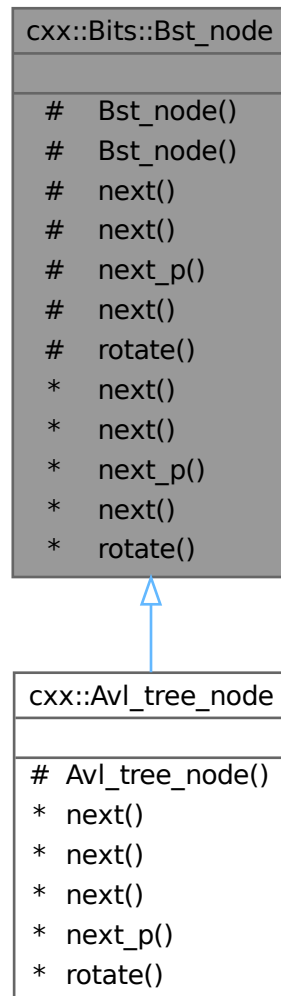
- [I4/cxx/bits/bst.h](#)

16.43 cxx::Bits::Bst_node Class Reference

Basic type of a node in a binary search tree (BST).

```
#include <bst_base.h>
```

Inheritance diagram for cxx::Bits::Bst_node:



Collaboration diagram for `cxx::Bits::Bst_node`:

cxx::Bits::Bst_node	
#	Bst_node()
#	Bst_node()
#	next()
#	next()
#	next_p()
#	next()
#	rotate()
*	next()
*	next()
*	next_p()
*	next()
*	rotate()

Protected Member Functions

- **Bst_node** ()
Create uninitialized node.
- **Bst_node** (bool)
Create initialized node.

Static Protected Member Functions

Access to BST linkage.

Provide access to the tree linkage to inherited classes. Inherited nodes, such as AVL nodes should make these methods private via 'using'

- static `Bst_node` * **next** (`Bst_node` const *p, `Direction` d)
Get next node in direction d.
- static void **next** (`Bst_node` *p, `Direction` d, `Bst_node` *n)
Set next node of p in direction d to n.
- static `Bst_node` ** **next_p** (`Bst_node` *p, `Direction` d)
Get pointer to link in direction d.
- template<typename Node>
static Node * **next** (`Bst_node` const *p, `Direction` d)
Get next node in direction d as type Node.
- static void **rotate** (`Bst_node` **t, `Direction` idir)
Rotate subtree t in the opposite direction of idir.

16.43.1 Detailed Description

Basic type of a node in a binary search tree (BST).

Definition at line 70 of file [bst_base.h](#).

The documentation for this class was generated from the following file:

- [l4/cxx/bits/bst_base.h](#)

16.44 cxx::Bits::Direction Struct Reference

The direction to go in a binary search tree.

```
#include <bst_base.h>
```

Collaboration diagram for cxx::Bits::Direction:

cxx::Bits::Direction	
+	Direction()
+	Direction()
+	Direction()
+	operator!()
+	operator==(())
+	operator!=(())
+	operator==(())
+	operator!=(())
*	operator==(())
*	operator!=(())
*	operator==(())
*	operator!=(())

Public Types

- enum [Direction_e](#) { [L](#) = 0 , [R](#) = 1 , [N](#) = 2 }

The literal direction values.

Public Member Functions

- **Direction** ()=default
Uninitialized direction.
- **Direction** ([Direction_e](#) d)
Convert a literal direction ([L](#), [R](#), [N](#)) to an object.
- **Direction** (bool b)
Convert a boolean to a direction (false == [L](#), true == [R](#)).
- **Direction operator!** () const
Negate the direction.

Comparison operators (equality and inequality)

- bool **operator==** ([Direction_e](#) o) const
Compare for equality.
- bool **operator!=** ([Direction_e](#) o) const
Compare for inequality.
- bool **operator==** ([Direction](#) o) const
Compare for equality.
- bool **operator!=** ([Direction](#) o) const
Compare for inequality.

16.44.1 Detailed Description

The direction to go in a binary search tree.

Definition at line 28 of file [bst_base.h](#).

16.44.2 Member Enumeration Documentation

16.44.2.1 Direction_e

```
enum cxx::Bits::Direction::Direction\_e
```

The literal direction values.

Enumerator

L	Go to the left child.
R	Go to the right child.
N	Stop.

Definition at line 31 of file [bst_base.h](#).

16.44.3 Member Function Documentation

16.44.3.1 `operator"!()`

```
Direction cxx::Bits::Direction::operator! () const [inline]
```

Negate the direction.

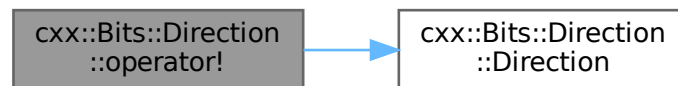
Note

This is only defined for a current value of `L` or `R`

Definition at line 52 of file `bst_base.h`.

References `Direction()`.

Here is the call graph for this function:



The documentation for this struct was generated from the following file:

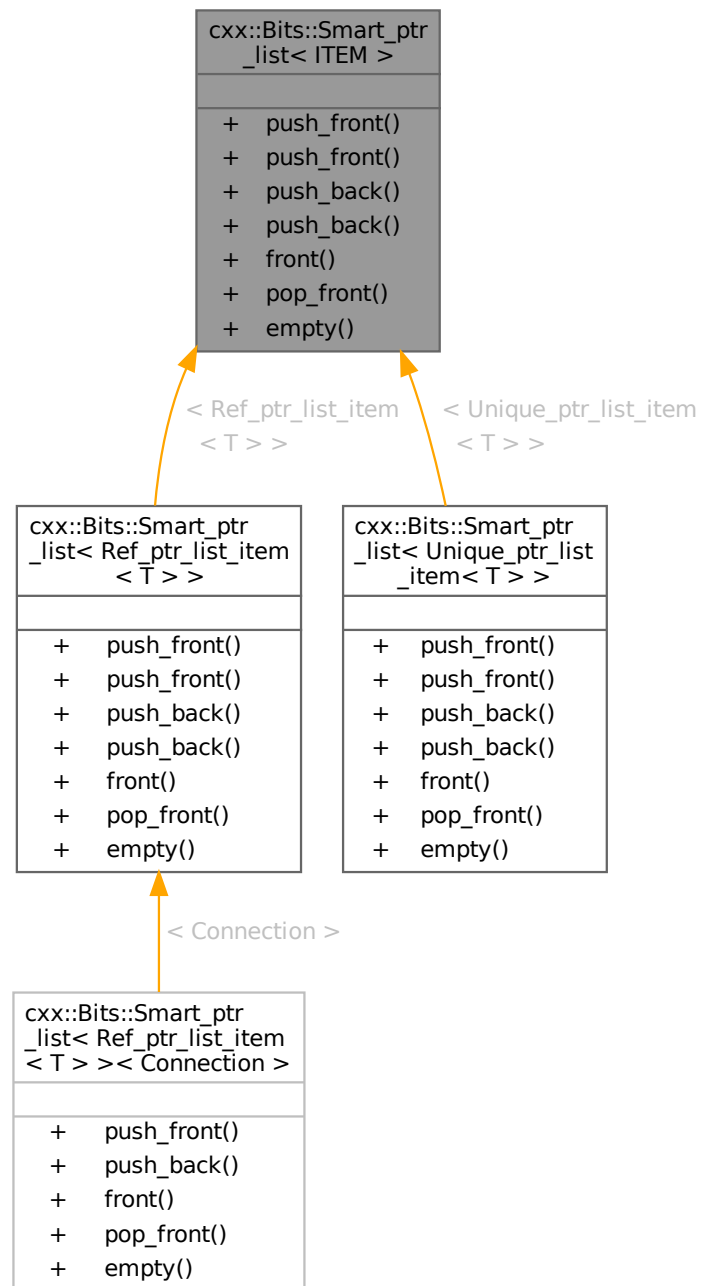
- `I4/cxx/bits/bst_base.h`

16.45 `cxx::Bits::Smart_ptr_list< ITEM >` Class Template Reference

[List](#) of smart-pointer-managed objects.

```
#include <smart_ptr_list.h>
```

Inheritance diagram for `cxx::Bits::Smart_ptr_list< ITEM >`:



Collaboration diagram for cxx::Bits::Smart_ptr_list< ITEM >:

cxx::Bits::Smart_ptr_list< ITEM >	
+	push_front()
+	push_front()
+	push_back()
+	push_back()
+	front()
+	pop_front()
+	empty()

Public Member Functions

- void **push_front** (Next_type &&e)
Add an element to the front of the list.
- void **push_front** (Next_type const &e)
Add an element to the front of the list.
- void **push_back** (Next_type &&e)
Add an element at the end of the list.
- void **push_back** (Next_type const &e)
Add an element at the end of the list.
- Value_type * **front** () const
Return a pointer to the first element in the list.
- Next_type **pop_front** ()
Remove the element in front of the list and return it.
- bool **empty** () const
Check if the list is empty.

16.45.1 Detailed Description

```
template<typename ITEM>
class cxx::Bits::Smart_ptr_list< ITEM >
```

List of smart-pointer-managed objects.

Template Parameters

<i>ITEM</i>	Type of the list items.
-------------	-------------------------

The list is implemented as a single-linked list connected via smart pointers, so that they are automatically cleaned up when they are removed from the list.

Definition at line 46 of file [smart_ptr_list.h](#).

16.45.2 Member Function Documentation

16.45.2.1 pop_front()

```
template<typename ITEM>
Next_type cxx::Bits::Smart_ptr_list< ITEM >::pop_front () [inline]
```

Remove the element in front of the list and return it.

Returns

The element that was previously in front of the list as a managed pointer or a nullptr-equivalent when the list was already empty.

Definition at line 149 of file [smart_ptr_list.h](#).

The documentation for this class was generated from the following file:

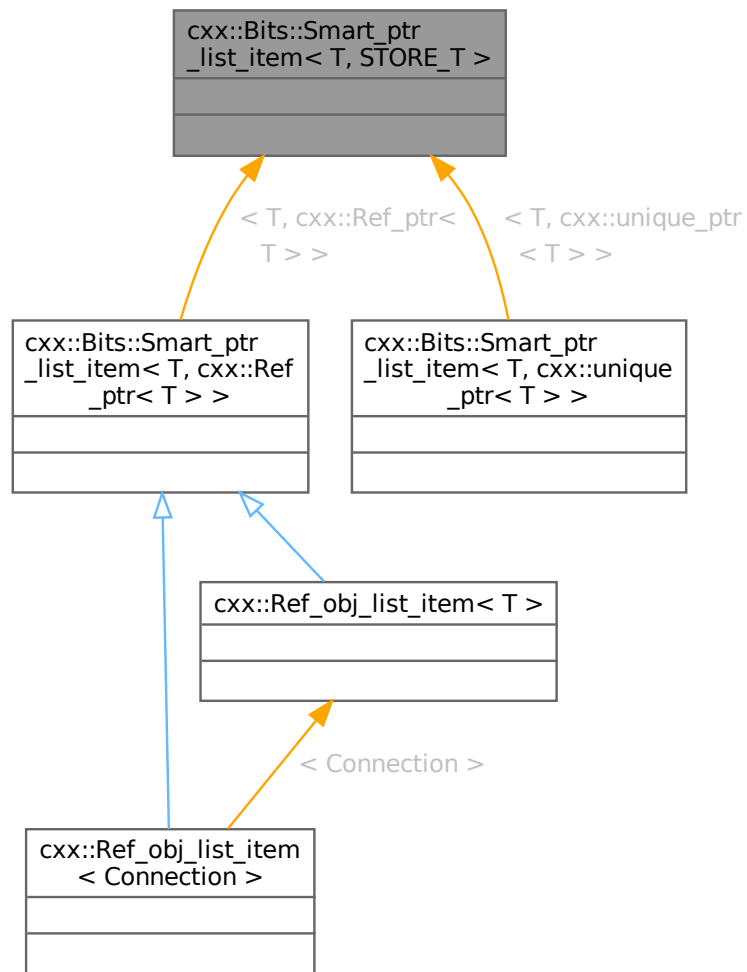
- [l4/cxx/bits/smart_ptr_list.h](#)

16.46 cxx::Bits::Smart_ptr_list_item< T, STORE_T > Class Template Reference

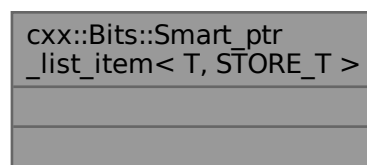
[List](#) item for an arbitrary item in a [Smart_ptr_list](#).

```
#include <smart_ptr_list.h>
```

Inheritance diagram for cxx::Bits::Smart_ptr_list_item< T, STORE_T >:



Collaboration diagram for cxx::Bits::Smart_ptr_list_item< T, STORE_T >:



16.46.1 Detailed Description

```
template<typename T, typename STORE_T>
class cxx::Bits::Smart_ptr_list_item< T, STORE_T >
```

List item for an arbitrary item in a [Smart_ptr_list](#).

Template Parameters

<i>T</i>	Type of object to be stored in the list.
<i>STORE_T</i>	Storage type for pointer to next item. The class must implement a <code>get()</code> function that returns a pointer to the stored object and destroy the stored object when the item goes out of scope.

Definition at line 27 of file [smart_ptr_list.h](#).

The documentation for this class was generated from the following file:

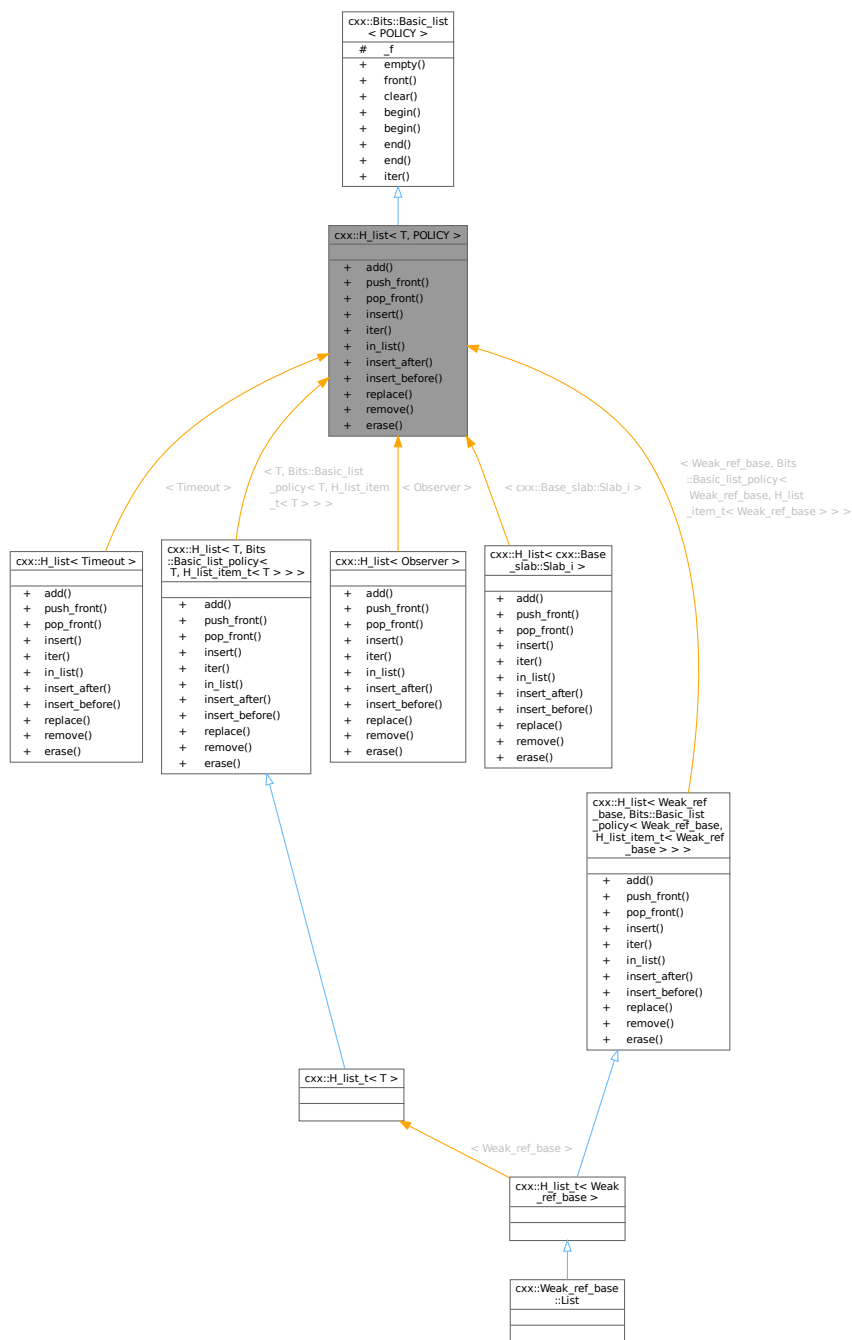
- [I4/cxx/bits/smart_ptr_list.h](#)

16.47 `cxx::H_list< T, POLICY >` Class Template Reference

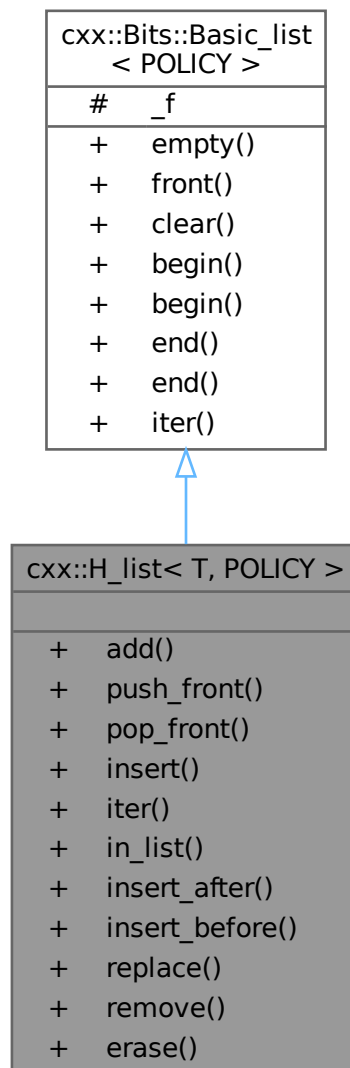
General double-linked list of unspecified [cxx::H_list_item](#) elements.

```
#include <hlist>
```


Inheritance diagram for `cxx::H_list< T, POLICY >`:



Collaboration diagram for `cxx::H_list< T, POLICY >`:



Public Member Functions

- void **add** (T *e)
Add element to the front of the list.
- void **push_front** (T *e)
Add element to the front of the list.
- T * **pop_front** ()
Remove and return the head element of the list.
- Iterator **insert** (T *e, Iterator const &pred)
Insert an element at the iterator position.

Public Member Functions inherited from `cxx::Bits::Basic_list< POLICY >`

- `bool empty () const`
Check if the list is empty.
- `Value_type front () const`
Return the first element in the list.
- `void clear ()`
Remove all elements from the list.
- `Iterator begin ()`
Return an iterator to the beginning of the list.
- `Const_iterator begin () const`
Return a const iterator to the beginning of the list.
- `Const_iterator end () const`
Return a const iterator to the end of the list.
- `Iterator end ()`
Return an iterator to the end of the list.

Static Public Member Functions

- `static Iterator iter (T *c)`
Return an iterator for an arbitrary list element.
- `static bool in_list (T const *e)`
Check if the given element is currently part of a list.
- `static Iterator insert_after (T *e, Iterator const &pred)`
Insert an element after the iterator position.
- `static void insert_before (T *e, Iterator const &succ)`
Insert an element before the iterator position.
- `static void replace (T *p, T *e)`
Replace an element in a list with a new element.
- `static void remove (T *e)`
Remove the given element from its list.
- `static Iterator erase (Iterator const &e)`
Remove the element at the given iterator position.

Static Public Member Functions inherited from `cxx::Bits::Basic_list< POLICY >`

- `static Const_iterator iter (Const_value_type c)`
Return a const iterator that begins at the given element.

Additional Inherited Members**Protected Attributes inherited from `cxx::Bits::Basic_list< POLICY >`**

- `POLICY::Head_type _f`
Pointer to front of the list.

16.47.1 Detailed Description

```
template<typename T, typename POLICY = Bits::Basic_list_policy< T, H_list_item>>
class cxx::H_list< T, POLICY >
```

General double-linked list of unspecified [cxx::H_list_item](#) elements.

Most of the time, you want to use [H_list_t](#).

Definition at line 69 of file [hlist](#).

16.47.2 Member Function Documentation

16.47.2.1 erase()

```
template<typename T, typename POLICY = Bits::Basic_list_policy< T, H_list_item>>
Iterator cxx::H_list< T, POLICY >::erase (
    Iterator const & e) [inline], [static]
```

Remove the element at the given iterator position.

Parameters

<i>e</i>	Iterator pointing to the element to be removed. Must not point to end() .
----------	---

Returns

New iterator pointing to the element after the removed one.

Note

The hlist implementation guarantees that the original iterator is still valid after the element has been removed. In fact, the iterator returned is the same as the one supplied in the *e* parameter.

Definition at line 236 of file [hlist](#).

16.47.2.2 insert()

```
template<typename T, typename POLICY = Bits::Basic_list_policy< T, H_list_item>>
Iterator cxx::H_list< T, POLICY >::insert (
    T * e,
    Iterator const & pred) [inline]
```

Insert an element at the iterator position.

Parameters

<i>e</i>	New Element to be inserted
<i>pred</i>	Iterator pointing to the element after which the element will be inserted. If <code>end()</code> is given, the element will be inserted at the beginning of the queue.

Returns

Iterator pointing to the newly inserted element.

Definition at line 133 of file [hlist](#).

16.47.2.3 `insert_after()`

```
template<typename T, typename POLICY = Bits::Basic_list_policy< T, H_list_item>>
Iterator cxx::H_list< T, POLICY >::insert_after (
    T * e,
    Iterator const & pred) [inline], [static]
```

Insert an element after the iterator position.

Parameters

<i>e</i>	New element to be inserted.
<i>pred</i>	Iterator pointing to the element after which the element will be inserted. Must not be <code>end()</code> .

Returns

Iterator pointing to the newly inserted element.

Precondition

The list must not be empty.

Definition at line 160 of file [hlist](#).

16.47.2.4 `insert_before()`

```
template<typename T, typename POLICY = Bits::Basic_list_policy< T, H_list_item>>
void cxx::H_list< T, POLICY >::insert_before (
    T * e,
    Iterator const & succ) [inline], [static]
```

Insert an element before the iterator position.

Parameters

<i>e</i>	New element to be inserted.
<i>succ</i>	Iterator pointing to the element before which the element will be inserted. Must not be end() .

Definition at line 180 of file [hlist](#).

16.47.2.5 `iter()`

```
template<typename T, typename POLICY = Bits::Basic_list_policy< T, H_list_item>>
Iterator cxx::H\_list< T, POLICY >::iter (
    T * c)    [inline], [static]
```

Return an iterator for an arbitrary list element.

Parameters

<i>c</i>	List element to start the iteration.
----------	--

Returns

A mutable forward iterator.

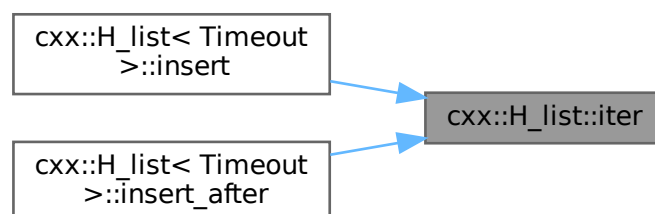
Precondition

The element must be in a list.

Definition at line 93 of file [hlist](#).

Referenced by [cxx::H_list< Timeout >::insert\(\)](#), and [cxx::H_list< Timeout >::insert_after\(\)](#).

Here is the caller graph for this function:



16.47.2.6 `pop_front()`

```
template<typename T, typename POLICY = Bits::Basic_list_policy< T, H_list_item>>
T * cxx::H_list< T, POLICY >::pop_front () [inline]
```

Remove and return the head element of the list.

Precondition

The list must not be empty or the behaviour will be undefined.

Definition at line 116 of file `hlist`.

16.47.2.7 `remove()`

```
template<typename T, typename POLICY = Bits::Basic_list_policy< T, H_list_item>>
void cxx::H_list< T, POLICY >::remove (
    T * e) [inline], [static]
```

Remove the given element from its list.

Parameters

<code>e</code>	Element to be removed. Must be in a list.
----------------	---

Definition at line 220 of file `hlist`.

Referenced by `cxx::H_list< Timeout >::pop_front()`.

Here is the caller graph for this function:

**16.47.2.8 `replace()`**

```
template<typename T, typename POLICY = Bits::Basic_list_policy< T, H_list_item>>
void cxx::H_list< T, POLICY >::replace (
    T * p,
    T * e) [inline], [static]
```

Replace an element in a list with a new element.

Parameters

<i>p</i>	Element in list to be replaced.
<i>e</i>	Replacement element, must not yet be in a list.

Precondition

p and *e* must not be NULL.

After the operation the *p* element is no longer in the list and may be reused.

Definition at line 204 of file [hlist](#).

The documentation for this class was generated from the following file:

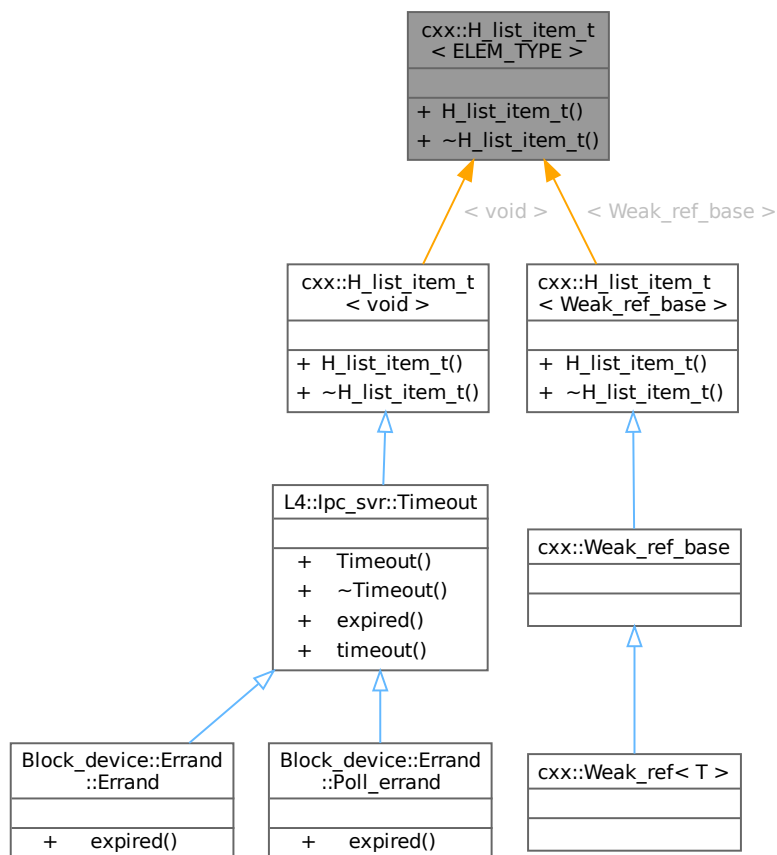
- l4/cxx/hlist

16.48 cxx::H_list_item_t< ELEM_TYPE > Class Template Reference

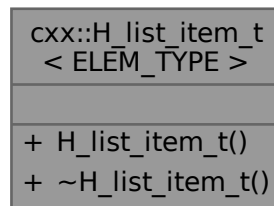
Basic element type for a double-linked [H_list](#).

```
#include <hlist>
```

Inheritance diagram for cxx::H_list_item_t< ELEM_TYPE >:



Collaboration diagram for `cxx::H_list_item_t< ELEM_TYPE >`:



Public Member Functions

- [H_list_item_t\(\)](#)
Constructor.
- [~H_list_item_t\(\)](#) noexcept
Destructor.

16.48.1 Detailed Description

```
template<typename ELEM_TYPE>
class cxx::H_list_item_t< ELEM_TYPE >
```

Basic element type for a double-linked [H_list](#).

Template Parameters

<code>ELEM_TYPE</code>	Base class of the list element.
------------------------	---------------------------------

Definition at line 22 of file [hlist](#).

16.48.2 Constructor & Destructor Documentation

16.48.2.1 H_list_item_t()

```
template<typename ELEM_TYPE>
cxx::H_list_item_t< ELEM_TYPE >::H_list_item_t () [inline]
```

Constructor.

Creates an element that is not in any list.

Definition at line 30 of file [hlist](#).

16.48.2.2 ~H_list_item_t()

```
template<typename ELEM_TYPE>
cxx::H_list_item_t< ELEM_TYPE >::~~H_list_item_t () [inline], [noexcept]
```

Destructor.

Automatically removes the element from any list it still might be enchainned in.

Definition at line 37 of file [hlist](#).

The documentation for this class was generated from the following file:

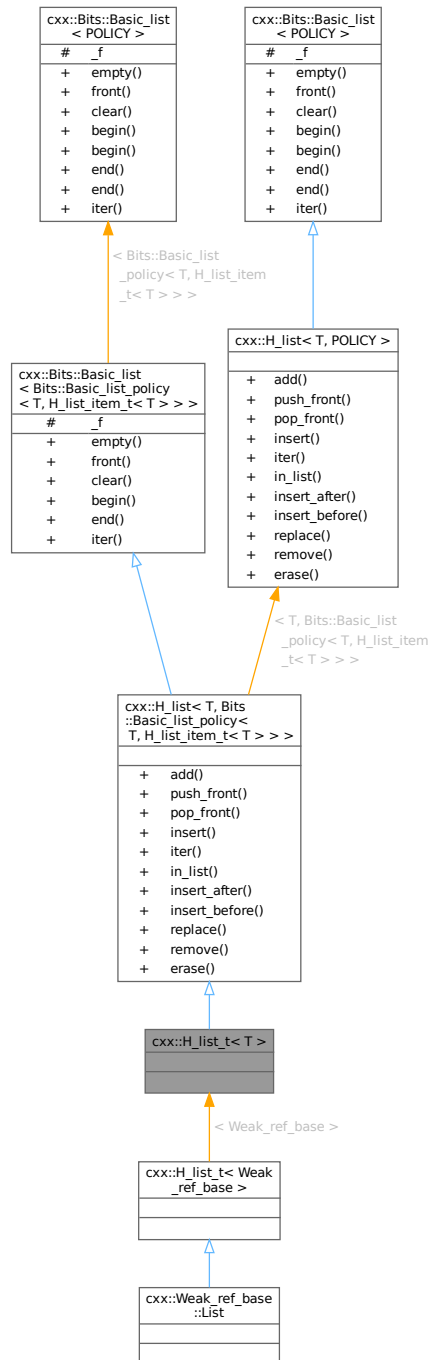
- I4/cxx/hlist

16.49 cxx::H_list_t< T > Struct Template Reference

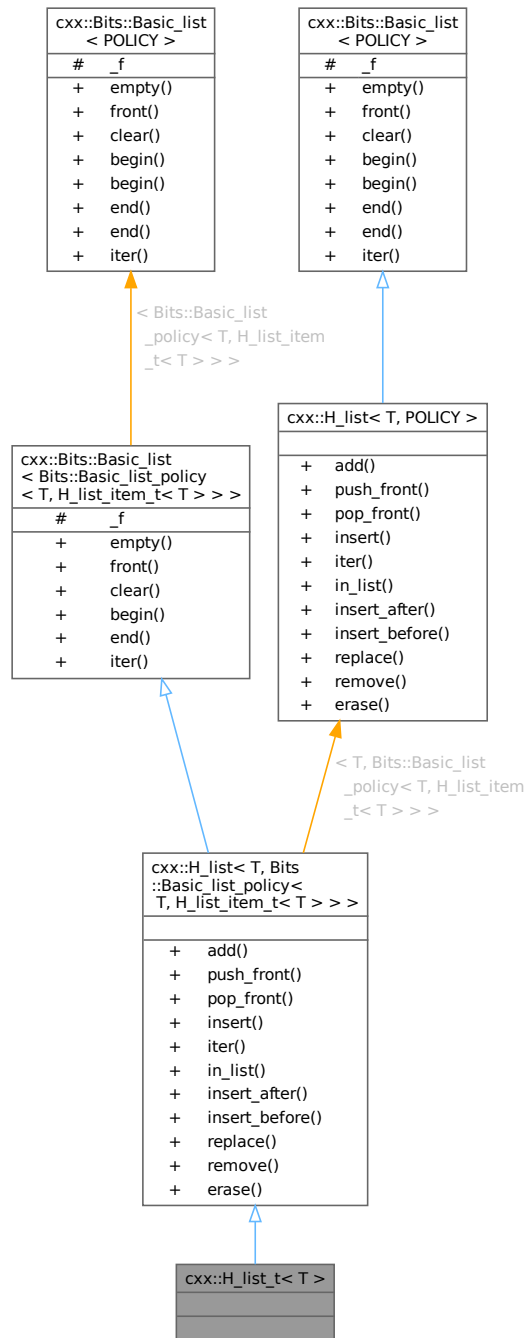
Double-linked list of typed [H_list_item_t](#) elements.

```
#include <hlist>
```

Inheritance diagram for cxx::H_list_t< T >:



Collaboration diagram for `cxx::H_list_t< T >`:



Additional Inherited Members

Public Member Functions inherited from

`cxx::H_list< T, Bits::Basic_list_policy< T, H_list_item_t< T > > >`

- `void add (T *e)`

- *Add element to the front of the list.*
- void **push_front** (T *e)
Add element to the front of the list.
- T * **pop_front** ()
Remove and return the head element of the list.
- Iterator **insert** (T *e, Iterator const &pred)
Insert an element at the iterator position.

Public Member Functions inherited from

`cxx::Bits::Basic_list< Bits::Basic_list_policy< T, H_list_item_t< T > > >`

- bool **empty** () const
Check if the list is empty.
- Value_type **front** () const
Return the first element in the list.
- void **clear** ()
Remove all elements from the list.
- Iterator **begin** ()
Return an iterator to the beginning of the list.
- Const_iterator **end** () const
Return a const iterator to the end of the list.

Static Public Member Functions inherited from

`cxx::H_list< T, Bits::Basic_list_policy< T, H_list_item_t< T > > >`

- static Iterator **iter** (T *c)
Return an iterator for an arbitrary list element.
- static bool **in_list** (T const *e)
Check if the given element is currently part of a list.
- static Iterator **insert_after** (T *e, Iterator const &pred)
Insert an element after the iterator position.
- static void **insert_before** (T *e, Iterator const &succ)
Insert an element before the iterator position.
- static void **replace** (T *p, T *e)
Replace an element in a list with a new element.
- static void **remove** (T *e)
Remove the given element from its list.
- static Iterator **erase** (Iterator const &e)
Remove the element at the given iterator position.

Static Public Member Functions inherited from

`cxx::Bits::Basic_list< Bits::Basic_list_policy< T, H_list_item_t< T > > >`

- static Const_iterator **iter** (Const_value_type c)
Return a const iterator that begins at the given element.

Protected Attributes inherited from**[cxx::Bits::Basic_list](#)< [Bits::Basic_list_policy](#)< T, [H_list_item_t](#)< T > > >**

- [Bits::Basic_list_policy](#)< T, [H_list_item_t](#)< T > >::Head_type _f

*Pointer to front of the list.***16.49.1 Detailed Description**

```
template<typename T>
struct cxx::H_list_t< T >
```

Double-linked list of typed [H_list_item_t](#) elements.

Note

H_lists are not self-cleaning. Elements that are still chained during destruction are not removed and will therefore be in an undefined state after the destruction.

Definition at line [248](#) of file [hlist](#).

The documentation for this struct was generated from the following file:

- [l4/cxx/hlist](#)

16.50 cxx::List< D, Alloc > Class Template Reference

Doubly linked list, with internal allocation.

```
#include <list>
```

Collaboration diagram for [cxx::List](#)< D, Alloc >:

cxx::List< D, Alloc >	
+	push_back()
+	push_front()
+	remove()
+	size()
+	operator[]()
+	operator[]()
+	items()

Data Structures

- class `Iter`
Iterator.

Public Member Functions

- void **push_back** (D const &d) noexcept
Add element at the end of the list.
- void **push_front** (D const &d) noexcept
Add element at the beginning of the list.
- void **remove** (`Iter` const &i) noexcept
Remove element pointed to by the iterator.
- unsigned long **size** () const noexcept
Get the length of the list.
- D const & **operator[]** (unsigned long idx) const noexcept
Random access.
- D & **operator[]** (unsigned long idx) noexcept
Random access.
- `Iter` **items** () noexcept
Get iterator for the list elements.

16.50.1 Detailed Description

`template<typename D, template< typename A > class Alloc = New_allocator>`
`class cxx::List< D, Alloc >`

Doubly linked list, with internal allocation.

Container for items of type D, implemented by a doubly linked list. Alloc defines the allocator policy.

Definition at line 323 of file `list`.

16.50.2 Member Function Documentation

16.50.2.1 `operator[]()` [1/2]

```
template<typename D, template< typename A > class Alloc = New_allocator>
D const & cxx::List< D, Alloc >::operator[] (
    unsigned long idx) const [inline], [noexcept]
```

Random access.

Complexity is O(n).

Definition at line 393 of file `list`.

16.50.2.2 operator[]() [2/2]

```
template<typename D, template< typename A > class Alloc = New_allocator>
D & cxx::List< D, Alloc >::operator[] (
    unsigned long idx) [inline], [noexcept]
```

Random access.

Complexity is O(n).

Definition at line 397 of file [list](#).

The documentation for this class was generated from the following file:

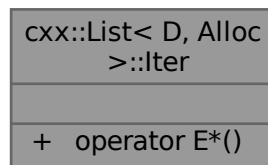
- l4/cxx/list

16.51 cxx::List< D, Alloc >::Iter Class Reference

Iterator.

```
#include <list>
```

Collaboration diagram for cxx::List< D, Alloc >::Iter:



Public Member Functions

- **operator E* ()** const noexcept
operator for testing validity (syntactically equal to pointers)

16.51.1 Detailed Description

```
template<typename D, template< typename A > class Alloc = New_allocator>
class cxx::List< D, Alloc >::Iter
```

Iterator.

Forward and backward iterable.

Definition at line 343 of file [list](#).

The documentation for this class was generated from the following file:

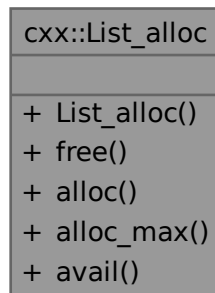
- l4/cxx/list

16.52 cxx::List_alloc Class Reference

Standard list-based allocator.

```
#include <list_alloc>
```

Collaboration diagram for cxx::List_alloc:



Public Member Functions

- [List_alloc\(\)](#)
Initializes an empty list allocator.
- void [free](#) (void *block, unsigned long size, bool initial_free=false)
Return a free memory block to the allocator.
- void * [alloc](#) (unsigned long size, unsigned long align, unsigned long lower=0, unsigned long upper=~0UL)
Allocate a memory block.
- void * [alloc_max](#) (unsigned long min, unsigned long *max, unsigned long align, unsigned granularity, unsigned long lower=0, unsigned long upper=~0UL)
Allocate a memory block of $min \leq size \leq max$.
- unsigned long [avail](#) ()
Get the amount of available memory.

16.52.1 Detailed Description

Standard list-based allocator.

Definition at line 21 of file [list_alloc](#).

16.52.2 Constructor & Destructor Documentation

16.52.2.1 List_alloc()

```
cxx::List_alloc::List_alloc () [inline]
```

Initializes an empty list allocator.

Note

To initialize the allocator with available memory use the [free\(\)](#) function.

Definition at line 46 of file [list_alloc](#).

16.52.3 Member Function Documentation

16.52.3.1 alloc()

```
void * cxx::List_alloc::alloc (
    unsigned long size,
    unsigned long align,
    unsigned long lower = 0,
    unsigned long upper = ~0UL) [inline]
```

Allocate a memory block.

Parameters

<i>size</i>	Size of the memory block.
<i>align</i>	Alignment constraint.
<i>lower</i>	Lower bound of the physical region the memory block should be allocated from.
<i>upper</i>	Upper bound of the physical region the memory block should be allocated from, value is inclusive.

Returns

Pointer to memory block

Precondition

$0 < \text{size} \leq \sim 0\text{UL} - 32$.

Definition at line 389 of file [list_alloc](#).

16.52.3.2 alloc_max()

```
void * cxx::List_alloc::alloc_max (
    unsigned long min,
    unsigned long * max,
    unsigned long align,
    unsigned granularity,
    unsigned long lower = 0,
    unsigned long upper = ~0UL) [inline]
```

Allocate a memory block of $\text{min} \leq \text{size} \leq \text{max}$.

Parameters

	<i>min</i>	Minimal size to allocate (in bytes).
<i>in, out</i>	<i>max</i>	Maximum size to allocate (in bytes). The actual allocated size is returned here.
	<i>align</i>	Alignment constraint.
	<i>granularity</i>	Granularity to use for the allocation (power of 2).

	<i>lower</i>	Lower bound of the physical region the memory block should be allocated from.
	<i>upper</i>	Upper bound of the physical region the memory block should be allocated from, value is inclusive.

Returns

Pointer to memory block

Precondition

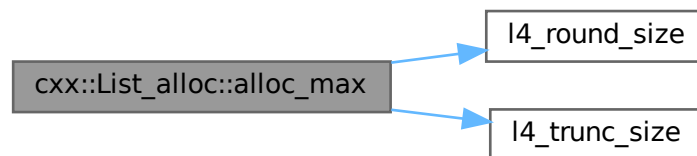
$0 < \text{min} \leq \sim 0\text{UL} - 32$.

$0 < \text{max}$.

Definition at line 269 of file `list_alloc`.

References `l4_round_size()`, and `l4_trunc_size()`.

Here is the call graph for this function:

**16.52.3.3 `avail()`**

```
unsigned long cxx::List_alloc::avail () [inline]
```

Get the amount of available memory.

Returns

Available memory in bytes

Definition at line 477 of file `list_alloc`.

16.52.3.4 `free()`

```
void cxx::List_alloc::free (
    void * block,
    unsigned long size,
    bool initial_free = false) [inline]
```

Return a free memory block to the allocator.

Parameters

<i>block</i>	Pointer to memory block.
<i>size</i>	Size of memory block.
<i>initial_free</i>	Set to true for putting fresh memory to the allocator. This will enforce alignment on that memory.

Precondition

block must not be NULL.
 $2 * \text{sizeof}(\text{void} *) \leq \text{size} \leq \sim 0\text{UL} - 32.$

Definition at line 228 of file [list_alloc](#).

The documentation for this class was generated from the following file:

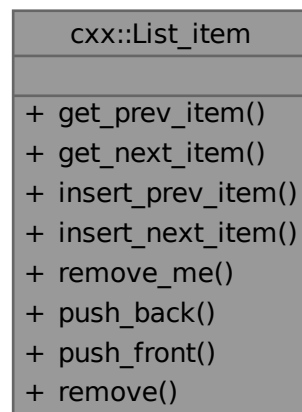
- `l4/cxx/list_alloc`

16.53 cxx::List_item Class Reference

Basic list item.

```
#include <list>
```

Collaboration diagram for `cxx::List_item`:

**Data Structures**

- class [Iter](#)
Iterator for a list of ListItem-s.
- class [T_iter](#)
Iterator for derived classes from ListItem.

Public Member Functions

- [List_item](#) * **get_prev_item** () const noexcept
Get previous item.
- [List_item](#) * **get_next_item** () const noexcept
Get next item.
- void **insert_prev_item** ([List_item](#) *p) noexcept
Insert item p before this item.
- void **insert_next_item** ([List_item](#) *p) noexcept
Insert item p after this item.
- void **remove_me** () noexcept
Remove this item from the list.

Static Public Member Functions

- template<typename C, typename N>
static C * **push_back** (C *head, N *p) noexcept
Append item to a list.
- template<typename C, typename N>
static C * **push_front** (C *head, N *p) noexcept
Prepend item to a list.
- template<typename C, typename N>
static C * **remove** (C *head, N *p) noexcept
Remove item from a list.

16.53.1 Detailed Description

Basic list item.

Basic item that can be member of a doubly linked, cyclic list.

Definition at line 26 of file [list](#).

16.53.2 Member Function Documentation

16.53.2.1 push_back()

```
template<typename C, typename N>
C * cxx::List_item::push_back (
    C * head,
    N * p) [inline], [static], [noexcept]
```

Append item to a list.

Convenience function for empty-head corner case.

Parameters

<i>head</i>	Pointer to the current list head.
<i>p</i>	Pointer to new item.

Returns

the pointer to the new head.

Definition at line 237 of file [list](#).

Referenced by [cxx::List< D, Alloc >::push_back\(\)](#).

Here is the caller graph for this function:

**16.53.2.2 push_front()**

```

template<typename C, typename N>
C * cxx::List_item::push_front (
    C * head,
    N * p) [inline], [static], [noexcept]
  
```

Prepend item to a list.

Convenience function for empty-head corner case.

Parameters

<i>head</i>	pointer to the current list head.
<i>p</i>	pointer to new item.

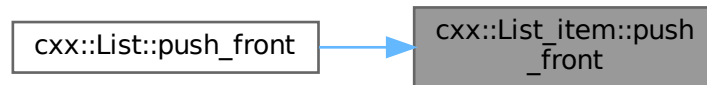
Returns

the pointer to the new head.

Definition at line 248 of file [list](#).

Referenced by [cxx::List< D, Alloc >::push_front\(\)](#).

Here is the caller graph for this function:



16.53.2.3 remove()

```

template<typename C, typename N>
C * cxx::List_item::remove (
    C * head,
    N * p) [inline], [static], [noexcept]
  
```

Remove item from a list.

Convenience function for remove-head corner case.

Parameters

<i>head</i>	pointer to the current list head.
<i>p</i>	pointer to the item to remove.

Returns

the pointer to the new head.

Definition at line 258 of file [list](#).

Referenced by [cxx::List< D, Alloc >::remove\(\)](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

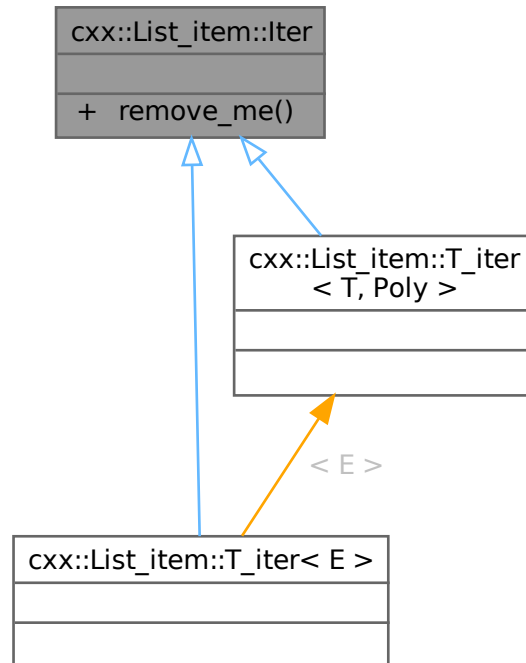
- `I4/cxx/list`

16.54 cxx::List_item::Iter Class Reference

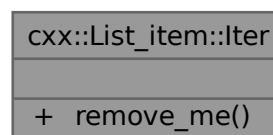
Iterator for a list of ListItem-s.

```
#include <list>
```

Inheritance diagram for cxx::List_item::Iter:



Collaboration diagram for cxx::List_item::Iter:



Public Member Functions

- [List_item](#) * **remove_me** () noexcept
Remove item pointed to by iterator, and return pointer to element.

16.54.1 Detailed Description

Iterator for a list of ListItem-s.

The Iterator iterates till it finds the first element again.

Definition at line 34 of file [list](#).

The documentation for this class was generated from the following file:

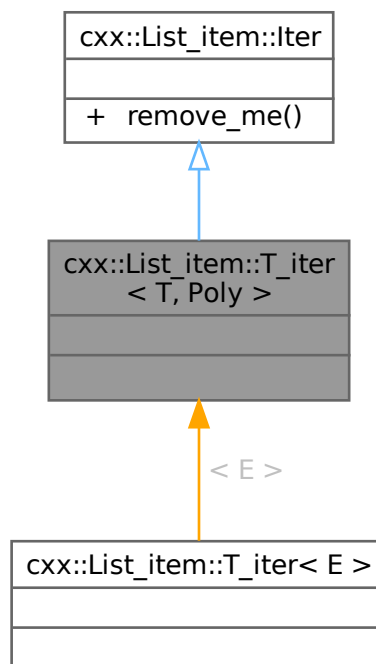
- l4/cxx/list

16.55 cxx::List_item::T_iter< T, Poly > Class Template Reference

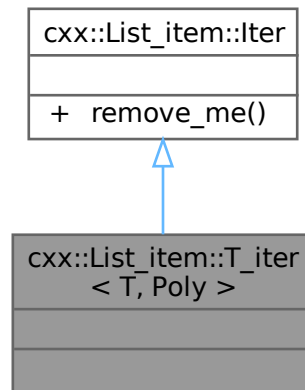
Iterator for derived classes from ListItem.

```
#include <list>
```

Inheritance diagram for cxx::List_item::T_iter< T, Poly >:



Collaboration diagram for `cxx::List_item::T_iter< T, Poly >`:



Additional Inherited Members

Public Member Functions inherited from `cxx::List_item::Iter`

- `List_item * remove_me ()` noexcept
Remove item pointed to by iterator, and return pointer to element.

16.55.1 Detailed Description

```
template<typename T, bool Poly = false>
class cxx::List_item::T_iter< T, Poly >
```

Iterator for derived classes from `ListItem`.

Allows direct access to derived classes by `*` operator.

Example: `class Foo : public ListItem { public: typedef T_iter<Foo> Iter; ... };`

Definition at line 108 of file `list`.

The documentation for this class was generated from the following file:

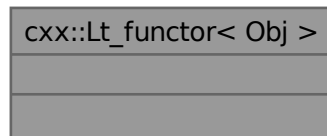
- `I4/cxx/list`

16.56 cxx::Lt_functor< Obj > Struct Template Reference

Generic comparator class that defaults to the less-than operator.

```
#include <std_ops>
```

Collaboration diagram for cxx::Lt_functor< Obj >:



16.56.1 Detailed Description

```
template<typename Obj>
struct cxx::Lt_functor< Obj >
```

Generic comparator class that defaults to the less-than operator.

Definition at line 18 of file [std_ops](#).

The documentation for this struct was generated from the following file:

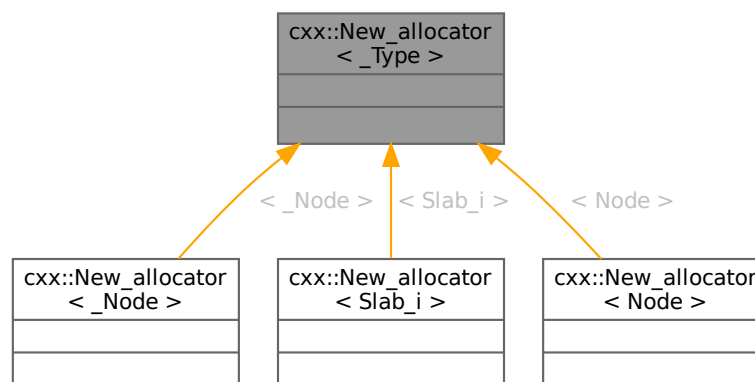
- l4/cxx/std_ops

16.57 cxx::New_allocator< _Type > Class Template Reference

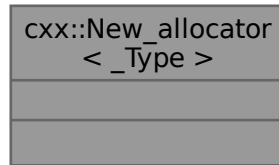
Standard allocator based on `operator new ()`.

```
#include <std_alloc>
```

Inheritance diagram for cxx::New_allocator< _Type >:



Collaboration diagram for `cxx::New_allocator<_Type>`:



16.57.1 Detailed Description

```
template<typename _Type>
class cxx::New_allocator<_Type>
```

Standard allocator based on `operator new ()`.

This allocator is the default allocator used for the *cxx Containers*, such as [cxx::Avl_set](#) and [cxx::Avl_map](#), to allocate the internal data structures.

Definition at line 56 of file [std_alloc](#).

The documentation for this class was generated from the following file:

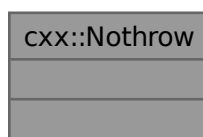
- `I4/cxx/std_alloc`

16.58 cxx::Nothrow Class Reference

Helper type to distinguish the `operator new` version that does not throw exceptions.

```
#include <std_alloc>
```

Collaboration diagram for `cxx::Nothrow`:



16.58.1 Detailed Description

Helper type to distinguish the `operator new` version that does not throw exceptions.

Definition at line 19 of file [std_alloc](#).

The documentation for this class was generated from the following file:

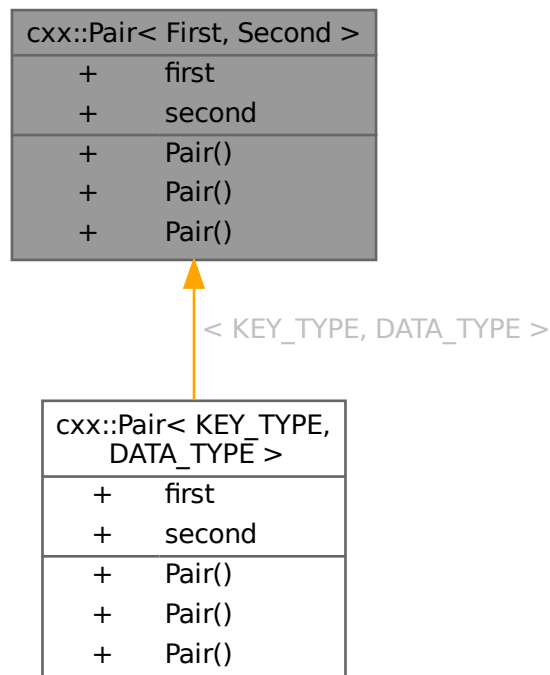
- `l4/cxx/std_alloc`

16.59 `cxx::Pair< First, Second >` Struct Template Reference

[Pair](#) of two values.

```
#include <pair>
```

Inheritance diagram for `cxx::Pair< First, Second >`:



Collaboration diagram for `cxx::Pair< First, Second >`:

<code>cxx::Pair< First, Second ></code>	
+	<code>first</code>
+	<code>second</code>
+	<code>Pair()</code>
+	<code>Pair()</code>
+	<code>Pair()</code>

Public Types

- typedef First **First_type**
Type of first value.
- typedef Second **Second_type**
Type of second value.

Public Member Functions

- `template<typename A1, typename A2>`
`Pair (A1 &&first, A2 &&second)`
Create a pair from the two values.
- `template<typename A1>`
`Pair (A1 &&first)`
Create a pair, default constructing the second value.
- `Pair ()=default`
Default construction.

Data Fields

- First **first**
First value.
- Second **second**
Second value.

16.59.1 Detailed Description

`template<typename First, typename Second>`
`struct cxx::Pair< First, Second >`

[Pair](#) of two values.

Standard container for a pair of values.

Parameters

<i>First</i>	Type of the first value.
<i>Second</i>	Type of the second value.

Definition at line 27 of file [pair](#).

16.59.2 Constructor & Destructor Documentation

16.59.2.1 `Pair()` [1/2]

```
template<typename First, typename Second>
template<typename A1, typename A2>
cxx::Pair< First, Second >::Pair (
    A1 && first,
    A2 && second) [inline]
```

Create a pair from the two values.

Parameters

<i>first</i>	The first value.
<i>second</i>	The second value.

Definition at line 45 of file [pair](#).

16.59.2.2 `Pair()` [2/2]

```
template<typename First, typename Second>
template<typename A1>
cxx::Pair< First, Second >::Pair (
    A1 && first) [inline]
```

Create a pair, default constructing the second value.

Parameters

<i>first</i>	The first value.
--------------	------------------

Definition at line 53 of file [pair](#).

The documentation for this struct was generated from the following file:

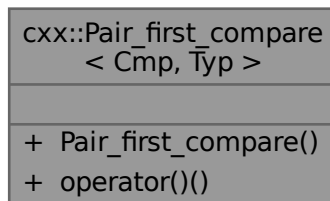
- [I4/cxx/pair](#)

16.60 `cxx::Pair_first_compare< Cmp, Typ >` Class Template Reference

Comparison functor for [Pair](#).

```
#include <pair>
```

Collaboration diagram for `cxx::Pair_first_compare< Cmp, Typ >`:



Public Member Functions

- [Pair_first_compare](#) (Cmp const &cmp=Cmp())
Construction.
- bool [operator\(\)](#) (Typ const &l, Typ const &r) const
Do the comparison based on the first value.

16.60.1 Detailed Description

```
template<typename Cmp, typename Typ>
class cxx::Pair_first_compare< Cmp, Typ >
```

Comparison functor for [Pair](#).

Parameters

<i>Cmp</i>	Comparison functor for the first value of the pair.
<i>Typ</i>	The pair type.

This functor can be used to compare [Pair](#) values with respect to the first value.

Definition at line 74 of file [pair](#).

16.60.2 Constructor & Destructor Documentation

16.60.2.1 `Pair_first_compare()`

```
template<typename Cmp, typename Typ>
cxx::Pair_first_compare< Cmp, Typ >::Pair_first_compare (
    Cmp const & cmp = Cmp()) [inline]
```

Construction.

Parameters

<code>cmp</code>	The comparison functor used for the first value.
------------------	--

Definition at line 84 of file [pair](#).

16.60.3 Member Function Documentation

16.60.3.1 `operator>()()`

```
template<typename Cmp, typename Typ>
bool cxx::Pair_first_compare< Cmp, Typ >::operator() (
    Typ const & l,
    Typ const & r) const [inline]
```

Do the comparison based on the first value.

Parameters

<code>l</code>	The lefthand value.
<code>r</code>	The righthand value.

Definition at line 91 of file [pair](#).

The documentation for this class was generated from the following file:

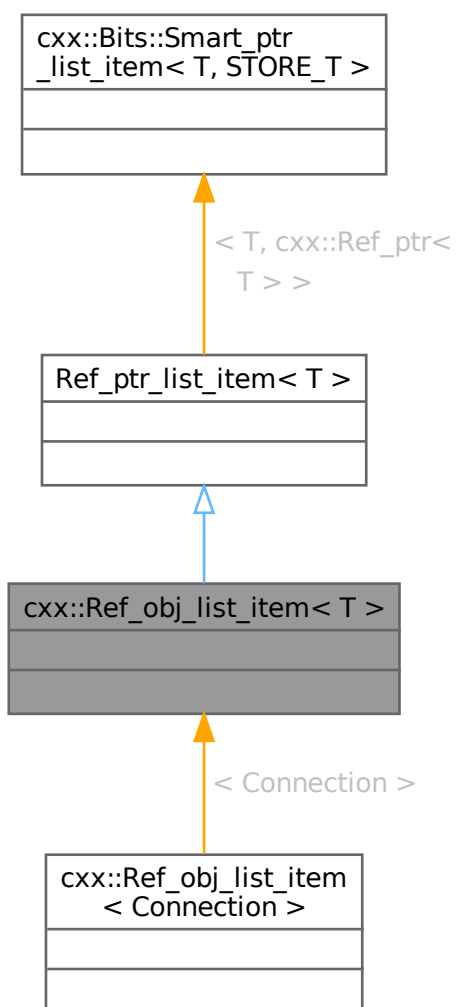
- [l4/cxx/pair](#)

16.61 `cxx::Ref_obj_list_item< T >` Struct Template Reference

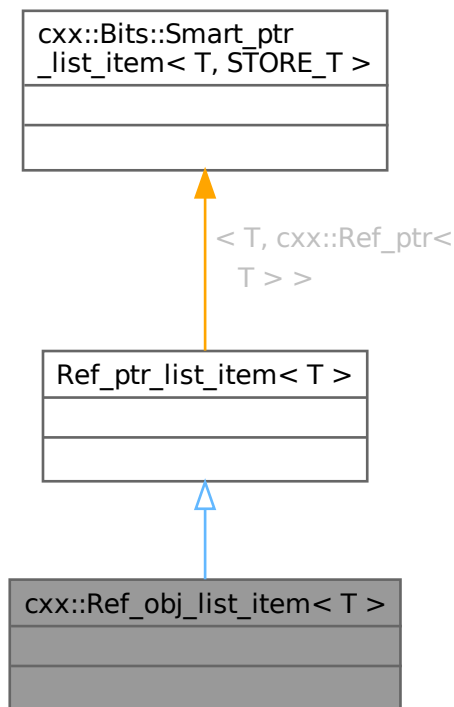
Item for list linked via [cxx::Ref_ptr](#) with default refence counting.

```
#include <ref_ptr_list>
```

Inheritance diagram for `cxx::Ref_obj_list_item< T >`:



Collaboration diagram for `cxx::Ref_obj_list_item< T >`:



16.61.1 Detailed Description

```
template<typename T>
struct cxx::Ref_obj_list_item< T >
```

Item for list linked via `cxx::Ref_ptr` with default reference counting.

Definition at line 26 of file `ref_ptr_list`.

The documentation for this struct was generated from the following file:

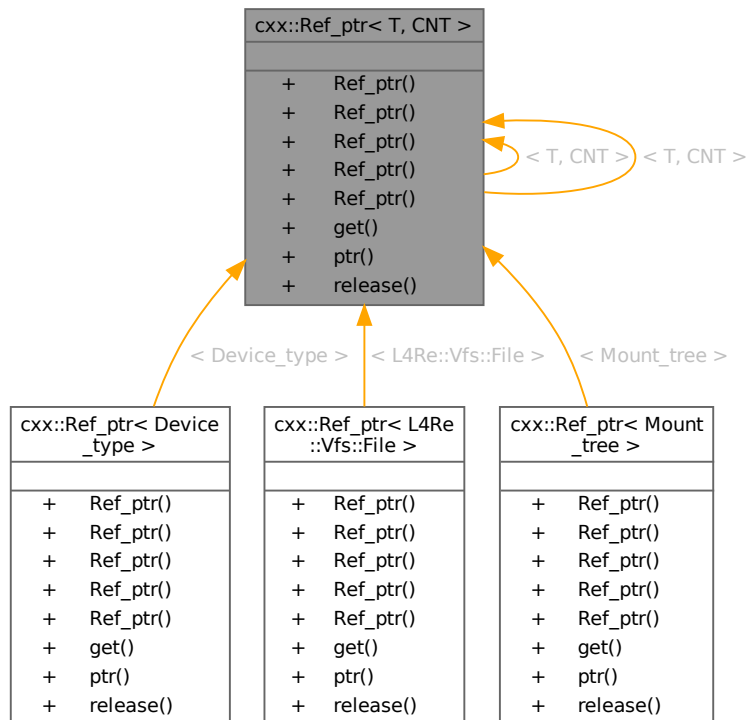
- `I4/cxx/ref_ptr_list`

16.62 `cxx::Ref_ptr< T, CNT >` Class Template Reference

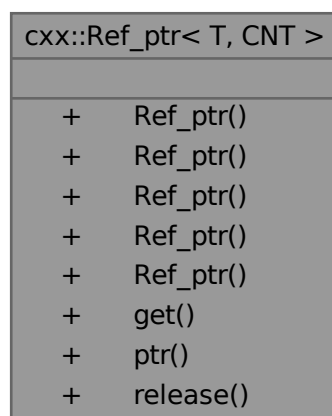
A reference-counting pointer with automatic cleanup.

```
#include <ref_ptr>
```

Inheritance diagram for `cxx::Ref_ptr< T, CNT >`:



Collaboration diagram for `cxx::Ref_ptr< T, CNT >`:



Public Member Functions

- `Ref_ptr()` noexcept

- Default constructor creates a pointer with no managed object.*
- `Ref_ptr (Wp const &o)` noexcept
Create a shared pointer from a weak pointer.
- `Ref_ptr (decltype(nullptr) n)` noexcept
allow creation from `nullptr`
- `template<typename X>`
`Ref_ptr (X *o)` noexcept
Create a shared pointer from a raw pointer.
- `Ref_ptr (T *o, bool d)` noexcept
Create a shared pointer from a raw pointer without creating a new reference.
- `T * get ()` const noexcept
Return a raw pointer to the object this shared pointer points to.
- `T * ptr ()` const noexcept
Return a raw pointer to the object this shared pointer points to.
- `T * release ()` noexcept
Release the shared pointer without removing the reference.

16.62.1 Detailed Description

`template<typename T = void, template< typename X > class CNT = Default_ref_counter>`
`class cxx::Ref_ptr< T, CNT >`

A reference-counting pointer with automatic cleanup.

Template Parameters

<i>T</i>	Type of object the pointer points to.
<i>CNT</i>	Type of management class that manages the life time of the object.

This pointer is similar to the standard C++-11 `shared_ptr` but it does the reference counting directly in the object being pointed to, so that no additional management structures need to be allocated from the heap.

Classes that use this pointer type must implement two functions:

```
int remove_ref()
```

is called when a reference is removed and must return 0 when there are no further references to the object.

```
void add_ref()
```

is called when another `ref_ptr` to the object is created.

`Ref_obj` provides a simple implementation of this interface from which classes may inherit.

Examples

[tmpfs/lib/src/fs.cc](#).

Definition at line 70 of file [ref_ptr](#).

16.62.2 Constructor & Destructor Documentation

16.62.2.1 Ref_ptr() [1/3]

```
template<typename T = void, template< typename X > class CNT = Default_ref_counter>
cxx::Ref_ptr< T, CNT >::Ref_ptr (
    Wp const & o) [inline], [noexcept]
```

Create a shared pointer from a weak pointer.

Increases references.

Definition at line 88 of file [ref_ptr](#).

16.62.2.2 Ref_ptr() [2/3]

```
template<typename T = void, template< typename X > class CNT = Default_ref_counter>
template<typename X>
cxx::Ref_ptr< T, CNT >::Ref_ptr (
    X * o) [inline], [explicit], [noexcept]
```

Create a shared pointer from a raw pointer.

In contrast to C++11 `shared_ptr` it is safe to use this constructor multiple times and have the same reference counter.

Definition at line 101 of file [ref_ptr](#).

16.62.2.3 Ref_ptr() [3/3]

```
template<typename T = void, template< typename X > class CNT = Default_ref_counter>
cxx::Ref_ptr< T, CNT >::Ref_ptr (
    T * o,
    bool d) [inline], [noexcept]
```

Create a shared pointer from a raw pointer without creating a new reference.

Parameters

<i>o</i>	Pointer to the object.
<i>d</i>	Dummy parameter to select this constructor at compile time. The value may be true or false.

This is the counterpart to [release\(\)](#).

Definition at line 114 of file [ref_ptr](#).

16.62.3 Member Function Documentation

16.62.3.1 get()

```
template<typename T = void, template< typename X > class CNT = Default_ref_counter>
T * cxx::Ref_ptr< T, CNT >::get () const [inline], [noexcept]
```

Return a raw pointer to the object this shared pointer points to.

This does not release the pointer or decrease the reference count.

Definition at line 121 of file [ref_ptr](#).

16.62.3.2 ptr()

```
template<typename T = void, template< typename X > class CNT = Default_ref_counter>
T * cxx::Ref_ptr< T, CNT >::ptr () const [inline], [noexcept]
```

Return a raw pointer to the object this shared pointer points to.

This does not release the pointer or decrease the reference count.

Definition at line 127 of file [ref_ptr](#).

16.62.3.3 release()

```
template<typename T = void, template< typename X > class CNT = Default_ref_counter>
T * cxx::Ref_ptr< T, CNT >::release () [inline], [noexcept]
```

Release the shared pointer without removing the reference.

Returns

A raw pointer to the managed object.

Definition at line 138 of file [ref_ptr](#).

The documentation for this class was generated from the following file:

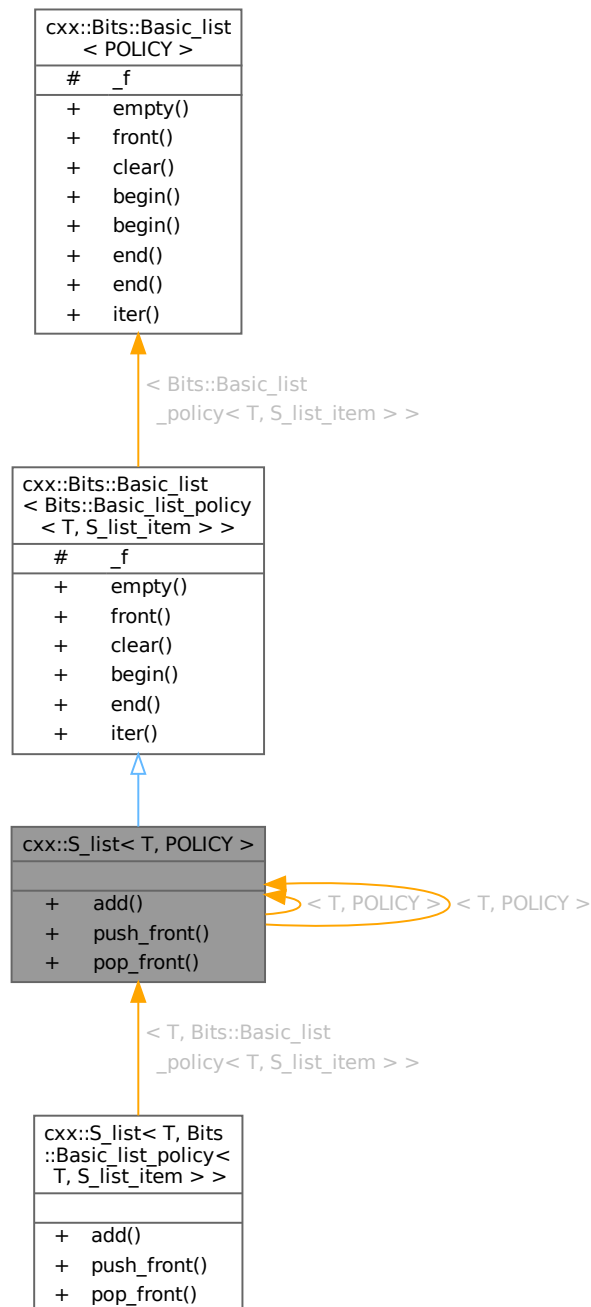
- l4/cxx/ref_ptr

16.63 cxx::S_list< T, POLICY > Class Template Reference

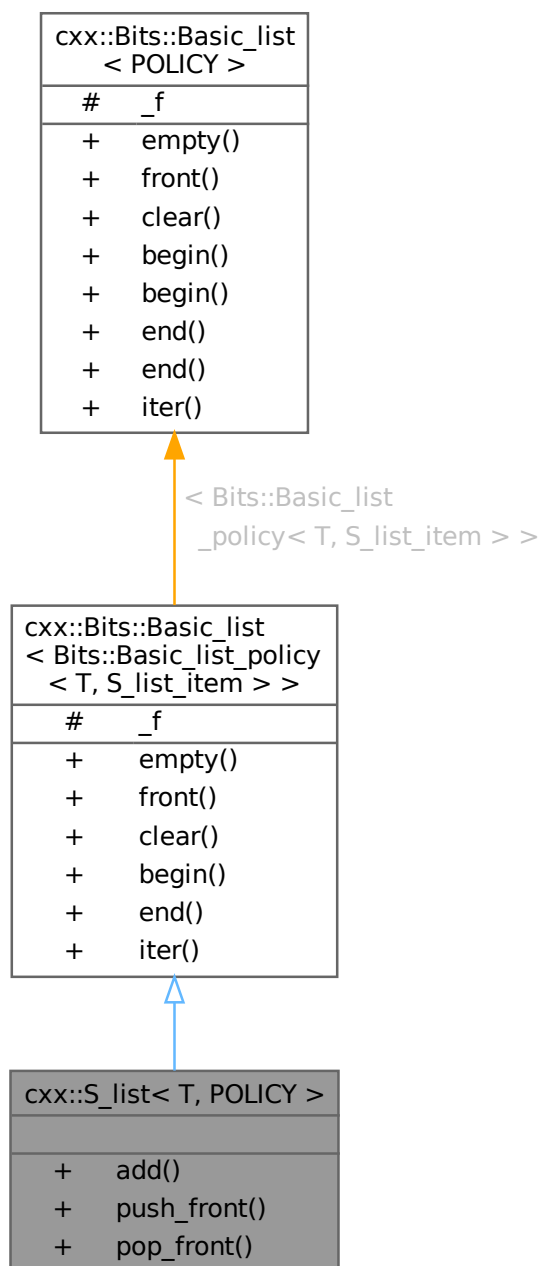
Simple single-linked list.

```
#include <slist>
```

Inheritance diagram for cxx::S_list< T, POLICY >:



Collaboration diagram for cxx::S_list< T, POLICY >:



Public Member Functions

- void **add** (T *e)
Add an element to the front of the list.
- void **push_front** (T *e)
Add an element to the front of the list.
- T * **pop_front** ()
Remove and return the head element of the list.

Public Member Functions inherited from**cxx::Bits::Basic_list< Bits::Basic_list_policy< T, S_list_item > >**

- bool **empty** () const
Check if the list is empty.
- Value_type **front** () const
Return the first element in the list.
- void **clear** ()
Remove all elements from the list.
- Iterator **begin** ()
Return an iterator to the beginning of the list.
- Const_iterator **end** () const
Return a const iterator to the end of the list.

Additional Inherited Members**Static Public Member Functions inherited from****cxx::Bits::Basic_list< Bits::Basic_list_policy< T, S_list_item > >**

- static Const_iterator **iter** (Const_value_type c)
Return a const iterator that begins at the given element.

Protected Attributes inherited from**cxx::Bits::Basic_list< Bits::Basic_list_policy< T, S_list_item > >**

- Bits::Basic_list_policy< T, S_list_item >::Head_type _f
Pointer to front of the list.

16.63.1 Detailed Description

```
template<typename T, typename POLICY = Bits::Basic_list_policy< T, S_list_item >>
class cxx::S_list< T, POLICY >
```

Simple single-linked list.

Template Parameters

<i>T</i>	Type of elements saved in the list. Must inherit from cxx::S_list_item
----------	--

Definition at line 40 of file [slist](#).

16.63.2 Member Function Documentation

16.63.2.1 `pop_front()`

```
template<typename T, typename POLICY = Bits::Basic_list_policy< T, S_list_item >>
T * cxx::S_list< T, POLICY >::pop_front () [inline]
```

Remove and return the head element of the list.

Precondition

The list must not be empty or the behaviour will be undefined.

Definition at line 89 of file `slist`.

The documentation for this class was generated from the following file:

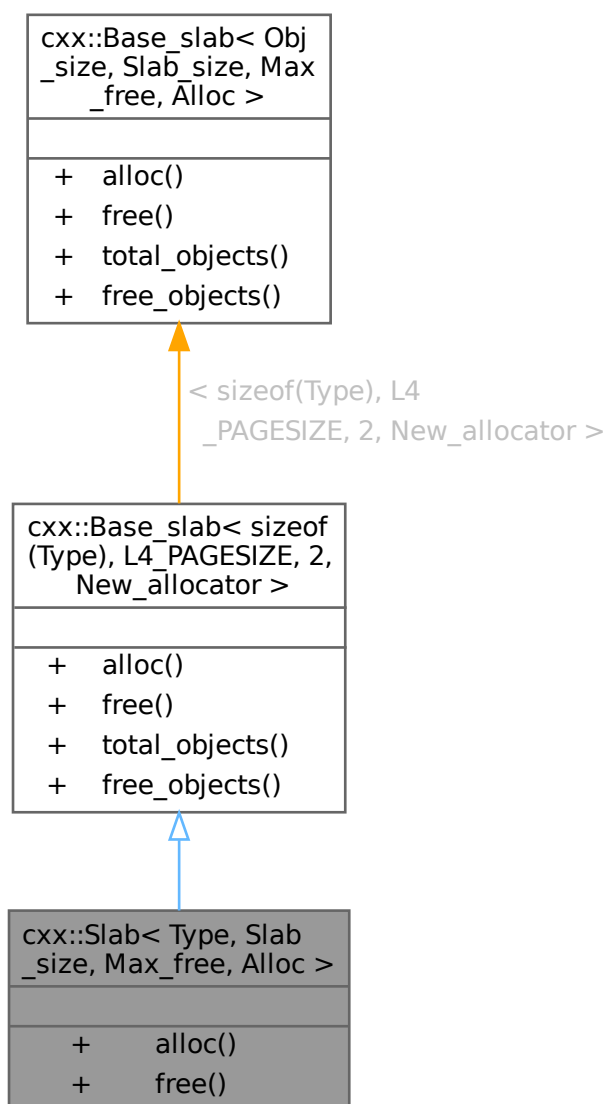
- `I4/cxx/slist`

16.64 `cxx::Slab< Type, Slab_size, Max_free, Alloc >` Class Template Reference

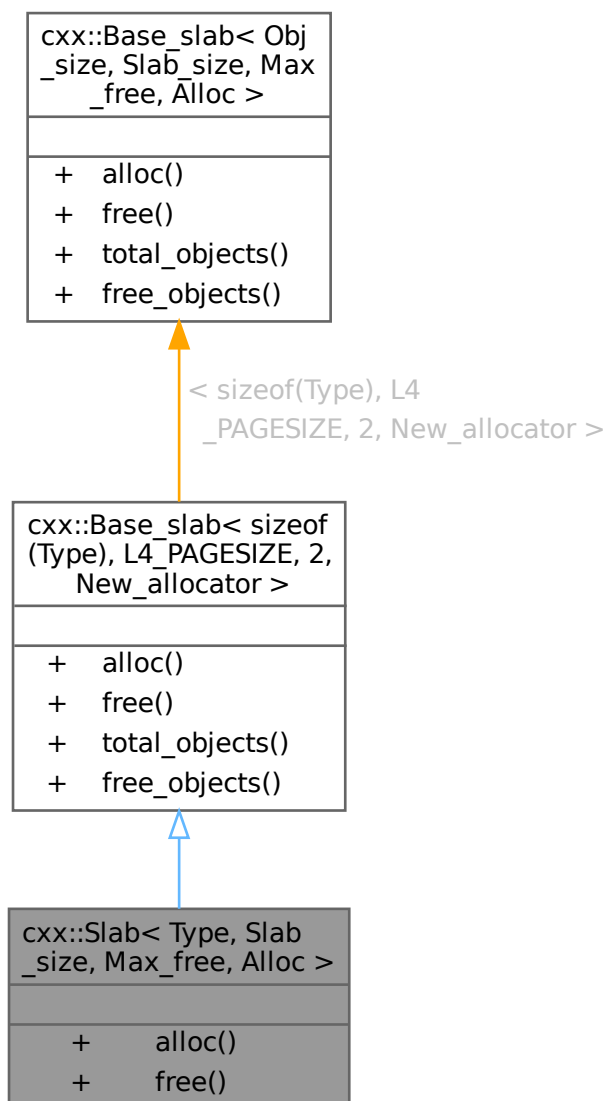
`Slab` allocator for object of type `Type`.

```
#include <slab_alloc>
```

Inheritance diagram for `cxx::Slab< Type, Slab_size, Max_free, Alloc >`:



Collaboration diagram for `cxx::Slab< Type, Slab_size, Max_free, Alloc >`:



Public Member Functions

- `Type * alloc ()` noexcept
Allocate an object of type `Type`.
- `void free (Type *o)` noexcept
Free the object addressed by `o`.

Public Member Functions inherited from

`cxx::Base_slab< sizeof(Type), L4_PAGESIZE, 2, New_allocator >`

- `void * alloc ()` noexcept

- *Allocate a new object.*
void [free](#) (void *_o) noexcept
- *Free the given object (_o).*
unsigned [total_objects](#) () const noexcept
- *Get the total number of objects managed by the slab allocator.*
unsigned [free_objects](#) () const noexcept
- *Get the number of objects which can be allocated before a new empty slab needs to be added to the slab allocator.*

Additional Inherited Members

Public Types inherited from

[cxx::Base_slab](#)< [sizeof\(Type\)](#), [L4_PAGESIZE](#), [2](#), [New_allocator](#) >

- typedef [New_allocator](#)< Slab_i > [Slab_alloc](#)
Type of the backend allocator.

16.64.1 Detailed Description

```
template<typename Type, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A > class
Alloc = New_allocator>
class cxx::Slab< Type, Slab_size, Max_free, Alloc >
```

[Slab](#) allocator for object of type [Type](#).

Template Parameters

<i>Type</i>	The type of the objects to manage.
<i>Slab_size</i>	Size of a slab.
<i>Max_free</i>	The maximum number of free slabs.
<i>Alloc</i>	The allocator for the slabs.

Definition at line [335](#) of file [slab_alloc](#).

16.64.2 Member Function Documentation

16.64.2.1 [alloc\(\)](#)

```
template<typename Type, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A >
class Alloc = New_allocator>
Type * cxx::Slab< Type, Slab_size, Max_free, Alloc >::alloc () [inline], [noexcept]
```

Allocate an object of type [Type](#).

Returns

A pointer to the object just allocated, or 0 on failure.

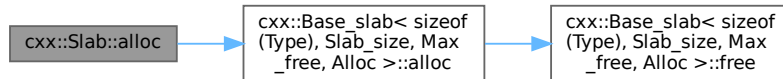
Note

The user is responsible for initializing the object.

Definition at line 355 of file [slab_alloc](#).

References [cxx::Base_slab< sizeof\(Type\), Slab_size, Max_free, Alloc >::alloc\(\)](#).

Here is the call graph for this function:

**16.64.2.2 free()**

```

template<typename Type, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A >
class Alloc = New_allocator>
void cxx::Slab< Type, Slab_size, Max_free, Alloc >::free (
    Type * o) [inline], [noexcept]
  
```

Free the object addressed by `o`.

Parameters

<code>o</code>	The pointer to the object to free.
----------------	------------------------------------

Precondition

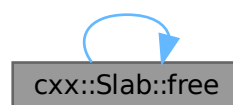
The object must have been allocated with this allocator.

Definition at line 366 of file [slab_alloc](#).

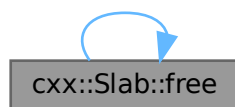
References [free\(\)](#).

Referenced by [free\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

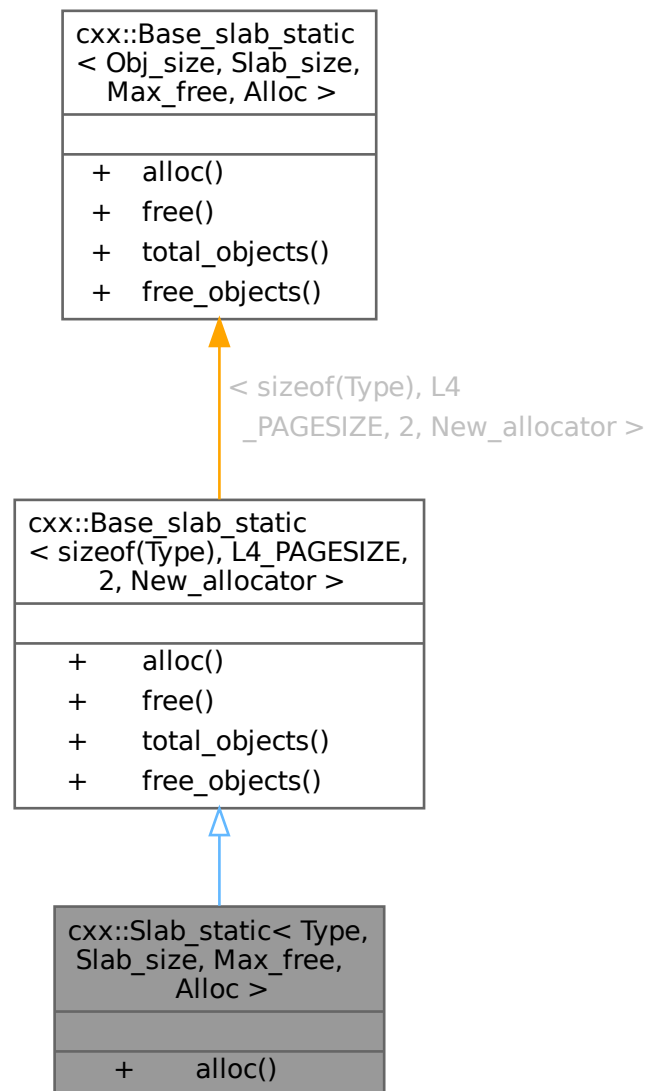
- `l4/cxx/slab_alloc`

16.65 `cxx::Slab_static< Type, Slab_size, Max_free, Alloc > Class` Template Reference

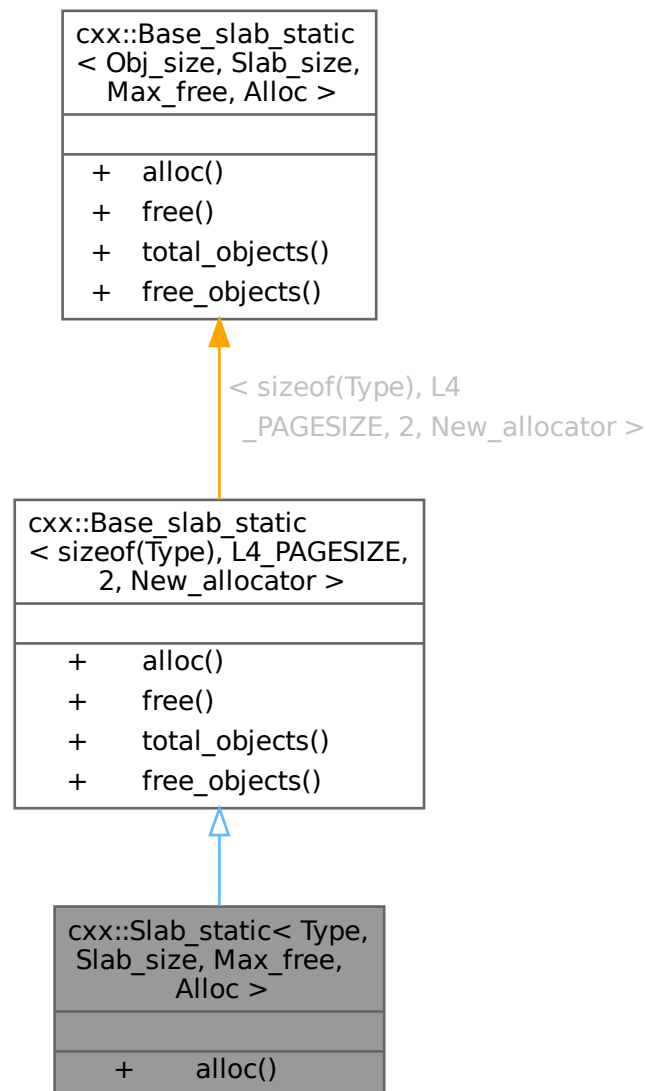
Merged slab allocator (allocators for objects of the same size are merged together).

```
#include <slab_alloc>
```


Inheritance diagram for cxx::Slab_static< Type, Slab_size, Max_free, Alloc >:



Collaboration diagram for `cxx::Slab_static< Type, Slab_size, Max_free, Alloc >`:



Public Member Functions

- `Type * alloc ()` noexcept
Allocate an object of type `Type`.

Public Member Functions inherited from

`cxx::Base_slab_static< sizeof(Type), L4_PAGESIZE, 2, New_allocator >`

- `void * alloc ()` noexcept
Allocate an object.

- void `free` (void *p) noexcept
Free the given object (p).
- unsigned `total_objects` () const noexcept
Get the total number of objects managed by the slab allocator.
- unsigned `free_objects` () const noexcept
Get the number of free objects in the slab allocator.

Additional Inherited Members

Public Types inherited from

`cxx::Base_slab_static< sizeof(Type), L4_PAGESIZE, 2, New_allocator >`

16.65.1 Detailed Description

```
template<typename Type, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A > class
Alloc = New_allocator>
class cxx::Slab_static< Type, Slab_size, Max_free, Alloc >
```

Merged slab allocator (allocators for objects of the same size are merged together).

Template Parameters

<i>Type</i>	The type of the objects to manage.
<i>Slab_size</i>	The size of a slab.
<i>Max_free</i>	The maximum number of free slabs.
<i>Alloc</i>	The allocator for the slabs.

This slab allocator class is useful for merging slab allocators with the same parameters (equal `sizeof(Type)`, `Slab_size`, `Max_free`, and `Alloc` parameters) together and share the overhead for the slab caches among all equal-sized objects.

Definition at line 465 of file `slab_alloc`.

16.65.2 Member Function Documentation

16.65.2.1 `alloc()`

```
template<typename Type, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A >
class Alloc = New_allocator>
Type * cxx::Slab_static< Type, Slab_size, Max_free, Alloc >::alloc () [inline], [noexcept]
```

Allocate an object of type `Type`.

Returns

A pointer to the just allocated object, or 0 on failure.

Note

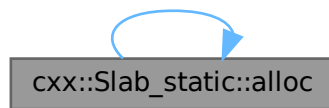
The object is not zeroed out by the slab allocator.

Definition at line 478 of file [slab_alloc](#).

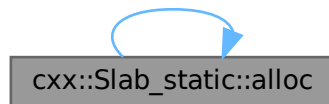
References [alloc\(\)](#).

Referenced by [alloc\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

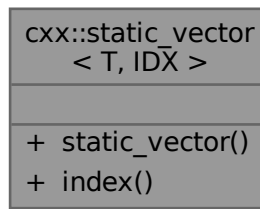
- `I4/cxx/slab_alloc`

16.66 `cxx::static_vector< T, IDX >` Class Template Reference

Simple encapsulation for a dynamically allocated array.

```
#include <static_vector>
```

Collaboration diagram for `cxx::static_vector< T, IDX >`:



Public Member Functions

- `template<typename X, typename = enable_if_t<is_convertible<X, T>::value>>`
static_vector ([static_vector](#)< X, IDX > const &o)
Conversion from compatible arrays.
- `index_type` **index** (value_type const *o) const
Get the index of the given element of the array.

16.66.1 Detailed Description

```
template<typename T, typename IDX = unsigned>
class cxx::static_vector< T, IDX >
```

Simple encapsulation for a dynamically allocated array.

The main purpose of this class is to support C++11 range for for simple dynamically allocated array with static size.

Definition at line 16 of file [static_vector](#).

The documentation for this class was generated from the following file:

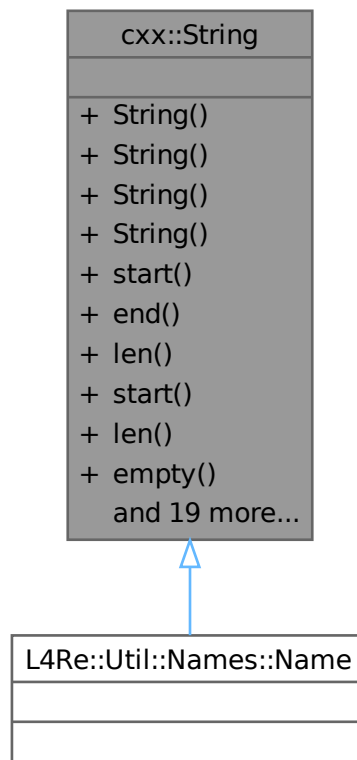
- `I4/cxx/static_vector`

16.67 cxx::String Class Reference

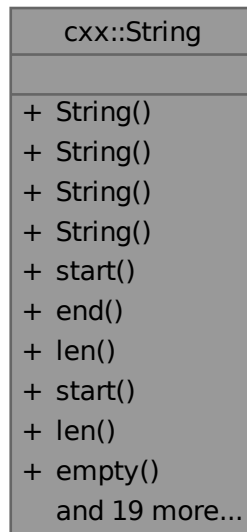
Allocation free string class with explicit length field.

```
#include <string>
```

Inheritance diagram for cxx::String:



Collaboration diagram for cxx::String:



Public Types

- typedef char const * **Index**
Character index type.

Public Member Functions

- **String** (char const *s) noexcept
Initialize from a zero-terminated string.
- **String** (char const *s, unsigned long len) noexcept
Initialize from a pointer to first character and a length.
- **String** (char const *s, char const *e) noexcept
Initialize with start and end pointer.
- **String** ()
Zero-initialize. Create an invalid string.
- **Index start** () const
Pointer to first character.
- **Index end** () const
Pointer to first byte behind the string.
- int **len** () const
Length.
- void **start** (char const *s)
Set start.
- void **len** (unsigned long len)
Set length.
- bool **empty** () const

- Check if the string has length zero.*
- **String head** (**Index end**) const
Return prefix up to index.
- **String head** (unsigned long **end**) const
*Prefix of length **end**.*
- **String substr** (unsigned long idx, unsigned long **len**=~0UL) const
*Substring of length **len** starting at **idx**.*
- **String substr** (char const ***start**, unsigned long **len**=0) const
*Substring of length **len** starting at **start**.*
- template<typename F>
char const * **find_match** (F &&match) const
*Find matching character. **match** should be a function such as **isspace**.*
- char const * **find** (char const *c) const
*Find character. Return **end()** if not found.*
- char const * **find** (int c) const
*Find character. Return **end()** if not found.*
- char const * **rfind** (char const *c) const
*Find right-most character. Return **end()** if not found.*
- **Index starts_with** (cxx::String const &c) const
*Check if **c** is a prefix of string.*
- char const * **find** (int c, char const *s) const
*Find character **c** starting at position **s**. Return **end()** if not found.*
- char const * **find** (char const *c, char const *s) const
Find character set at position.
- char const & **operator[]** (unsigned long idx) const
*Get character at **idx**.*
- char const & **operator[]** (int idx) const
*Get character at **idx**.*
- char const & **operator[]** (**Index idx**) const
*Get character at **idx**.*
- bool **eof** (char const *s) const
*Check if pointer **s** points behind string.*
- template<typename INT>
int **from_dec** (INT *v) const
Convert decimal string to integer.
- template<typename INT>
int **from_hex** (INT *v) const
Convert hex string to integer.
- bool **operator==** (String const &o) const
Equality.
- bool **operator!=** (String const &o) const
Inequality.

16.67.1 Detailed Description

Allocation free string class with explicit length field.

This class is used to group characters of a string which belong to one syntactical token types number, identifier, string, whitespace or another single character.

Stings in this class can contain null bytes and may denote parts of other strings.

Examples

[tmpfs/lib/src/fs.cc](#).

Definition at line 30 of file [string](#).

16.67.2 Constructor & Destructor Documentation

16.67.2.1 String()

```
cxx::String::String (  
    char const * s,  
    char const * e) [inline], [noexcept]
```

Initialize with start and end pointer.

Parameters

<i>s</i>	first character of the string
<i>e</i>	pointer to first byte behind the string

Definition at line 48 of file [string](#).

16.67.3 Member Function Documentation

16.67.3.1 find()

```
char const * cxx::String::find (  
    char const * c,  
    char const * s) const [inline]
```

Find character set at position.

Parameters

<i>c</i>	zero-terminated string of characters to search for
<i>s</i>	start position of search in string

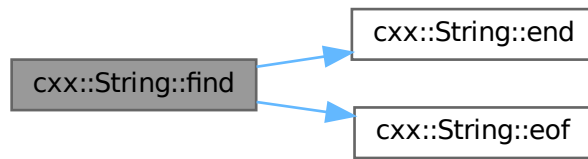
Return values

end()	if no char in <i>c</i> is contained in string at or behind <i>s</i> .
<i>position</i>	in string of some character in <i>c</i> .

Definition at line 191 of file [string](#).

References [end\(\)](#), and [eof\(\)](#).

Here is the call graph for this function:



16.67.3.2 from_dec()

```

template<typename INT>
int cxx::String::from_dec (
    INT * v) const [inline]
  
```

Convert decimal string to integer.

Template Parameters

<i>INT</i>	result integer type
------------	---------------------

Parameters

out	v	conversion result
-----	---	-------------------

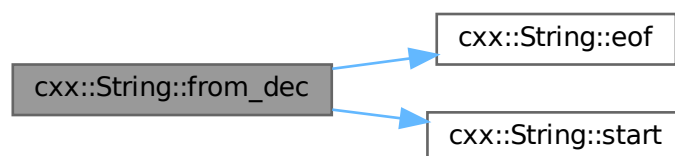
Returns

position of first character not converted.

Definition at line 228 of file [string](#).

References [eof\(\)](#), and [start\(\)](#).

Here is the call graph for this function:



16.67.3.3 from_hex()

```
template<typename INT>
int cxx::String::from_hex (
    INT * v) const [inline]
```

Convert hex string to integer.

Template Parameters

<i>INT</i>	result integer type
------------	---------------------

Parameters

out	<i>v</i>	conversion result
-----	----------	-------------------

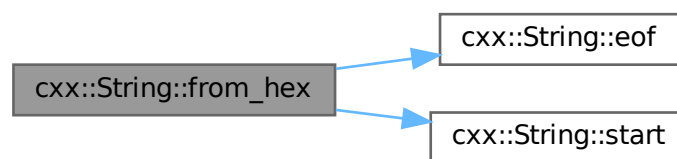
Return values

<i>-1</i>	if the maximal amount of digits fitting into <i>INT</i> have been read,
<i>position</i>	of first character not converted otherwise.

Definition at line [257](#) of file [string](#).

References [eof\(\)](#), and [start\(\)](#).

Here is the call graph for this function:



16.67.3.4 starts_with()

```
Index cxx::String::starts_with (
    cxx::String const & c) const [inline]
```

Check if *c* is a prefix of string.

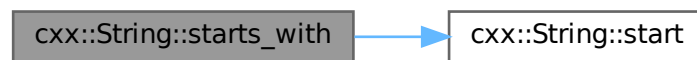
Returns

0 if `c` is not a prefix, if it is a prefix, return first position not in `c` (which might be [end\(\)](#)).

Definition at line [155](#) of file [string](#).

References [start\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

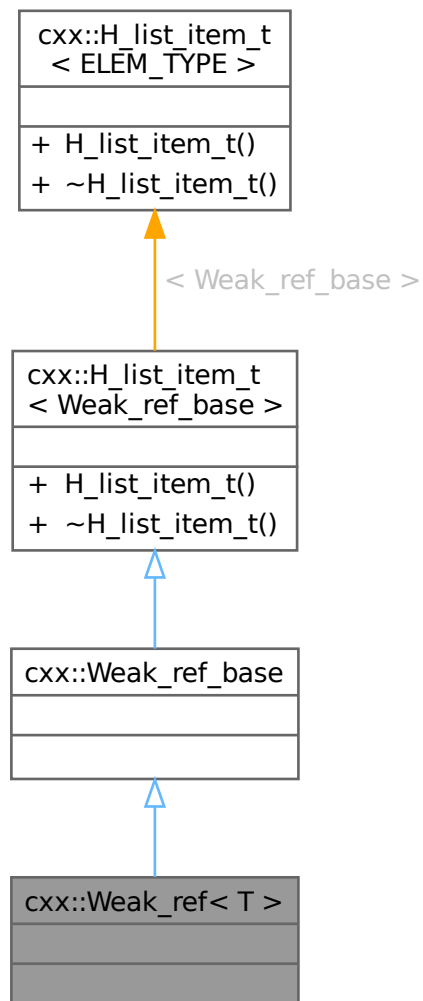
- `l4/cxx/string`

16.68 `cxx::Weak_ref< T >` Class Template Reference

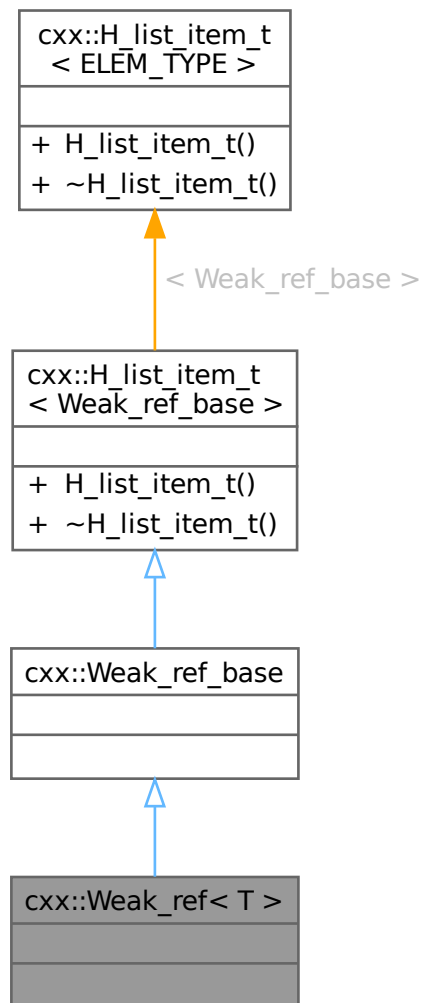
Typed weak reference to an object of type `T`.

```
#include <weak_ref>
```

Inheritance diagram for cxx::Weak_ref< T >:



Collaboration diagram for `cxx::Weak_ref< T >`:



Additional Inherited Members

Public Member Functions inherited from `cxx::H_list_item_t< Weak_ref_base >`

- `H_list_item_t()`
Constructor.
- `~H_list_item_t()` noexcept
Destructor.

16.68.1 Detailed Description

```
template<typename T>
class cxx::Weak_ref< T >
```

Typed weak reference to an object of type `T`.

Template Parameters

<i>T</i>	The type of the referenced object.
----------	------------------------------------

A weak reference is a reference that is invalidated when the referenced object is about to be deleted. All weak references to an object are kept in a linked list (see [Weak_ref_base::List](#)) and all the weak references are iterated and reset by the [Weak_ref_base::List](#) destructor or `Weak_ref_base::List::reset()`.

The type `T` must provide two methods that handle the housekeeping of weak references: `remove_weak_ref(Weak_ref_base *)` and `add_weak_ref(Weak_ref_base *)`. These functions must handle the insertion and removal of the weak reference into the respective [Weak_ref_base::List](#) object. For convenience one can use the `cxx::Weak_ref_obj` as a base class that handles weak references for you.

For example:

```
class C : public cxx::Weak_ref_obj {};

int main()
{
    cxx::Weak_ref<C> r; // r is nullptr
    {
        C c;
        r = &c; // now r points to c
    } // c is destructed, which implies resetting all weak references to c
    // now r is nullptr
    return 0;
}
```

Note

Weak references have no effect on the lifetime of the referenced object. Hence, a referenced object is *not* deleted when all weak references for it are gone. If automatic deletion is needed, see [cxx::Ref_ptr](#).

Definition at line 95 of file [weak_ref](#).

The documentation for this class was generated from the following file:

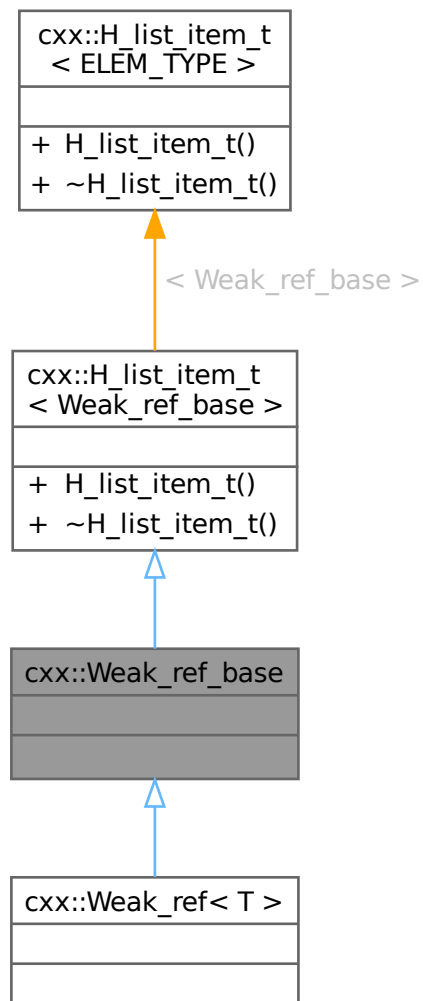
- `I4/cxx/weak_ref`

16.69 cxx::Weak_ref_base Class Reference

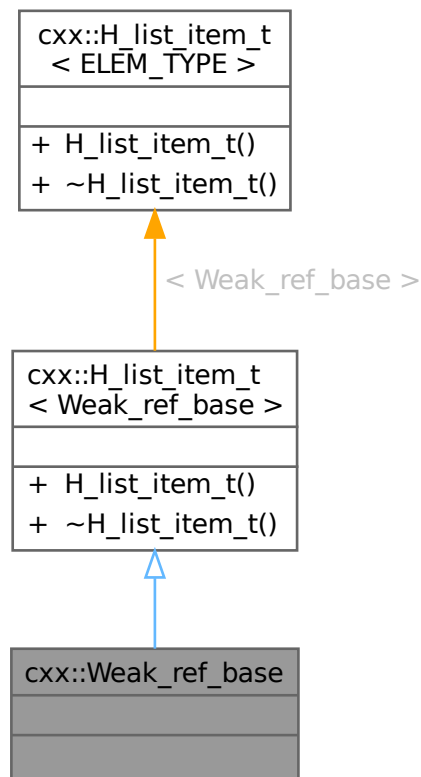
Generic (base) weak reference to some object.

```
#include <weak_ref>
```


Inheritance diagram for cxx::Weak_ref_base:



Collaboration diagram for `cxx::Weak_ref_base`:



Data Structures

- struct [List](#)

The list type for keeping all weak references to an object.

Additional Inherited Members

Public Member Functions inherited from `cxx::H_list_item_t< Weak_ref_base >`

- [H_list_item_t\(\)](#)
Constructor.
- [~H_list_item_t\(\)](#) noexcept
Destructor.

16.69.1 Detailed Description

Generic (base) weak reference to some object.

A weak reference is a reference that gets reset to NULL when the object shall be deleted. All weak references to the same object are kept in a linked list of weak references.

For typed weak references see [cxx::Weak_ref](#).

Definition at line 24 of file [weak_ref](#).

The documentation for this class was generated from the following file:

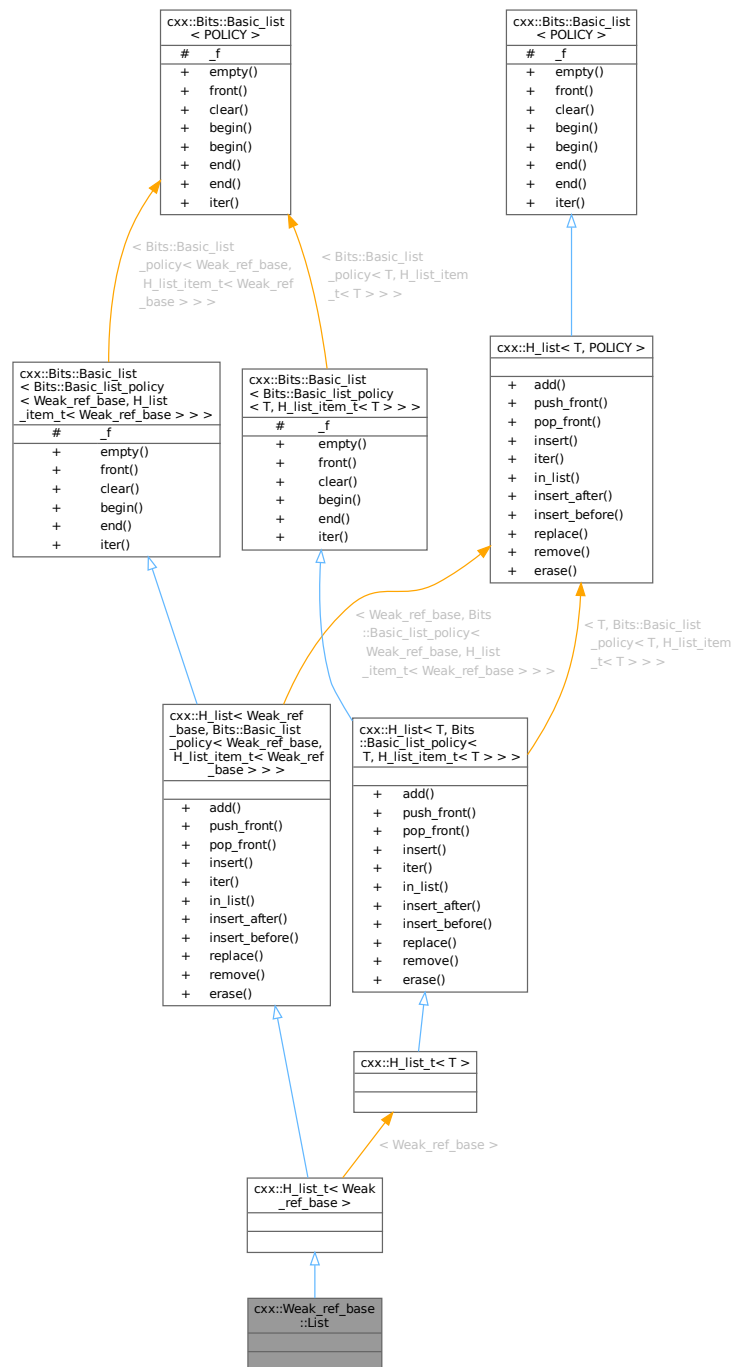
- `I4/cxx/weak_ref`

16.70 `cxx::Weak_ref_base::List` Struct Reference

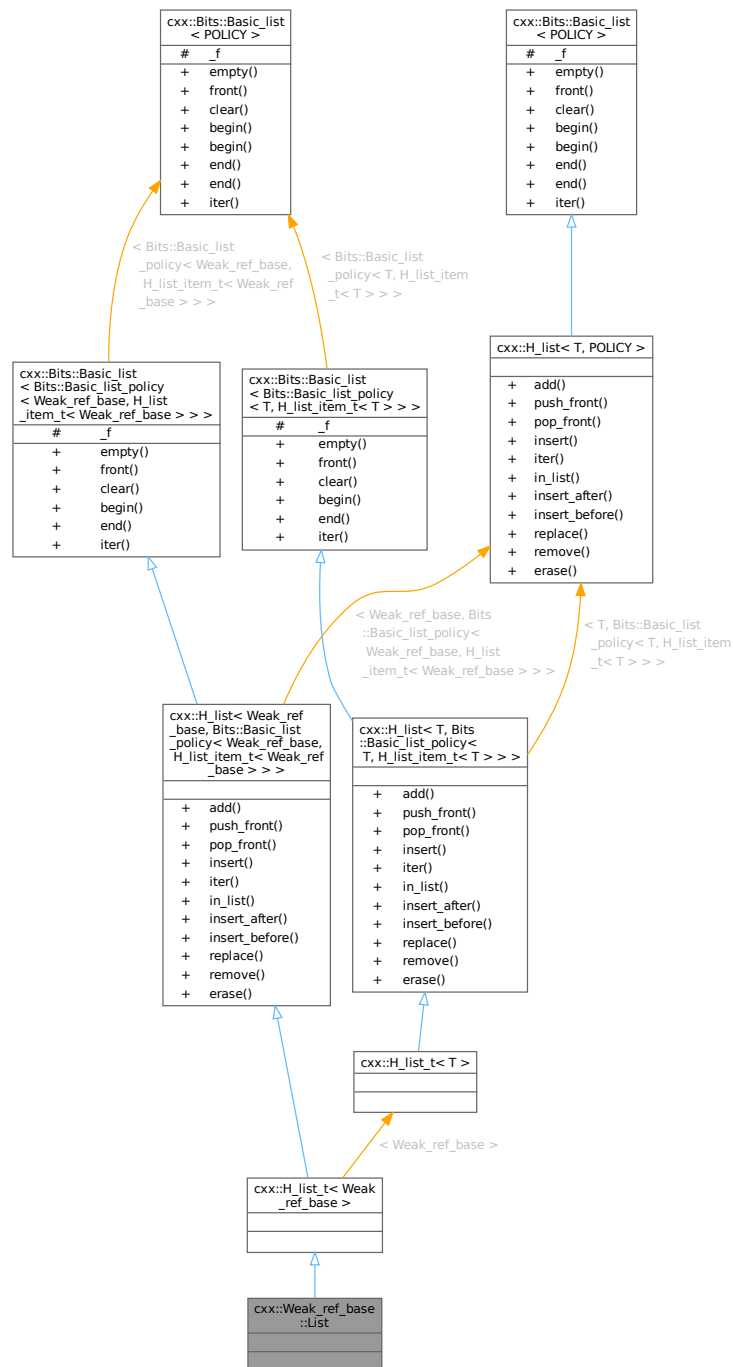
The list type for keeping all weak references to an object.

```
#include <weak_ref>
```

Inheritance diagram for `cxx::Weak_ref_base::List`:



Collaboration diagram for cxx::Weak_ref_base::List:



Additional Inherited Members

Public Member Functions inherited from

`cxx::H_list< Weak_ref_base, Bits::Basic_list_policy< Weak_ref_base, H_list_item_t< Weak_ref_base >`

- void `add` (`Weak_ref_base *e`)

- *Add element to the front of the list.*
- void **push_front** ([Weak_ref_base](#) *e)
Add element to the front of the list.
- [Weak_ref_base](#) * **pop_front** ()
Remove and return the head element of the list.
- Iterator **insert** ([Weak_ref_base](#) *e, Iterator const &pred)
Insert an element at the iterator position.

Public Member Functions inherited from

[cxx::Bits::Basic_list](#)< [Bits::Basic_list_policy](#)< [Weak_ref_base](#), [H_list_item_t](#)< [Weak_ref_base](#) > > >

- bool **empty** () const
Check if the list is empty.
- Value_type **front** () const
Return the first element in the list.
- void **clear** ()
Remove all elements from the list.
- Iterator **begin** ()
Return an iterator to the beginning of the list.
- Const_iterator **end** () const
Return a const iterator to the end of the list.

Static Public Member Functions inherited from

[cxx::H_list](#)< [Weak_ref_base](#), [Bits::Basic_list_policy](#)< [Weak_ref_base](#), [H_list_item_t](#)< [Weak_ref_base](#) > > >

- static Iterator **iter** ([Weak_ref_base](#) *c)
Return an iterator for an arbitrary list element.
- static bool **in_list** ([Weak_ref_base](#) const *e)
Check if the given element is currently part of a list.
- static Iterator **insert_after** ([Weak_ref_base](#) *e, Iterator const &pred)
Insert an element after the iterator position.
- static void **insert_before** ([Weak_ref_base](#) *e, Iterator const &succ)
Insert an element before the iterator position.
- static void **replace** ([Weak_ref_base](#) *p, [Weak_ref_base](#) *e)
Replace an element in a list with a new element.
- static void **remove** ([Weak_ref_base](#) *e)
Remove the given element from its list.
- static Iterator **erase** (Iterator const &e)
Remove the element at the given iterator position.

Static Public Member Functions inherited from

[cxx::Bits::Basic_list](#)< [Bits::Basic_list_policy](#)< [Weak_ref_base](#), [H_list_item_t](#)< [Weak_ref_base](#) > > >

- static Const_iterator **iter** (Const_value_type c)
Return a const iterator that begins at the given element.

Protected Attributes inherited from

[cxx::Bits::Basic_list](#)< [Bits::Basic_list_policy](#)< [Weak_ref_base](#), [H_list_item_t](#)< [Weak_ref_base](#) > > >

- [Bits::Basic_list_policy](#)< [Weak_ref_base](#), [H_list_item_t](#)< [Weak_ref_base](#) > >::Head_type_f
Pointer to front of the list.

16.70.1 Detailed Description

The list type for keeping all weak references to an object.

On destruction of a list, all weak references to the respective object are set to `nullptr`.

Definition at line 38 of file [weak_ref](#).

The documentation for this struct was generated from the following file:

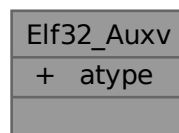
- `l4/cxx/weak_ref`

16.71 Elf32_Auxv Struct Reference

Auxiliary vector (32-bit).

```
#include <elf.h>
```

Collaboration diagram for Elf32_Auxv:

**Data Fields**

- [Elf32_Word](#) atype

16.71.1 Detailed Description

Auxiliary vector (32-bit).

Definition at line 962 of file [elf.h](#).

16.71.2 Field Documentation

16.71.2.1 atype

[Elf32_Word](#) `Elf32_Auxv::atype`

See also

[Elf_ATs](#)

Definition at line [964](#) of file [elf.h](#).

The documentation for this struct was generated from the following file:

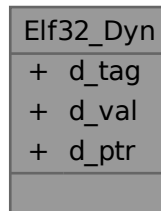
- [l4/util/elf.h](#)

16.72 Elf32_Dyn Struct Reference

ELF32 dynamic entry.

```
#include <elf.h>
```

Collaboration diagram for Elf32_Dyn:



Data Fields

- [Elf32_Sword](#) `d_tag`

16.72.1 Detailed Description

ELF32 dynamic entry.

Definition at line [513](#) of file [elf.h](#).

16.72.2 Field Documentation

16.72.2.1 d_tag

[Elf32_Sword](#) `Elf32_Dyn::d_tag`

See also

[Elf_DTs](#)

Definition at line 515 of file [elf.h](#).

The documentation for this struct was generated from the following file:

- [l4/util/elf.h](#)

16.73 Elf32_Ehdr Struct Reference

ELF32 header.

```
#include <elf.h>
```

Collaboration diagram for Elf32_Ehdr:

Elf32_Ehdr
+ e_ident
+ e_type
+ e_machine
+ e_version
+ e_entry
+ e_phoff
+ e_shoff
+ e_flags
+ e_ehsize
+ e_phentsize
+ e_phnum
+ e_shentsize
+ e_shnum
+ e_shstrndx

Data Fields

- unsigned char **e_ident** [[EI_NIDENT](#)]
see [Elf_EI](#)s
- [Elf32_Half](#) **e_type**
type of ELF file
- [Elf32_Half](#) **e_machine**
required architecture
- [Elf32_Word](#) **e_version**
file version
- [Elf32_Addr](#) **e_entry**
initial program counter
- [Elf32_Off](#) **e_phoff**
offset of program header table
- [Elf32_Off](#) **e_shoff**
offset of file header table
- [Elf32_Word](#) **e_flags**
processor-specific flags
- [Elf32_Half](#) **e_ehsize**
size of ELF header
- [Elf32_Half](#) **e_phentsize**
size of program header entry
- [Elf32_Half](#) **e_phnum**
number of entries in program header table
- [Elf32_Half](#) **e_shentsize**
size of section header entry
- [Elf32_Half](#) **e_shnum**
number of entries in section header table
- [Elf32_Half](#) **e_shstrndx**
section header table index of strtab

16.73.1 Detailed Description

ELF32 header.

Definition at line [125](#) of file [elf.h](#).

16.73.2 Field Documentation

16.73.2.1 e_flags

[Elf32_Word](#) [Elf32_Ehdr::e_flags](#)

processor-specific flags

See also

[Elf_EF_ARM_s](#)

Definition at line [134](#) of file [elf.h](#).

16.73.2.2 e_machine

`Elf32_Half Elf32_Ehdr::e_machine`

required architecture

See also

[Elf_EMs](#)

Definition at line 129 of file [elf.h](#).

16.73.2.3 e_type

`Elf32_Half Elf32_Ehdr::e_type`

type of ELF file

See also

[Elf_ETs](#)

Definition at line 128 of file [elf.h](#).

16.73.2.4 e_version

`Elf32_Word Elf32_Ehdr::e_version`

file version

See also

[Elf_EVs](#)

Definition at line 130 of file [elf.h](#).

The documentation for this struct was generated from the following file:

- [l4/util/elf.h](#)

16.74 Elf32_Phdr Struct Reference

ELF32 program header.

```
#include <elf.h>
```

Collaboration diagram for Elf32_Phdr:

Elf32_Phdr
+ p_type
+ p_offset
+ p_vaddr
+ p_paddr
+ p_filesz
+ p_memsz
+ p_flags
+ p_align

Data Fields

- [Elf32_Word p_type](#)
type of program section
- [Elf32_Off p_offset](#)
file offset of program section
- [Elf32_Addr p_vaddr](#)
memory address of prog section
- [Elf32_Addr p_paddr](#)
physical address (ignored)
- [Elf32_Word p_filesz](#)
file size of program section
- [Elf32_Word p_memsz](#)
memory size of program section
- [Elf32_Word p_flags](#)
flags
- [Elf32_Word p_align](#)
alignment of section

16.74.1 Detailed Description

ELF32 program header.

Definition at line 425 of file [elf.h](#).

16.74.2 Field Documentation

16.74.2.1 p_flags

`Elf32_Word Elf32_Phdr::p_flags`

flags

See also

[Elf_PFs](#)

Definition at line [433](#) of file [elf.h](#).

16.74.2.2 p_type

`Elf32_Word Elf32_Phdr::p_type`

type of program section

See also

[Elf_PTs](#)

Definition at line [427](#) of file [elf.h](#).

The documentation for this struct was generated from the following file:

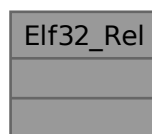
- [l4/util/elf.h](#)

16.75 Elf32_Rel Struct Reference

ELF32 relocation entry w/o addend.

```
#include <elf.h>
```

Collaboration diagram for Elf32_Rel:



16.75.1 Detailed Description

ELF32 relocation entry w/o addend.

Definition at line 631 of file [elf.h](#).

The documentation for this struct was generated from the following file:

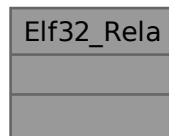
- [l4/util/elf.h](#)

16.76 Elf32_Rela Struct Reference

ELF32 relocation entry w/ addend.

```
#include <elf.h>
```

Collaboration diagram for Elf32_Rela:



16.76.1 Detailed Description

ELF32 relocation entry w/ addend.

Definition at line 638 of file [elf.h](#).

The documentation for this struct was generated from the following file:

- [l4/util/elf.h](#)

16.77 Elf32_Shdr Struct Reference

ELF32 section header.

```
#include <elf.h>
```

Collaboration diagram for Elf32_Shdr:

Elf32_Shdr
+ sh_name
+ sh_type
+ sh_flags
+ sh_addr
+ sh_offset
+ sh_size
+ sh_link
+ sh_info
+ sh_addralign
+ sh_entsize

Data Fields

- [Elf32_Word](#) **sh_name**
name of sect (idx into strtab)
- [Elf32_Word](#) **sh_type**
section's type
- [Elf32_Word](#) **sh_flags**
section's flags
- [Elf32_Addr](#) **sh_addr**
memory address of section
- [Elf32_Off](#) **sh_offset**
file offset of section
- [Elf32_Word](#) **sh_size**
file size of section
- [Elf32_Word](#) **sh_link**
idx to associated header section
- [Elf32_Word](#) **sh_info**
extra info of header section
- [Elf32_Word](#) **sh_addralign**
address alignment constraints
- [Elf32_Word](#) **sh_entsize**
size of entry if sect is table

16.77.1 Detailed Description

ELF32 section header.

Definition at line 347 of file [elf.h](#).

16.77.2 Field Documentation

16.77.2.1 sh_flags

[Elf32_Word](#) `Elf32_Shdr::sh_flags`

section's flags

See also

[Elf_SHFs](#)

Definition at line 351 of file [elf.h](#).

16.77.2.2 sh_type

[Elf32_Word](#) `Elf32_Shdr::sh_type`

section's type

See also

[Elf_SHTs](#)

Definition at line 350 of file [elf.h](#).

The documentation for this struct was generated from the following file:

- [l4/util/elf.h](#)

16.78 Elf32_Sym Struct Reference

ELF32 symbol table entry.

```
#include <elf.h>
```

Collaboration diagram for `Elf32_Sym`:

Elf32_Sym
+ st_name
+ st_value
+ st_size
+ st_info
+ st_other
+ st_shndx

Data Fields

- [Elf32_Word](#) **st_name**
name of symbol (idx symstrtab)
- [Elf32_Addr](#) **st_value**
value of associated symbol
- [Elf32_Word](#) **st_size**
size of associated symbol
- unsigned char **st_info**
type and binding info
- unsigned char **st_other**
undefined
- [Elf32_Half](#) **st_shndx**
associated section header

16.78.1 Detailed Description

ELF32 symbol table entry.

Definition at line 871 of file [elf.h](#).

The documentation for this struct was generated from the following file:

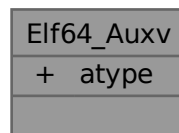
- [l4/util/elf.h](#)

16.79 Elf64_Auxv Struct Reference

Auxiliary vector (64-bit).

```
#include <elf.h>
```

Collaboration diagram for Elf64_Auxv:



Data Fields

- [Elf64_Word](#) **atype**

16.79.1 Detailed Description

Auxiliary vector (64-bit).

Definition at line 969 of file [elf.h](#).

16.79.2 Field Documentation

16.79.2.1 atype

[Elf64_Word](#) [Elf64_Auxv::atype](#)

See also

[Elf_ATs](#)

Definition at line 971 of file [elf.h](#).

The documentation for this struct was generated from the following file:

- [l4/util/elf.h](#)

16.80 Elf64_Dyn Struct Reference

ELF64 dynamic entry.

```
#include <elf.h>
```

Collaboration diagram for Elf64_Dyn:



Data Fields

- [Elf64_Sxword](#) `d_tag`

16.80.1 Detailed Description

ELF64 dynamic entry.

Definition at line 524 of file [elf.h](#).

16.80.2 Field Documentation

16.80.2.1 d_tag

[Elf64_Sxword](#) Elf64_Dyn::d_tag

See also

[Elf_DTs](#)

Definition at line 526 of file [elf.h](#).

The documentation for this struct was generated from the following file:

- [l4/util/elf.h](#)

16.81 Elf64_Ehdr Struct Reference

ELF64 header.

```
#include <elf.h>
```

Collaboration diagram for Elf64_Ehdr:

Elf64_Ehdr
+ e_ident
+ e_type
+ e_machine
+ e_version
+ e_entry
+ e_phoff
+ e_shoff
+ e_flags
+ e_ehsize
+ e_phentsize
+ e_phnum
+ e_shentsize
+ e_shnum
+ e_shstrndx

Data Fields

- unsigned char **e_ident** [[EI_NIDENT](#)]
see [Elf_EI](#)s
- [Elf64_Half](#) **e_type**
type of ELF file
- [Elf64_Half](#) **e_machine**
required architecture
- [Elf64_Word](#) **e_version**
file version
- [Elf64_Addr](#) **e_entry**
initial program counter
- [Elf64_Off](#) **e_phoff**
offset of program header table
- [Elf64_Off](#) **e_shoff**
offset of file header table
- [Elf64_Word](#) **e_flags**
processor-specific flags
- [Elf64_Half](#) **e_ehsize**
size of ELF header
- [Elf64_Half](#) **e_phentsize**
size of program header entry
- [Elf64_Half](#) **e_phnum**
number of entries in program header table
- [Elf64_Half](#) **e_shentsize**
size of section header entry
- [Elf64_Half](#) **e_shnum**
number of entries in section header table
- [Elf64_Half](#) **e_shstrndx**
section header table index of strtab

16.81.1 Detailed Description

ELF64 header.

Definition at line [146](#) of file [elf.h](#).

16.81.2 Field Documentation

16.81.2.1 e_flags

[Elf64_Word](#) [Elf64_Ehdr::e_flags](#)

processor-specific flags

See also

[Elf_EF_ARM_s](#)

Definition at line [155](#) of file [elf.h](#).

16.81.2.2 e_machine

`Elf64_Half Elf64_Ehdr::e_machine`

required architecture

See also

[Elf_EMs](#)

Definition at line 150 of file [elf.h](#).

16.81.2.3 e_type

`Elf64_Half Elf64_Ehdr::e_type`

type of ELF file

See also

[Elf_ETs](#)

Definition at line 149 of file [elf.h](#).

16.81.2.4 e_version

`Elf64_Word Elf64_Ehdr::e_version`

file version

See also

[Elf_EVs](#)

Definition at line 151 of file [elf.h](#).

The documentation for this struct was generated from the following file:

- [l4/util/elf.h](#)

16.82 Elf64_Phdr Struct Reference

ELF64 program header.

```
#include <elf.h>
```

Collaboration diagram for Elf64_Phdr:

Elf64_Phdr
+ p_type
+ p_flags
+ p_offset
+ p_vaddr
+ p_paddr
+ p_filesz
+ p_memsz
+ p_align

Data Fields

- [Elf64_Word p_type](#)
type of program section
- [Elf64_Word p_flags](#)
flags
- [Elf64_Off p_offset](#)
file offset of program section
- [Elf64_Addr p_vaddr](#)
memory address of prog section
- [Elf64_Addr p_paddr](#)
physical address (ignored)
- [Elf64_Xword p_filesz](#)
file size of program section
- [Elf64_Xword p_memsz](#)
memory size of program section
- [Elf64_Xword p_align](#)
alignment of section

16.82.1 Detailed Description

ELF64 program header.

Definition at line 438 of file [elf.h](#).

16.82.2 Field Documentation

16.82.2.1 p_flags

`Elf64_Word Elf64_Phdr::p_flags`

flags

See also

[Elf_PFs](#)

Definition at line [441](#) of file [elf.h](#).

16.82.2.2 p_type

`Elf64_Word Elf64_Phdr::p_type`

type of program section

See also

[Elf_PTs](#)

Definition at line [440](#) of file [elf.h](#).

The documentation for this struct was generated from the following file:

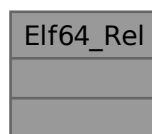
- [l4/util/elf.h](#)

16.83 Elf64_Rel Struct Reference

ELF64 relocation entry w/o addend.

```
#include <elf.h>
```

Collaboration diagram for Elf64_Rel:



16.83.1 Detailed Description

ELF64 relocation entry w/o addend.

Definition at line 646 of file [elf.h](#).

The documentation for this struct was generated from the following file:

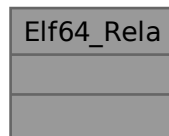
- [l4/util/elf.h](#)

16.84 Elf64_Rela Struct Reference

ELF64 relocation entry w/ addend.

```
#include <elf.h>
```

Collaboration diagram for Elf64_Rela:



16.84.1 Detailed Description

ELF64 relocation entry w/ addend.

Definition at line 653 of file [elf.h](#).

The documentation for this struct was generated from the following file:

- [l4/util/elf.h](#)

16.85 Elf64_Shdr Struct Reference

ELF64 section header.

```
#include <elf.h>
```

Collaboration diagram for Elf64_Shdr:

Elf64_Shdr
+ sh_name
+ sh_type
+ sh_flags
+ sh_addr
+ sh_offset
+ sh_size
+ sh_link
+ sh_info
+ sh_addralign
+ sh_entsize

Data Fields

- [Elf64_Word](#) **sh_name**
name of sect (idx into strtab)
- [Elf64_Word](#) **sh_type**
section's type
- [Elf64_Xword](#) **sh_flags**
section's flags
- [Elf64_Addr](#) **sh_addr**
memory address of section
- [Elf64_Off](#) **sh_offset**
file offset of section
- [Elf64_Xword](#) **sh_size**
file size of section
- [Elf64_Word](#) **sh_link**
idx to associated header section
- [Elf64_Word](#) **sh_info**
extra info of header section
- [Elf64_Xword](#) **sh_addralign**
address alignment constraints
- [Elf64_Xword](#) **sh_entsize**
size of entry if sect is table

16.85.1 Detailed Description

ELF64 section header.

Definition at line 362 of file [elf.h](#).

16.85.2 Field Documentation

16.85.2.1 sh_flags

[Elf64_Xword](#) `Elf64_Shdr::sh_flags`

section's flags

See also

[Elf_SHFs](#)

Definition at line 366 of file [elf.h](#).

16.85.2.2 sh_type

[Elf64_Word](#) `Elf64_Shdr::sh_type`

section's type

See also

[Elf_SHTs](#)

Definition at line 365 of file [elf.h](#).

The documentation for this struct was generated from the following file:

- [l4/util/elf.h](#)

16.86 Elf64_Sym Struct Reference

ELF64 symbol table entry.

```
#include <elf.h>
```

Collaboration diagram for `Elf64_Sym`:

Elf64_Sym
+ st_name
+ st_info
+ st_other
+ st_shndx
+ st_value
+ st_size

Data Fields

- [Elf64_Word](#) **st_name**
name of symbol (idx symstrtab)
- unsigned char **st_info**
type and binding info
- unsigned char **st_other**
undefined
- [Elf64_Half](#) **st_shndx**
associated section header
- [Elf64_Addr](#) **st_value**
value of associated symbol
- [Elf64_Xword](#) **st_size**
size of associated symbol

16.86.1 Detailed Description

ELF64 symbol table entry.

Definition at line 882 of file [elf.h](#).

The documentation for this struct was generated from the following file:

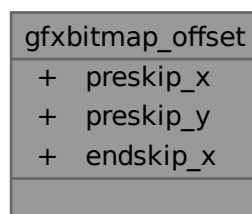
- [l4/util/elf.h](#)

16.87 gfxbitmap_offset Struct Reference

offsets in pmap[] and bmap[]

```
#include <bitmap.h>
```

Collaboration diagram for gfxbitmap_offset:



Data Fields

- [l4_uint32_t](#) **preskip_x**
skip pixels at beginning of line
- [l4_uint32_t](#) **preskip_y**
skip lines
- [l4_uint32_t](#) **endskip_x**
skip pixels at end of line

16.87.1 Detailed Description

offsets in `pmap[]` and `bmap[]`

Definition at line 68 of file [bitmap.h](#).

The documentation for this struct was generated from the following file:

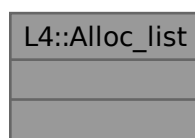
- [l4/libgfxbitmap/bitmap.h](#)

16.88 L4::Alloc_list Class Reference

A simple list-based allocator.

```
#include <alloc.h>
```

Collaboration diagram for L4::Alloc_list:



16.88.1 Detailed Description

A simple list-based allocator.

Definition at line 20 of file [alloc.h](#).

The documentation for this class was generated from the following file:

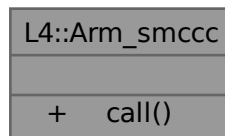
- [l4/cxx/alloc.h](#)

16.89 L4::Arm_smccc Class Reference

Wrapper for function calls that follow the ARM SMC/HVC calling convention.

```
#include <arm_smccc>
```

Collaboration diagram for L4::Arm_smccc:



Public Member Functions

- [l4_msgtag_t](#) [call](#) ([l4_umword_t](#) func, [l4_umword_t](#) in0, [l4_umword_t](#) in1, [l4_umword_t](#) in2, [l4_umword_t](#) in3, [l4_umword_t](#) in4, [l4_umword_t](#) in5, [l4_umword_t](#) *out0, [l4_umword_t](#) *out1, [l4_umword_t](#) *out2, [l4_umword_t](#) *out3, [l4_umword_t](#) client_id)

ARM SMC/HVC function call.

16.89.1 Detailed Description

Wrapper for function calls that follow the ARM SMC/HVC calling convention.

See [l4_arm_smccc_call\(\)](#) for the corresponding C interface.

Definition at line 23 of file [arm_smccc](#).

16.89.2 Member Function Documentation

16.89.2.1 call()

```

l4_msgtag_t L4::Arm_smccc::call (
    l4_umword_t func,
    l4_umword_t in0,
    l4_umword_t in1,
    l4_umword_t in2,
    l4_umword_t in3,
    l4_umword_t in4,
    l4_umword_t in5,
    l4_umword_t * out0,
    l4_umword_t * out1,
    l4_umword_t * out2,
    l4_umword_t * out3,
    l4_umword_t client_id)
  
```

ARM SMC/HVC function call.

The input parameters consist of a function identifier, 6 arguments and a client id. Results are returned in 4 output parameters.

Parameters

	<i>func</i>	Function identifier. <ul style="list-style-type: none"> • Bit 31 has to be set: This marks the call as <i>Fast Call</i>. <i>Yielding Calls</i> (bit 31 unset) are rejected by the kernel. • Bit 30 defines the calling convention: • Bit 30 == 1: 64-bit calling convention. • Bit 30 == 0: 32-bit calling convention. • Bits 24..29 determine the service call ID. The permitted IDs are set in the kernel configuration. By default only service IDs $\geq 0x30000000$ (<i>Trusted Application Calls</i> and <i>Trusted OS Calls</i>) are allowed.
in	<i>in0</i>	First input parameter.
in	<i>in1</i>	Second input parameter.
in	<i>in2</i>	Third input parameter.
in	<i>in3</i>	Fourth input parameter.
in	<i>in4</i>	Fifth input parameter.
in	<i>in5</i>	Sixth input parameter.
out	<i>out0</i>	First output parameter.
out	<i>out1</i>	Second output parameter.
out	<i>out2</i>	Third output parameter.
out	<i>out3</i>	Fourth output parameter.
in	<i>client↔ _id</i>	Client ID. According to the specification, this value might be ignored by certain functions.

Return values

<code>-L4_ENOSYS</code>	Either bit 31 of the function call not set or service ID outside the range permitted by kernel configuration.
<code>-L4_EINVAL</code>	Invalid number of parameters.
<code>< 0</code>	Other L4 error.
<code>0</code>	Success.

The documentation for this class was generated from the following file:

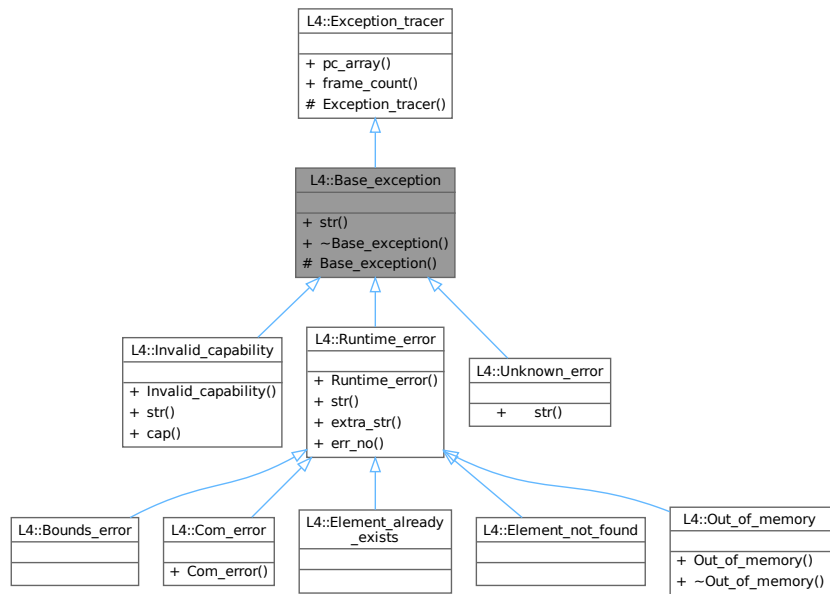
- `l4/sys/arm_smccc`

16.90 L4::Base_exception Class Reference

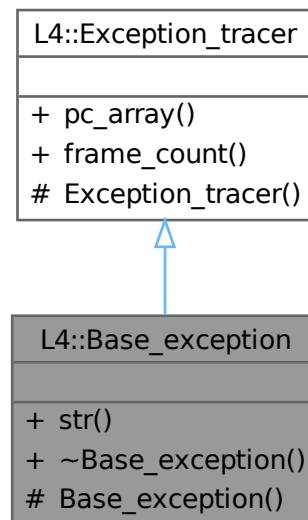
Base class for all exceptions, thrown by the [L4Re](#) framework.

```
#include <l4/cxx/exceptions>
```

Inheritance diagram for L4::Base_exception:



Collaboration diagram for L4::Base_exception:



Public Member Functions

- virtual char const * **str** () const noexcept=0

Return a human readable string for the exception.

- virtual `~Base_exception ()` noexcept

Destruction.

Public Member Functions inherited from [L4::Exception_tracer](#)

- void const *const * `pc_array ()` const noexcept

Get the array containing the call trace.

- int `frame_count ()` const noexcept

Get the number of entries that are valid in the call trace.

Protected Member Functions

- `Base_exception ()` noexcept

Create a base exception.

Protected Member Functions inherited from [L4::Exception_tracer](#)

- `Exception_tracer ()` noexcept

Create a back trace.

16.90.1 Detailed Description

Base class for all exceptions, thrown by the [L4Re](#) framework.

This is the abstract base of all exceptions thrown within the [L4Re](#) framework. It is basically also a good idea to use it as base of all user defined exceptions.

Definition at line [105](#) of file [exceptions](#).

The documentation for this class was generated from the following file:

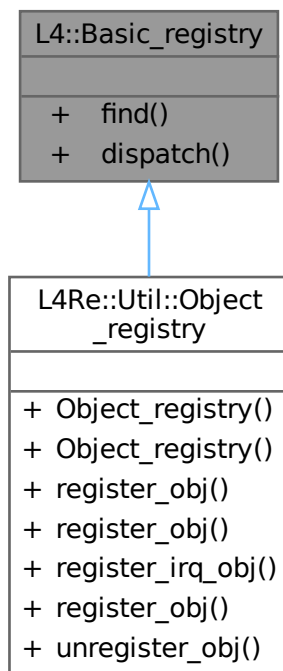
- [l4/cxx/exceptions](#)

16.91 L4::Basic_registry Class Reference

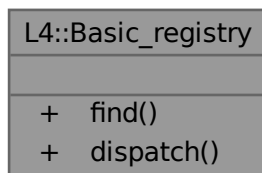
This registry returns the corresponding server object based on the label of an [lpc_gate](#).

```
#include <ipc_epiface>
```

Inheritance diagram for L4::Basic_registry:



Collaboration diagram for L4::Basic_registry:



Static Public Member Functions

- static [Value](#) * [find](#) ([l4_umword_t](#) label)
Get the server object for an [lpc_gate](#) label.
- static [l4_msgtag_t](#) [dispatch](#) ([l4_msgtag_t](#) tag, [l4_umword_t](#) label, [l4_utcb_t](#) *utcb)
The dispatch function called by the server loop.

16.91.1 Detailed Description

This registry returns the corresponding server object based on the label of an [lpc_gate](#).

Definition at line 529 of file [ipc_epiface](#).

16.91.2 Member Function Documentation

16.91.2.1 dispatch()

```
l4\_msgtag\_t L4::Basic_registry::dispatch (
    l4\_msgtag\_t tag,
    l4\_umword\_t label,
    l4\_utcb\_t * utcb) [inline], [static]
```

The dispatch function called by the server loop.

This function forwards the message to the server object identified by the given *label*.

Parameters

<i>tag</i>	The message tag used for the invocation.
<i>label</i>	The label used to find the object including the rights bits of the invoked capability.
<i>utcb</i>	The UTCB used for the invocation.

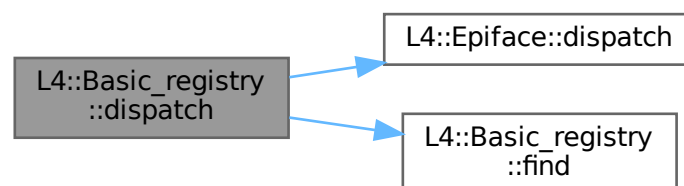
Returns

The return code from the object's dispatch function or -L4_ENOENT if the object does not exist.

Definition at line 554 of file [ipc_epiface](#).

References [L4::Epiface::dispatch\(\)](#), and [find\(\)](#).

Here is the call graph for this function:



16.91.2.2 find()

```
Value * L4::Basic_registry::find (  
    l4_umword_t label) [inline], [static]
```

Get the server object for an [lpc_gate](#) label.

Parameters

<i>label</i>	The label usually stored in an lpc_gate .
--------------	---

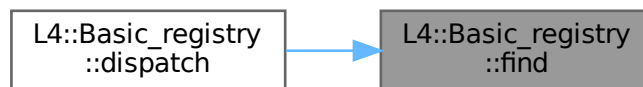
Returns

A pointer to the [Epiface](#) identified by the given label.

Definition at line 538 of file [ipc_epiface](#).

Referenced by [dispatch\(\)](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

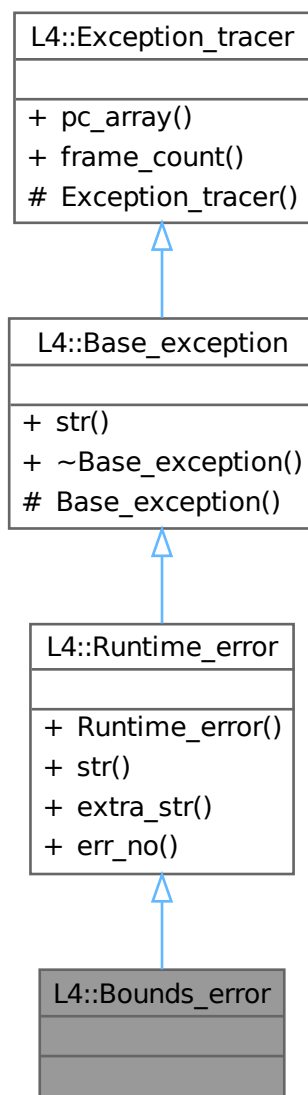
- `l4/sys/cxx/ipc_epiface`

16.92 L4::Bounds_error Class Reference

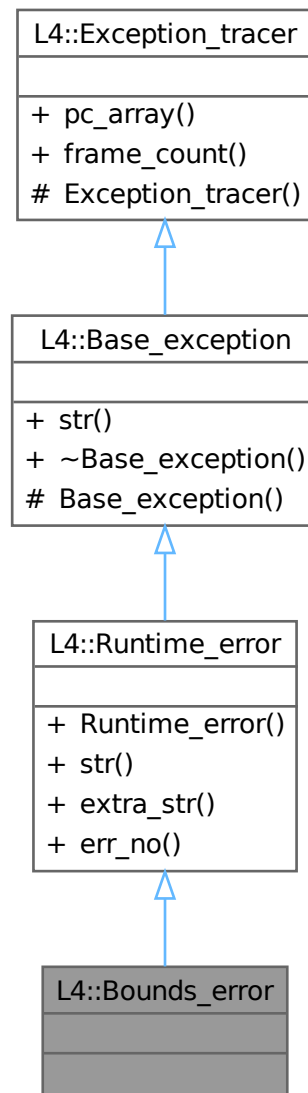
Access out of bounds.

```
#include <exceptions>
```

Inheritance diagram for L4::Bounds_error:



Collaboration diagram for L4::Bounds_error:

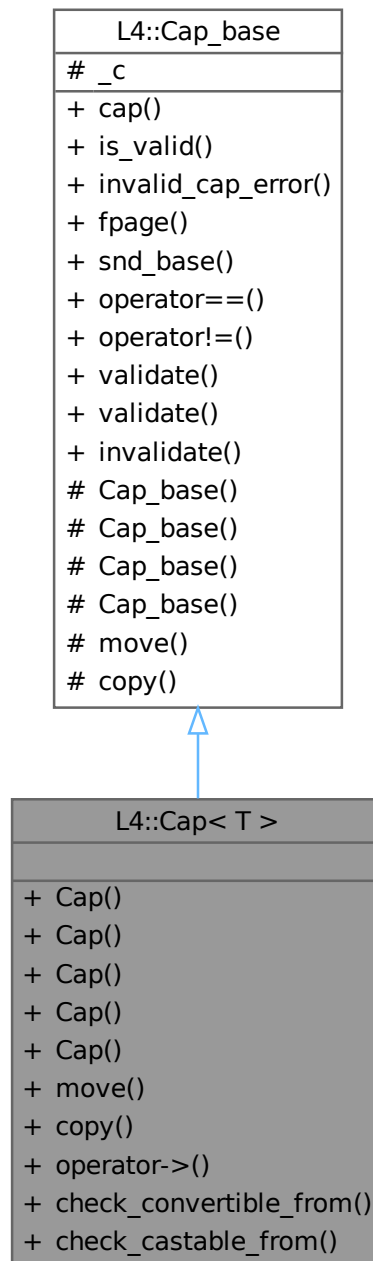


Additional Inherited Members

Public Member Functions inherited from [L4::Runtime_error](#)

- [Runtime_error](#) (long [err_no](#), char const *extra=0) noexcept
Create a new [Runtime_error](#).
- char const * [str](#) () const noexcept override
Return a human readable string for the exception.
- char const * [extra_str](#) () const
Get the description text for this runtime error.
- long [err_no](#) () const noexcept
Get the error value for this runtime error.

Collaboration diagram for L4::Cap< T >:



Public Member Functions

- `template<typename O>`
`Cap (Cap< O > const &o) noexcept`
Create a copy from o, supporting implicit type casting.
- `Cap (Cap_type cap) noexcept`
Constructor to create an invalid capability selector.

- [Cap](#) ([l4_default_caps_t](#) cap) noexcept
Initialize capability with one of the default capability selectors.
- [Cap](#) ([l4_cap_idx_t](#) idx=[L4_INVALID_CAP](#)) noexcept
Initialize capability, defaults to the invalid capability selector.
- [Cap](#) ([No_init_type](#)) noexcept
Create an uninitialized cap selector.
- [Cap](#) move ([Cap](#) const &src) const
Move a capability to this cap slot.
- [Cap](#) copy ([Cap](#) const &src) const
Copy a capability to this cap slot.
- [T](#) * **operator->** () const noexcept
Member access of a [T](#).

Public Member Functions inherited from [L4::Cap_base](#)

- [l4_cap_idx_t](#) cap () const noexcept
Return capability selector.
- bool [is_valid](#) () const noexcept
Test whether the capability is a valid capability index (i.e., not [L4_INVALID_CAP](#)).
- int [invalid_cap_error](#) () const noexcept
Return the transported error code in an invalid capability index.
- [l4_fpage_t](#) fpage (unsigned rights=[L4_CAP_FPAGE_RWS](#)) const noexcept
Return flexpage for the capability.
- [l4_umword_t](#) snd_base (unsigned grant=[L4_MAP_ITEM_MAP](#), [l4_cap_idx_t](#) base=[L4_INVALID_CAP](#)) const noexcept
Return send base.
- bool **operator==** ([Cap_base](#) const &o) const noexcept
Test if two capabilities are equal.
- bool **operator!=** ([Cap_base](#) const &o) const noexcept
Test if two capabilities are not equal.
- [l4_msgtag_t](#) validate ([l4_utcb_t](#) *u=[l4_utcb\(\)](#)) const noexcept
Check whether a capability is present (refers to an object).
- [l4_msgtag_t](#) validate ([Cap](#)< [Task](#) > task, [l4_utcb_t](#) *u=[l4_utcb\(\)](#)) const noexcept
Check whether a capability is present (refers to an object).
- void **invalidate** () noexcept
Set this capability to invalid ([L4_INVALID_CAP](#)).

Static Public Member Functions

- template<typename From>
static void [check_convertible_from](#) () noexcept
Perform the type conversion that needs to compile in order for a capability of type From to be convertible to one of type T.
- template<typename From>
static void [check_castable_from](#) () noexcept
Perform the type conversion that needs to compile in order for a capability of type From to be castable (via the correct cap_cast) to one of type T.

Friends

- class [L4::Kobject](#)

Additional Inherited Members

Public Types inherited from L4::Cap_base

- enum [No_init_type](#) { [No_init](#) }
Special value for uninitialized capability objects.
- enum [Cap_type](#) { [Invalid](#) = L4_INVALID_CAP }
Invalid capability type.

Protected Member Functions inherited from L4::Cap_base

- [Cap_base](#) ([l4_cap_idx_t](#) c) noexcept
Generate a capability from its C representation.
- [Cap_base](#) ([Cap_type](#) cap) noexcept
Constructor to create an invalid capability.
- [Cap_base](#) ([l4_default_caps_t](#) cap) noexcept
Initialize capability with one of the default capabilities.
- [Cap_base](#) () noexcept
Create an uninitialized instance.
- void [move](#) ([Cap_base](#) const &src) const
Replace this capability with the contents of `src`.
- void [copy](#) ([Cap_base](#) const &src) const
Copy a capability.

Protected Attributes inherited from L4::Cap_base

- [l4_cap_idx_t_c](#)
The C representation of a capability selector.

16.93.1 Detailed Description

```
template<typename T>
class L4::Cap< T >
```

C++ interface for capabilities.

Template Parameters

<i>T</i>	Type of the object the capability points to.
----------	--

The C++ version of a capability is comparable to a pointer, in fact it is a kind of smart pointer for our kernel objects and the objects derived from the kernel objects ([L4::Kobject](#)).

Add

```
#include <l4/sys/capability>
```

to your code to use the capability interface.

Examples

[examples/clntsrv/src/client.cc](#), [examples/libs/l4re/c++/mem_alloc/ma+rm.cc](#), [examples/libs/l4re/c++/shared_ds/ds_clnt.cc](#), [examples/libs/l4re/c++/shared_ds/ds_srv.cc](#), [examples/libs/l4re/streammap/client.cc](#), and [examples/sys/migrate/thread_migrate](#)

Definition at line 223 of file [capability.h](#).

16.93.2 Constructor & Destructor Documentation

16.93.2.1 Cap() [1/4]

```
template<typename T>
template<typename O>
L4::Cap< T >::Cap (
    Cap< O > const & o) [inline], [noexcept]
```

Create a copy from `o`, supporting implicit type casting.

Parameters

<code>o</code>	The source selector that shall be copied (and casted).
----------------	--

Definition at line 275 of file [capability.h](#).

16.93.2.2 Cap() [2/4]

```
template<typename T>
L4::Cap< T >::Cap (
    Cap_type cap) [inline], [noexcept]
```

Constructor to create an invalid capability selector.

Parameters

<code>cap</code>	Capability selector.
------------------	----------------------

Definition at line 282 of file [capability.h](#).

16.93.2.3 Cap() [3/4]

```
template<typename T>
L4::Cap< T >::Cap (
    l4_default_caps_t cap) [inline], [noexcept]
```

Initialize capability with one of the default capability selectors.

Parameters

<code>cap</code>	Capability selector.
------------------	----------------------

Definition at line 288 of file [capability.h](#).

16.93.2.4 Cap() [4/4]

```
template<typename T>
L4::Cap< T >::Cap (
    l4_cap_idx_t idx = L4_INVALID_CAP) [inline], [explicit], [noexcept]
```

Initialize capability, defaults to the invalid capability selector.

Parameters

<i>idx</i>	Capability selector.
------------	----------------------

Definition at line 294 of file [capability.h](#).

16.93.3 Member Function Documentation

16.93.3.1 check_castable_from()

```
template<typename T>
template<typename From>
void L4::Cap< T >::check_castable_from () [inline], [static], [noexcept]
```

Perform the type conversion that needs to compile in order for a capability of type From to be castable (via the correct `cap_cast`) to one of type T.

Template Parameters

<i>From</i>	Type to convert from
-------------	----------------------

Definition at line 264 of file [capability.h](#).

16.93.3.2 check_convertible_from()

```
template<typename T>
template<typename From>
void L4::Cap< T >::check_convertible_from () [inline], [static], [noexcept]
```

Perform the type conversion that needs to compile in order for a capability of type From to be convertible to one of type T.

Template Parameters

<i>From</i>	Type to convert from
-------------	----------------------

Definition at line 251 of file [capability.h](#).

16.93.3.3 copy()

```
template<typename T>
Cap L4::Cap< T >::copy (
    Cap< T > const & src) const [inline]
```

Copy a capability to this cap slot.

Parameters

<i>src</i>	the source capability slot.
------------	-----------------------------

Definition at line 317 of file [capability.h](#).

16.93.3.4 move()

```
template<typename T>
Cap L4::Cap< T >::move (
    Cap< T > const & src) const [inline]
```

Move a capability to this cap slot.

Parameters

<i>src</i>	the source capability slot.
------------	-----------------------------

After this operation the source slot is no longer valid.

Definition at line 307 of file [capability.h](#).

The documentation for this class was generated from the following file:

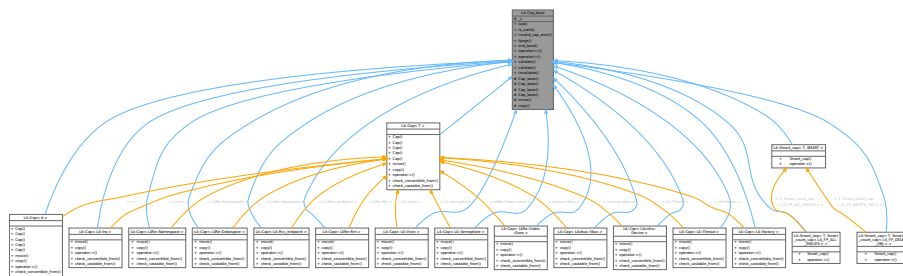
- l4/sys/cxx/capability.h

16.94 L4::Cap_base Class Reference

Base class for all kinds of capabilities.

```
#include <l4/sys/capability>
```

Inheritance diagram for L4::Cap_base:



Collaboration diagram for L4::Cap_base:

L4::Cap_base
_c
+ cap()
+ is_valid()
+ invalid_cap_error()
+ fpage()
+ snd_base()
+ operator==()
+ operator!=()
+ validate()
+ validate()
+ invalidate()
Cap_base()
Cap_base()
Cap_base()
Cap_base()
move()
copy()

Public Types

- enum [No_init_type](#) { [No_init](#) }
Special value for uninitialized capability objects.
- enum [Cap_type](#) { [Invalid](#) = L4_INVALID_CAP }
Invalid capability type.

Public Member Functions

- [l4_cap_idx_t](#) [cap](#) () const noexcept
Return capability selector.
- bool [is_valid](#) () const noexcept
Test whether the capability is a valid capability index (i.e., not L4_INVALID_CAP).
- int [invalid_cap_error](#) () const noexcept
Return the transported error code in an invalid capability index.
- [l4_fpage_t](#) [fpage](#) (unsigned rights=[L4_CAP_FPAGE_RWS](#)) const noexcept
Return flexpage for the capability.
- [l4_umword_t](#) [snd_base](#) (unsigned grant=[L4_MAP_ITEM_MAP](#), [l4_cap_idx_t](#) base=[L4_INVALID_CAP](#)) const noexcept
Return send base.

- bool **operator==** ([Cap_base](#) const &o) const noexcept
Test if two capabilities are equal.
- bool **operator!=** ([Cap_base](#) const &o) const noexcept
Test if two capabilities are not equal.
- [l4_msgtag_t validate](#) ([l4_utcb_t](#) *u=[l4_utcb](#)()) const noexcept
Check whether a capability is present (refers to an object).
- [l4_msgtag_t validate](#) ([Cap](#)< [Task](#) > task, [l4_utcb_t](#) *u=[l4_utcb](#)()) const noexcept
Check whether a capability is present (refers to an object).
- void **invalidate** () noexcept
Set this capability to invalid ([L4_INVALID_CAP](#)).

Protected Member Functions

- [Cap_base](#) ([l4_cap_idx_t](#) c) noexcept
Generate a capability from its C representation.
- **Cap_base** ([Cap_type](#) cap) noexcept
Constructor to create an invalid capability.
- [Cap_base](#) ([l4_default_caps_t](#) cap) noexcept
Initialize capability with one of the default capabilities.
- **Cap_base** () noexcept
Create an uninitialized instance.
- void [move](#) ([Cap_base](#) const &src) const
*Replace this capability with the contents of *src*.*
- void [copy](#) ([Cap_base](#) const &src) const
Copy a capability.

Protected Attributes

- [l4_cap_idx_t _c](#)
The C representation of a capability selector.

16.94.1 Detailed Description

Base class for all kinds of capabilities.

Attention

This class is not for direct use, use [L4::Cap](#) instead.

This class contains all the things that are independent of the type of the object referred by the capability.

See also

[L4::Cap](#) for typed capabilities.

Definition at line 25 of file [capability.h](#).

16.94.2 Member Enumeration Documentation

16.94.2.1 Cap_type

```
enum L4::Cap\_base::Cap\_type
```

Invalid capability type.

Enumerator

Invalid	Invalid capability selector.
---------	------------------------------

Definition at line 40 of file [capability.h](#).

16.94.2.2 No_init_type

```
enum L4::Cap_base::No_init_type
```

Special value for uninitialized capability objects.

Enumerator

No_init	Special value for constructing uninitialized Cap objects.
---------	---

Definition at line 29 of file [capability.h](#).

16.94.3 Constructor & Destructor Documentation

16.94.3.1 Cap_base() [1/2]

```
L4::Cap_base::Cap_base (
    l4_cap_idx_t c) [inline], [explicit], [protected], [noexcept]
```

Generate a capability from its C representation.

Parameters

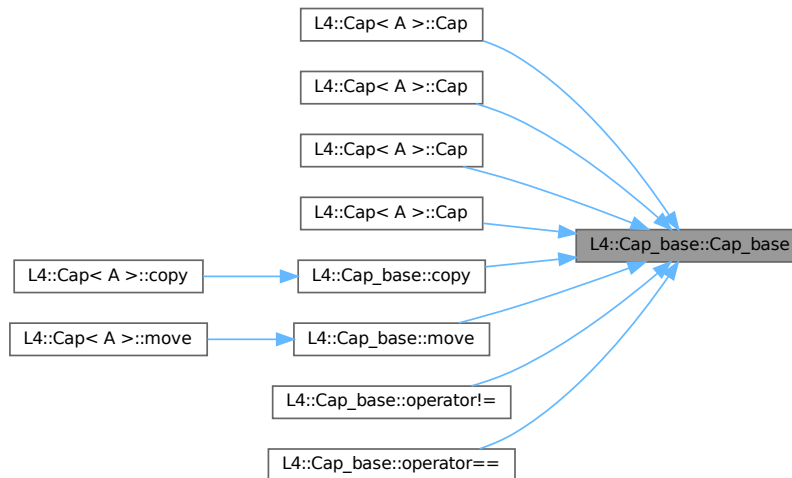
<i>c</i>	The C capability
----------	------------------

Definition at line 149 of file [capability.h](#).

References [_c](#).

Referenced by [L4::Cap< A >::Cap\(\)](#), [L4::Cap< A >::Cap\(\)](#), [L4::Cap< A >::Cap\(\)](#), [L4::Cap< A >::Cap\(\)](#), [copy\(\)](#), [move\(\)](#), [operator!=\(\)](#), and [operator==\(\)](#).

Here is the caller graph for this function:



16.94.3.2 Cap_base() [2/2]

```

L4::Cap_base::Cap_base (
    l4_default_caps_t cap) [inline], [explicit], [protected], [noexcept]

```

Initialize capability with one of the default capabilities.

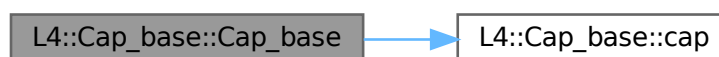
Parameters

<i>cap</i>	Capability.
------------	-------------

Definition at line 160 of file [capability.h](#).

References [_c](#), and [cap\(\)](#).

Here is the call graph for this function:



16.94.4 Member Function Documentation

16.94.4.1 cap()

```
l4_cap_idx_t L4::Cap_base::cap () const [inline], [noexcept]
```

Return capability selector.

Returns

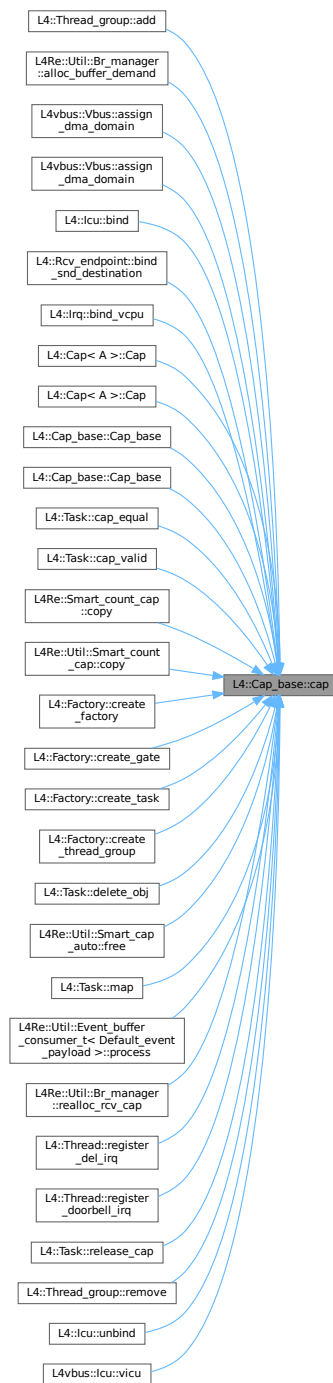
Capability selector.

Definition at line 49 of file [capability.h](#).

References [_c](#).

Referenced by [L4::Thread_group::add\(\)](#), [L4Re::Util::Br_manager::alloc_buffer_demand\(\)](#), [L4vbus::Vbus::assign_dma_domain\(\)](#), [L4vbus::Vbus::assign_dma_domain\(\)](#), [L4::lcu::bind\(\)](#), [L4::Rcv_endpoint::bind_snd_destination\(\)](#), [L4::Irq::bind_vcpu\(\)](#), [L4::Cap< A >::Cap\(\)](#), [L4::Cap< A >::Cap\(\)](#), [Cap_base\(\)](#), [Cap_base\(\)](#), [L4::Task::cap_equal\(\)](#), [L4::Task::cap_valid\(\)](#), [L4Re::Smart_count_cap< Unmap_flags >::copy\(\)](#), [L4Re::Util::Smart_count_cap< Unmap_flags >::copy\(\)](#), [L4::Factory::create_factory\(\)](#), [L4::Factory::create_gate\(\)](#), [L4::Factory::create_task\(\)](#), [L4::Factory::create_thread_group\(\)](#), [L4::Task::delete_obj\(\)](#), [L4Re::Util::Smart_cap_auto< Unmap_flags >::free\(\)](#), [L4::Task::map\(\)](#), [L4Re::Util::Event_buffer_consumer_t< T >::map\(\)](#), [L4Re::Util::Br_manager::realloc_rcv_cap\(\)](#), [L4::Thread::register_del_irq\(\)](#), [L4::Thread::register_doorbell_irq\(\)](#), [L4::Task::release_cap\(\)](#), [L4::Thread_group::remove\(\)](#), [L4::lcu::unbind\(\)](#), and [L4vbus::lcu::vicu\(\)](#).

Here is the caller graph for this function:



16.94.4.2 copy()

```
void L4::Cap_base::copy (
    Cap_base const & src) const [inline], [protected]
```

Copy a capability.

Parameters

<i>src</i>	the source capability.
------------	------------------------

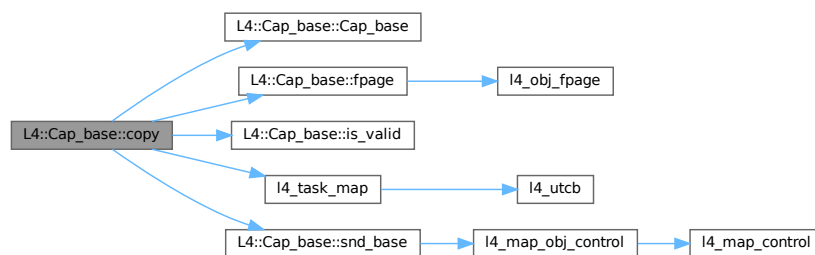
After this operation this capability refers to the same object as `src`.

Definition at line 192 of file `capability.h`.

References `Cap_base()`, `fpage()`, `is_valid()`, `L4_BASE_TASK_CAP`, `L4_CAP_FPAGE_RWSD`, `L4_FPAGE_C_OBJ_RIGHTS`, `l4_task_map()`, and `snd_base()`.

Referenced by `L4::Cap< A >::copy()`.

Here is the call graph for this function:



Here is the caller graph for this function:



16.94.4.3 fpage()

```

l4_fpage_t L4::Cap_base::fpage (
    unsigned rights = L4_CAP_FPAGE_RWS) const [inline], [noexcept]
  
```

Return flexpage for the capability.

Parameters

<i>rights</i>	Rights, defaults to 'rws'
---------------	---------------------------

Returns

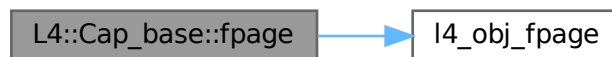
flexpage

Definition at line 74 of file [capability.h](#).

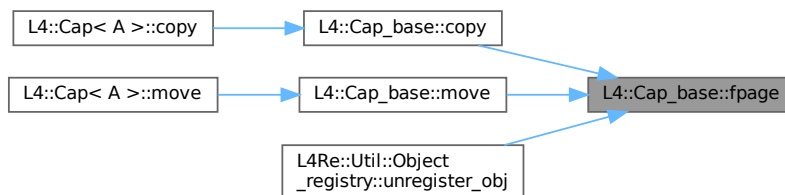
References [_c](#), [L4_CAP_FPAGE_RWS](#), and [l4_obj_fpage\(\)](#).

Referenced by [copy\(\)](#), [move\(\)](#), and [L4Re::Util::Object_registry::unregister_obj\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**16.94.4.4 is_valid()**

```
bool L4::Cap_base::is_valid () const [inline], [noexcept]
```

Test whether the capability is a valid capability index (i.e., not `L4_INVALID_CAP`).

Returns

True if capability is not invalid, false if invalid

Examples

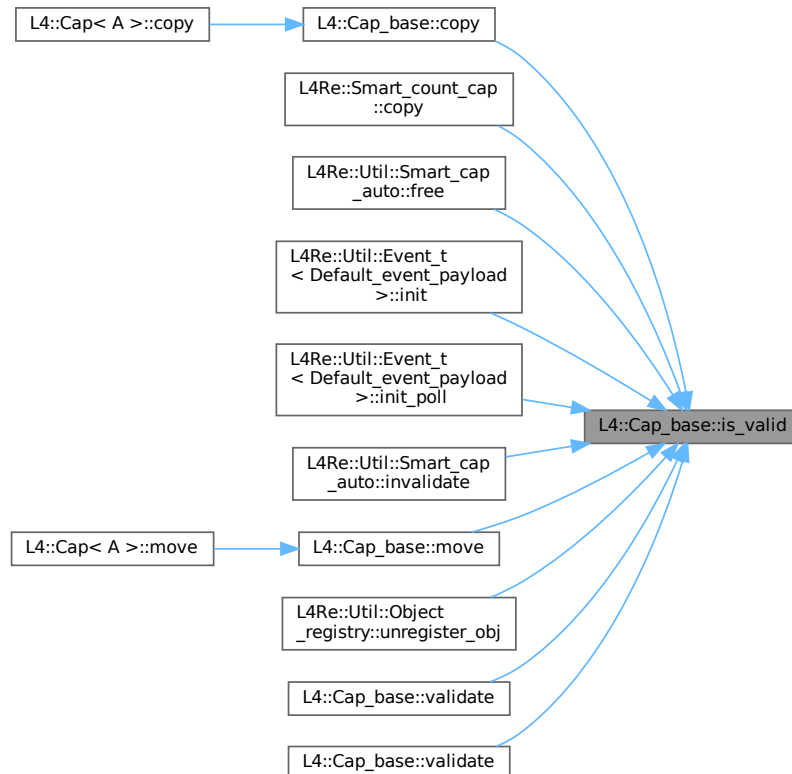
[examples/libs/l4re/c++/mem_alloc/ma+rm.cc](#), and [examples/libs/l4re/c++/shared_ds/ds_clnt.cc](#).

Definition at line 57 of file [capability.h](#).

References [_c](#).

Referenced by [copy\(\)](#), [L4Re::Smart_count_cap< Unmap_flags >::copy\(\)](#), [L4Re::Util::Smart_cap_auto< Unmap_flags >::free\(\)](#), [L4Re::Util::Event_t< Default_event_payload >::init\(\)](#), [L4Re::Util::Event_t< Default_event_payload >::init_poll\(\)](#), [L4Re::Util::Smart_cap_auto< Unmap_flags >::invalidate\(\)](#), [move\(\)](#), [L4Re::Util::Object_registry::unregister_obj\(\)](#), [validate\(\)](#), and [validate\(\)](#).

Here is the caller graph for this function:



16.94.4.5 move()

```
void L4::Cap_base::move (
    Cap_base const & src) const [inline], [protected]
```

Replace this capability with the contents of `src`.

Parameters

<code>src</code>	the source capability.
------------------	------------------------

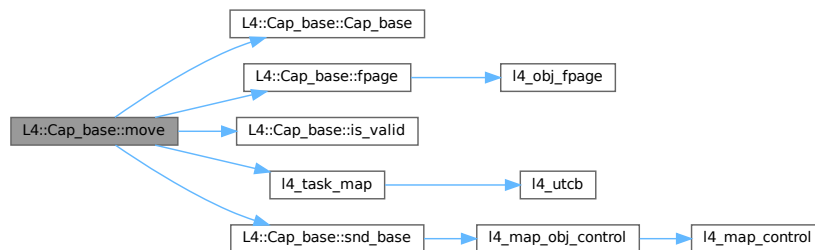
After the operation this capability refers to the object formerly referred to by the source capability `src`, and the source capability no longer refers to an object.

Definition at line 176 of file [capability.h](#).

References [Cap_base\(\)](#), [fpage\(\)](#), [is_valid\(\)](#), [L4_BASE_TASK_CAP](#), [L4_CAP_FPAGE_RWSD](#), [L4_FPAGE_C_OBJ_RIGHTS](#), [L4_MAP_ITEM_GRANT](#), [l4_task_map\(\)](#), and [snd_base\(\)](#).

Referenced by [L4::Cap< A >::move\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.94.4.6 snd_base()

```

l4_umword_t L4::Cap_base::snd_base (
    unsigned grant = L4_MAP_ITEM_MAP,
    l4_cap_idx_t base = L4_INVALID_CAP) const [inline], [noexcept]
  
```

Return send base.

Parameters

<i>grant</i>	Indicates if object shall be granted. Allowed values: L4_MAP_ITEM_MAP , L4_MAP_ITEM_GRANT .
<i>base</i>	Base capability (first in a bundle of aligned capabilities)

Returns

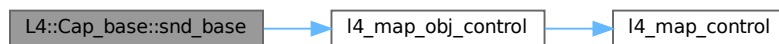
Map object.

Definition at line 86 of file [capability.h](#).

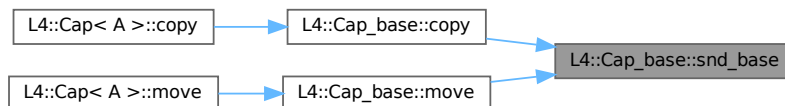
References [_c](#), [L4_INVALID_CAP](#), [L4_MAP_ITEM_MAP](#), and [l4_map_obj_control\(\)](#).

Referenced by [copy\(\)](#), and [move\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**16.94.4.7 validate() [1/2]**

```

l4_msgtag_t L4::Cap_base::validate (
    Cap< Task > task,
    l4_utcb_t * u = l4_utcb()) const [inline], [noexcept]
  
```

Check whether a capability is present (refers to an object).

Parameters

<i>task</i>	Task to check the capability in.
<i>u</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Return values

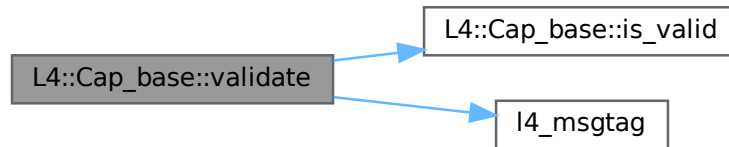
<code>l4_msgtag_t::label() > 0</code>	Capability is present (refers to an object).
<code>l4_msgtag_t::label() == 0</code>	No capability present (void object or invalid capability slot).

A capability is considered present when it refers to an existing kernel object.

Definition at line 74 of file [capability](#).

References [_c](#), [is_valid\(\)](#), and [l4_msgtag\(\)](#).

Here is the call graph for this function:



16.94.4.8 validate() [2/2]

```
l4_msgtag_t L4::Cap_base::validate (
    l4_utcb_t * u = l4_utcb()) const [inline], [noexcept]
```

Check whether a capability is present (refers to an object).

Parameters

<i>u</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .
----------	--

Return values

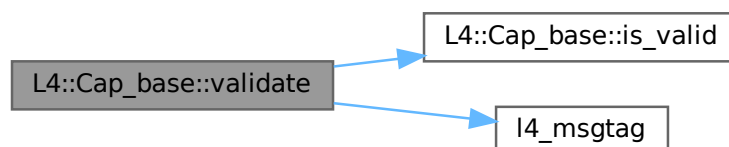
<i>l4_msgtag_t::label()</i> > 0	Capability is present (refers to an object).
<i>l4_msgtag_t::label()</i> == 0	No capability present (void object or invalid capability slot).

A capability is considered present when it refers to an existing kernel object.

Definition at line 81 of file [capability](#).

References [_c](#), [is_valid\(\)](#), [L4_BASE_TASK_CAP](#), and [l4_msgtag\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following files:

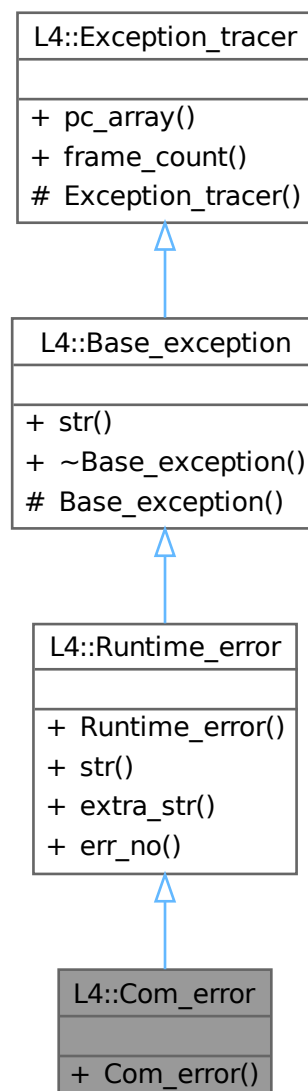
- [l4/sys/cxx/capability.h](#)
- [l4/sys/capability](#)

16.95 L4::Com_error Class Reference

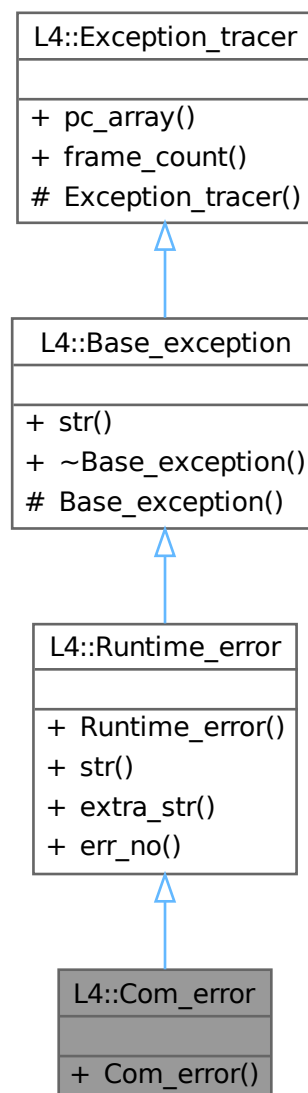
Error conditions during IPC.

```
#include <l4/cxx/exceptions>
```

Inheritance diagram for L4::Com_error:



Collaboration diagram for L4::Com_error:



Public Member Functions

- [Com_error](#) (long err) noexcept
Create a [Com_error](#) for the given [L4](#) IPC error code.

Public Member Functions inherited from [L4::Runtime_error](#)

- [Runtime_error](#) (long err_no, char const *extra=0) noexcept
Create a new [Runtime_error](#).
- char const * [str](#) () const noexcept override

Return a human readable string for the exception.

- char const * [extra_str](#) () const

Get the description text for this runtime error.

- long [err_no](#) () const noexcept

Get the error value for this runtime error.

Public Member Functions inherited from [L4::Base_exception](#)

- virtual ~**Base_exception** () noexcept

Destruction.

Public Member Functions inherited from [L4::Exception_tracer](#)

- void const *const * **pc_array** () const noexcept

Get the array containing the call trace.

- int **frame_count** () const noexcept

Get the number of entries that are valid in the call trace.

Additional Inherited Members

Protected Member Functions inherited from [L4::Base_exception](#)

- **Base_exception** () noexcept

Create a base exception.

Protected Member Functions inherited from [L4::Exception_tracer](#)

- **Exception_tracer** () noexcept

Create a back trace.

16.95.1 Detailed Description

Error conditions during IPC.

This exception encapsulates all IPC error conditions of [L4](#) IPC.

Definition at line [263](#) of file [exceptions](#).

16.95.2 Constructor & Destructor Documentation

16.95.2.1 Com_error()

```
L4::Com_error::Com_error (
    long err) [inline], [explicit], [noexcept]
```

Create a [Com_error](#) for the given [L4](#) IPC error code.

Parameters

<i>err</i>	The L4 IPC error code (l4_ipc... return value).
------------	---

Definition at line 270 of file [exceptions](#).

The documentation for this class was generated from the following file:

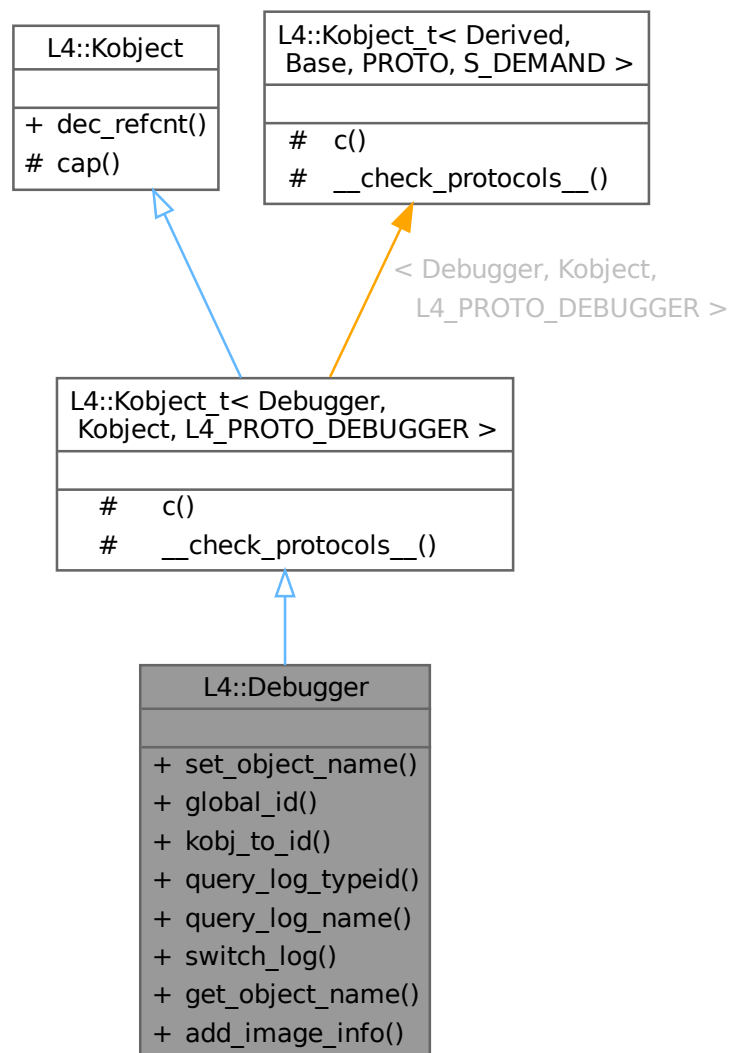
- [l4/cxx/exceptions](#)

16.96 L4::Debugger Class Reference

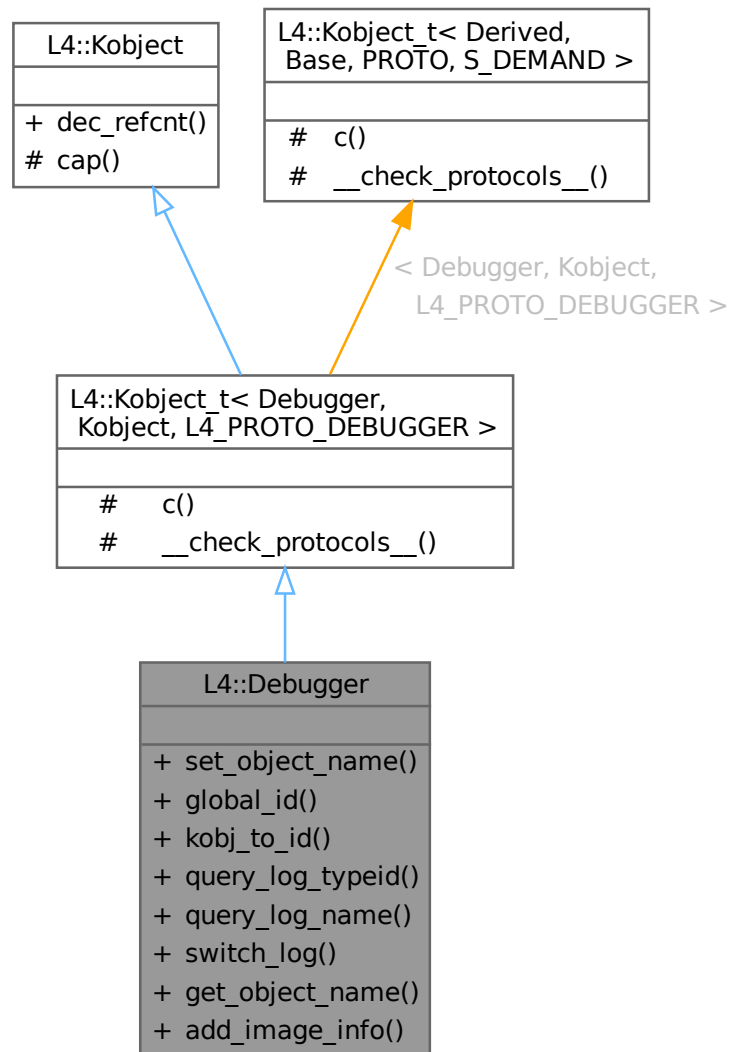
C++ kernel debugger API.

```
#include <debugger>
```

Inheritance diagram for L4::Debugger:



Collaboration diagram for L4::Debugger:



Public Member Functions

- [l4_msgtag_t set_object_name](#) (const char *name, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept
Set the name of a kernel object.
- unsigned long [global_id](#) ([l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept
Get the globally unique ID of the object behind a capability.
- unsigned long [kobj_to_id](#) ([l4_addr_t](#) kobjp, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept
Get the globally unique ID of the object behind the kobject pointer.
- long [query_log_typeid](#) (const char *name, unsigned idx, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept
Query the log-id for a log type.
- long [query_log_name](#) (unsigned idx, char *name, unsigned namelen, char *shortname, unsigned short-namelen, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept

Query the name of a log type given the ID.

- [l4_msgtag_t switch_log](#) (const char *name, unsigned on_off, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept

Set or unset log.

- [l4_msgtag_t get_object_name](#) (unsigned id, char *name, unsigned size, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept

*Get name of object with Id *i* d.*

- [l4_msgtag_t add_image_info](#) ([l4_addr_t](#) base, const char *name, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept

Add loaded image information for a task.

Public Member Functions inherited from [L4::Kobject](#)

- [l4_msgtag_t dec_refcnt](#) ([l4_mword_t](#) diff, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#))

Decrement the in kernel reference counter for the object.

Additional Inherited Members

Protected Types inherited from

[L4::Kobject_t](#)< [Debugger](#), [Kobject](#), [L4_PROTO_DEBUGGER](#) >

- typedef [Debugger](#) **Class**

The target interface type (inheriting from [Kobject_t](#)).

- typedef Typeid::Iface< [PROTO](#), [Debugger](#) > **__iface**

The interface description for the derived class.

- typedef Typeid::Merge_list< Typeid::Iface_list< **__iface** >, typename [Kobject](#)::__iface_list > **__iface_list**

The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from

[L4::Kobject_t](#)< [Debugger](#), [Kobject](#), [L4_PROTO_DEBUGGER](#) >

- [L4::Cap](#)< **Class** > **c** () const noexcept

Get the capability to ourselves.

Protected Member Functions inherited from [L4::Kobject](#)

- [l4_cap_idx_t cap](#) () const noexcept

Return capability selector.

Static Protected Member Functions inherited from

[L4::Kobject_t](#)< [Debugger](#), [Kobject](#), [L4_PROTO_DEBUGGER](#) >

- static void **__check_protocols** () noexcept

Helper to check for protocol conflicts.

16.96.1 Detailed Description

C++ kernel debugger API.

Attention

This API is subject to change! Do not rely on it in production code.

This API is to be used for debugging exclusively.

This is the API for accessing kernel-debugger functionality from user-level programs. Specifically, it provides functionality to enrich the kernel debugger with insights into the program. The purpose is to facilitate debugging with the kernel debugger. For instance, a developer might choose to name the threads of her program so that she can find them in the kernel debugger thread list.

This API interacts with a kernel object that interfaces with the kernel debugger, the jdb-kernel object. The jdb-kernel object is fix and only available when the kernel debugger is built into the microkernel. The developer needs to pass the capability through to her program.

Include File

```
#include <l4/sys/debugger>
```

Definition at line 42 of file [debugger](#).

16.96.2 Member Function Documentation

16.96.2.1 add_image_info()

```
l4_msgtag_t L4::Debugger::add_image_info (
    l4_addr_t base,
    const char * name,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
```

Add loaded image information for a task.

Parameters

<i>base</i>	Image load base address
<i>name</i>	Image name
<i>utcb</i>	The UTCB to use for the operation.

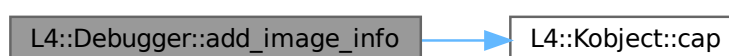
Returns

System call return tag.

Definition at line 161 of file [debugger](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



16.96.2.2 `get_object_name()`

```
l4_msgtag_t L4::Debugger::get_object_name (
    unsigned id,
    char * name,
    unsigned size,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
```

Get name of object with Id `id`.

Parameters

	<i>id</i>	Id of the object whose name is asked.
out	<i>name</i>	Buffer to copy the name into. The buffer must be allocated by the caller.
	<i>size</i>	Length of the <code>name</code> buffer.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

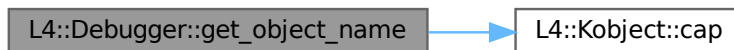
Returns

Syscall return tag

Definition at line 148 of file [debugger](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



16.96.2.3 `global_id()`

```
unsigned long L4::Debugger::global_id (
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
```

Get the globally unique ID of the object behind a capability.

Parameters

<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .
-------------	--

Return values

$\sim 0UL$	The capability is invalid.
≥ 0	The global debugger id.

Definition at line 71 of file [debugger](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



16.96.2.4 kobj_to_id()

```

unsigned long L4::Debugger::kobj_to_id (
    l4_addr_t kobjp,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Get the globally unique ID of the object behind the kobject pointer.

Parameters

<i>kobjp</i>	Kobject pointer
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Return values

$\sim 0UL$	The capability or the Kobject pointer are invalid.
≥ 0	The globally unique id.

Definition at line 83 of file [debugger](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



16.96.2.5 query_log_name()

```
long L4::Debugger::query_log_name (
    unsigned idx,
    char * name,
    unsigned namelen,
    char * shortname,
    unsigned shortnamelen,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
```

Query the name of a log type given the ID.

Parameters

	<i>idx</i>	ID to query.
out	<i>name</i>	Buffer to copy name to. The buffer must be allocated by the caller.
	<i>namelen</i>	Buffer length of name.
out	<i>shortname</i>	Buffer to copy shortname to. The buffer must be allocated by the caller.
	<i>shortnamelen</i>	Buffer length of shortname.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Return values

0	Success
<0	Error

Definition at line 116 of file [debugger](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



16.96.2.6 query_log_typeid()

```
long L4::Debugger::query_log_typeid (
    const char * name,
    unsigned idx,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
```

Query the log-id for a log type.

Parameters

<i>name</i>	Name to query for.
<i>idx</i>	Idx to start searching, start with 0
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Return values

≥ 0	Id
< 0	Error

Definition at line 97 of file [debugger](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



16.96.2.7 set_object_name()

```

l4_msgtag_t L4::Debugger::set_object_name (
    const char * name,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Set the name of a kernel object.

Parameters

<i>name</i>	Name
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

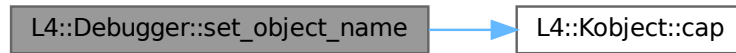
Returns

System call return tag.

Definition at line 59 of file [debugger](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



16.96.2.8 switch_log()

```

l4_msgtag_t L4::Debugger::switch_log (
    const char * name,
    unsigned on_off,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Set or unset log.

Parameters

<i>name</i>	Name of the log type.
<i>on_off</i>	1: turn log on, 0: turn log off
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Returns

Syscall return tag

Definition at line 133 of file [debugger](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

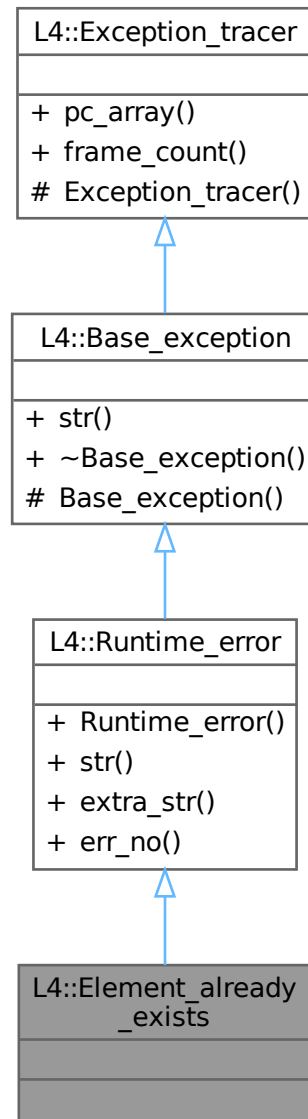
- [l4/sys/debugger](#)

16.97 L4::Element_already_exists Class Reference

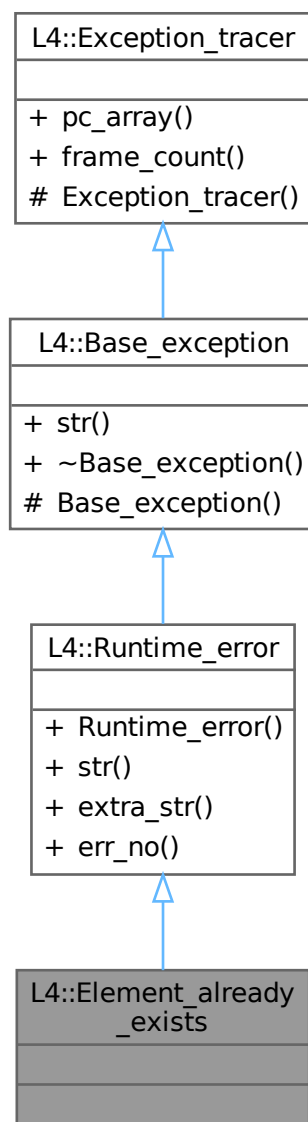
[Exception](#) for duplicate element insertions.

```
#include <l4/cxx/exceptions>
```

Inheritance diagram for L4::Element_already_exists:



Collaboration diagram for L4::Element_already_exists:



Additional Inherited Members

Public Member Functions inherited from [L4::Runtime_error](#)

- [Runtime_error](#) (long [err_no](#), char const *extra=0) noexcept
Create a new [Runtime_error](#).
- char const * [str](#) () const noexcept override
Return a human readable string for the exception.
- char const * [extra_str](#) () const
Get the description text for this runtime error.
- long [err_no](#) () const noexcept
Get the error value for this runtime error.

Public Member Functions inherited from [L4::Base_exception](#)

- virtual **~Base_exception** () noexcept
Destruction.

Public Member Functions inherited from [L4::Exception_tracer](#)

- void const *const * **pc_array** () const noexcept
Get the array containing the call trace.
- int **frame_count** () const noexcept
Get the number of entries that are valid in the call trace.

Protected Member Functions inherited from [L4::Base_exception](#)

- **Base_exception** () noexcept
Create a base exception.

Protected Member Functions inherited from [L4::Exception_tracer](#)

- **Exception_tracer** () noexcept
Create a back trace.

16.97.1 Detailed Description

[Exception](#) for duplicate element insertions.

Definition at line 192 of file [exceptions](#).

The documentation for this class was generated from the following file:

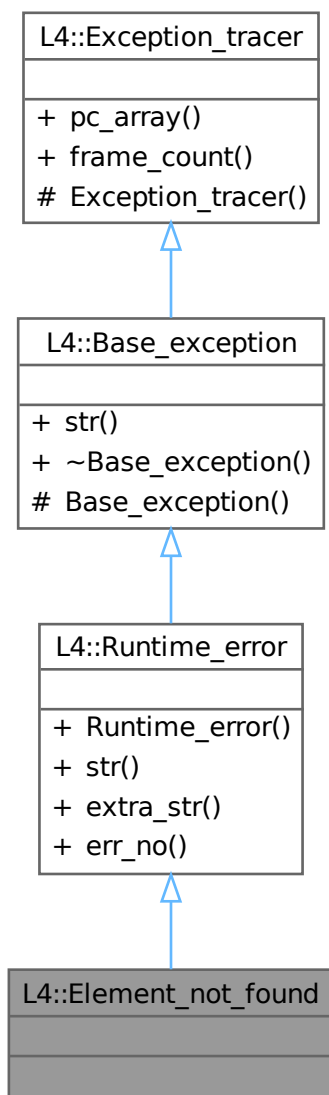
- [l4/cxx/exceptions](#)

16.98 [L4::Element_not_found](#) Class Reference

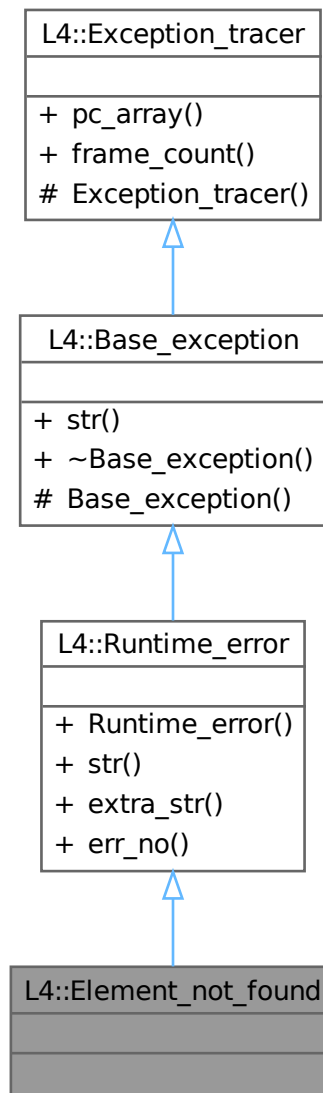
[Exception](#) for a failed lookup (element not found).

```
#include <l4/cxx/exceptions>
```


Inheritance diagram for L4::Element_not_found:



Collaboration diagram for L4::Element_not_found:



Additional Inherited Members

Public Member Functions inherited from [L4::Runtime_error](#)

- [Runtime_error](#) (long [err_no](#), char const *extra=0) noexcept
Create a new [Runtime_error](#).
- char const * [str](#) () const noexcept override
Return a human readable string for the exception.
- char const * [extra_str](#) () const
Get the description text for this runtime error.
- long [err_no](#) () const noexcept
Get the error value for this runtime error.

Public Member Functions inherited from L4::Base_exception

- virtual ~**Base_exception** () noexcept
Destruction.

Public Member Functions inherited from L4::Exception_tracer

- `void const *const * pc_array () const noexcept`
Get the array containing the call trace.
- `int frame_count () const noexcept`
Get the number of entries that are valid in the call trace.

Protected Member Functions inherited from L4::Base_exception

- **Base_exception ()** noexcept
Create a base exception.

Protected Member Functions inherited from [L4::Exception_tracer](#)

- **Exception_tracer ()** noexcept
Create a back trace.

16.98.1 Detailed Description

Exception for a failed lookup (element not found).

Definition at line 220 of file exceptions.

The documentation for this class was generated from the following file:

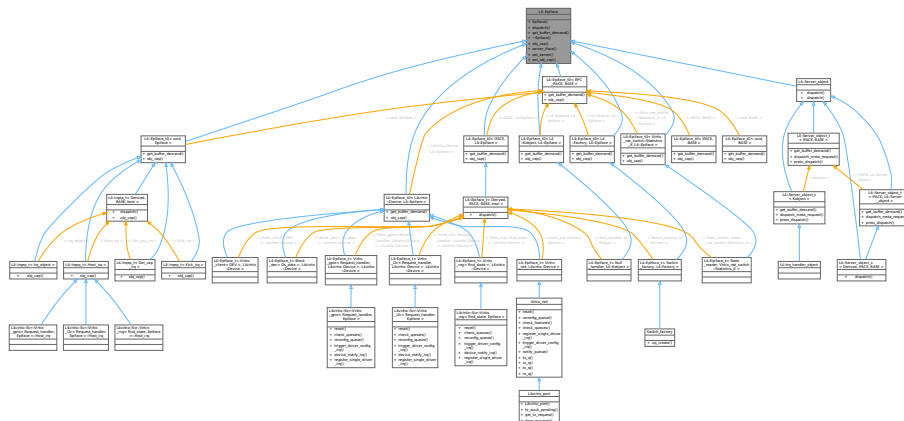
- l4/cxx/exceptions

16.99 L4::Epiface Struct Reference

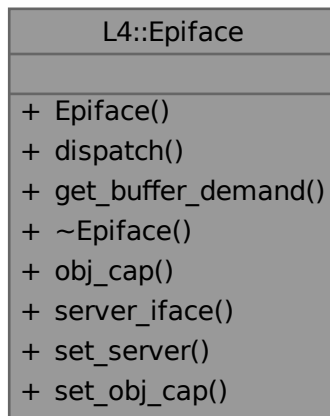
Base class for interface implementations.

```
#include <ipc_epiface>
```

Inheritance diagram for L4::Epiface:



Collaboration diagram for L4::Epiface:



Public Types

- typedef [lpc_svr::Server_iface](#) **Server_iface**
Type for abstract server interface.
- typedef [lpc_svr::Server_iface::Demand](#) **Demand**
Type for server-side receive buffer demand.

Public Member Functions

- **Epiface ()**
Make a server object.
- virtual [l4_msgtag_t](#) **dispatch** ([l4_msgtag_t](#) tag, unsigned rights, [l4_utcb_t](#) *utcb)=0
The abstract handler for client requests to the object.
- virtual [Demand](#) **get_buffer_demand** () const =0
Get the server-side receive buffer demand for this object.
- virtual **~Epiface** ()=0
Destroy the object.
- Stored_cap **obj_cap** () const
Get the capability to the kernel object belonging to this object.
- [Server_iface](#) * **server_iface** () const
Get pointer to server interface at which the object is currently registered.
- int **set_server** ([Server_iface](#) *srv, [Cap](#)< void > cap, bool managed=false)
Set server registration info for the object.
- void **set_obj_cap** ([Cap](#)< void > const &cap)
Deprecated server registration function.

16.99.1 Detailed Description

Base class for interface implementations.

An [Epiface](#) is the base interface of objects registered in the server loop. Incoming IPC gets dispatched to the appropriate [Epiface](#) object where the call is then handled appropriately.

Note

[Server](#) loops are allowed to internally keep raw pointers to [Epiface](#) objects for dispatching calls. Instances must therefore never be copied or moved.

Definition at line 145 of file [ipc_epiface](#).

16.99.2 Member Function Documentation

16.99.2.1 dispatch()

```
virtual l4_msgtag_t L4::Epiface::dispatch (
    l4_msgtag_t tag,
    unsigned rights,
    l4_utcb_t * utcb) [pure virtual]
```

The abstract handler for client requests to the object.

Parameters

<i>tag</i>	The message tag for this invocation.
<i>rights</i>	The rights bits in the invoked capability.
<i>utcb</i>	The UTCB used for the invocation.

Return values

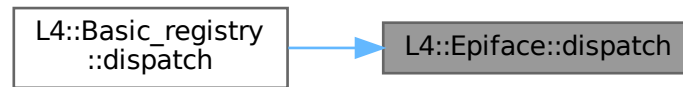
<code>-L4_ENOREPLY</code>	No reply message is send.
<code><0</code>	Error, reply with error code.
<code>>=0</code>	Success, reply with return value.

This function must be implemented by application specific server objects.

Implemented in [L4::Epiface_t< Derived, IFACE, BASE, bool >](#), [L4::Epiface_t< Block_dev< Ds_data >, L4virtio::Device >](#), [L4::Epiface_t< Null_handler, L4::Kobject >](#), [L4::Epiface_t< Stats_reader, Virtio_net_switch::Statistics_if >](#), [L4::Epiface_t< Switch_factory, L4::Factory >](#), [L4::Epiface_t< Virtio_client< DEV >, L4virtio::Device >](#), [L4::Epiface_t< Virtio_gpio< Virtio_i2c< Request_handler, L4virtio::Device >, L4virtio::Device >](#), [L4::Epiface_t< Virtio_net, L4virtio::Device >](#), [L4::Epiface_t< Virtio_rng< Rnd_state >, L4virtio::Device >](#), [L4::lrqep_t< Derived, BASE, bool >](#), [L4::lrqep_t< Del_cap_irq >](#), [L4::lrqep_t< Host_irq >](#), [L4::lrqep_t< Irq_object >](#), [L4::lrqep_t< Kick_irq >](#), and [L4::Server_object](#).

Referenced by [L4::Basic_registry::dispatch\(\)](#).

Here is the caller graph for this function:



16.99.2.2 `get_buffer_demand()`

```
virtual Demand L4::Epiface::get_buffer_demand () const [pure virtual]
```

Get the server-side receive buffer demand for this object.

Note

This function is usually not implemented directly, but by using `Server_object_t` template with an IPC interface definition.

Returns

The needed server-side receive buffers for this object

Implemented in `L4::Epiface_t0< RPC_IFACE, BASE >`, `L4::Epiface_t0< IFACE, L4::Epiface >`, `L4::Epiface_t0< L4::Factory, L4::Epiface >`, `L4::Epiface_t0< L4::Kobject, L4::Epiface >`, `L4::Epiface_t0< L4virtio::Device, L4::Epiface >`, `L4::Epiface_t0< Virtio_net_switch::State, L4::Epiface >`, `L4::Epiface_t0< void, Epiface >`, `L4::Server_object_t< IFACE, BASE >`, `L4::Server_object_t< IFACE, L4::Server_object >`, and `L4::Server_object_t< Kobject >`.

16.99.2.3 `obj_cap()`

```
Stored_cap L4::Epiface::obj_cap () const [inline]
```

Get the capability to the kernel object belonging to this object.

Returns

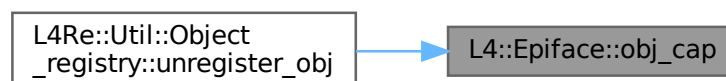
Capability for the kernel object behind the server.

This is usually either an `lpc_gate` or an `lrc`.

Definition at line 206 of file `ipc_epiface`.

Referenced by `L4Re::Util::Object_registry::unregister_obj()`.

Here is the caller graph for this function:



16.99.2.4 server_iface()

```
Server_iface * L4::Epiface::server_iface () const [inline]
```

Get pointer to server interface at which the object is currently registered.

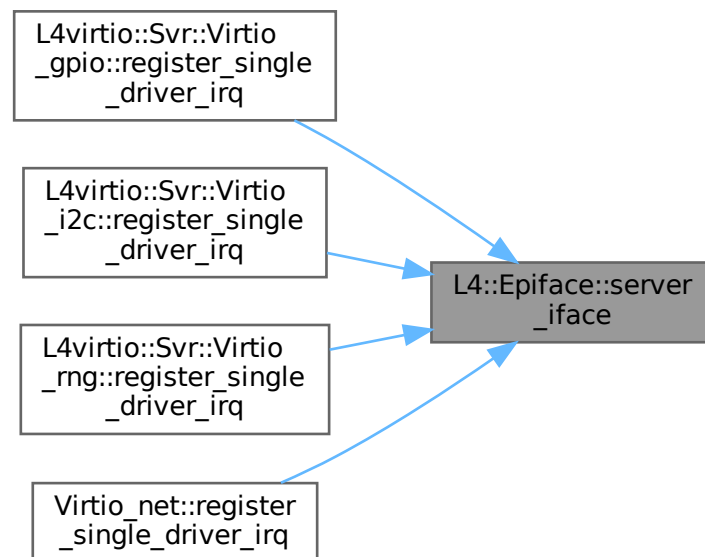
Returns

Pointer to the server at which the object is currently registered, NULL if the object is not registered at any server.

Definition at line 213 of file [ipc_epiface](#).

Referenced by [L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::register_single_driver_irq\(\)](#), [L4virtio::Svr::Virtio_i2c< Request_handler, Epiface >::register_single_driver_irq\(\)](#), [L4virtio::Svr::Virtio_rng< Rnd_state, Epiface >::register_single_driver_irq\(\)](#), and [Virtio_net::register_single_driver_irq\(\)](#).

Here is the caller graph for this function:



16.99.2.5 set_server()

```
int L4::Epiface::set_server (
    Server_iface * srv,
    Cap< void > cap,
    bool managed = false) [inline]
```

Set server registration info for the object.

Parameters

<i>srv</i>	The server to register at
<i>cap</i>	The capability that connects the object.
<i>managed</i>	Mark the capability as managed or unmanaged. Typical server implementations use this flag to remember whether the capability was internally allocated or not.

Returns

0 on success, -L4_EINVAL if the *srv* and *cap* are not consistent.

Definition at line 224 of file [ipc_epiface](#).

References [L4_EINVAL](#).

Referenced by [L4Re::Util::Object_registry::unregister_obj\(\)](#).

Here is the caller graph for this function:



The documentation for this struct was generated from the following file:

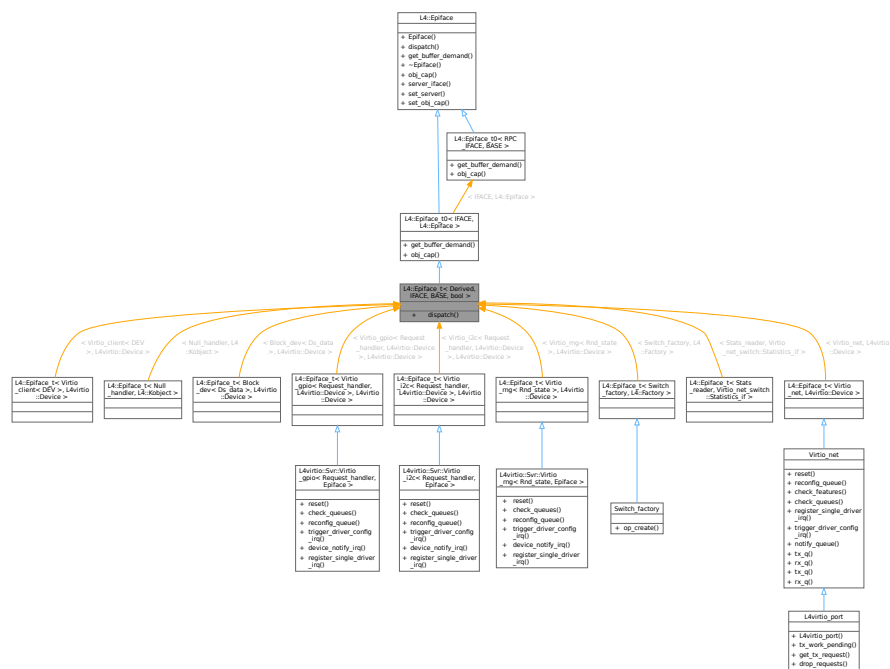
- [l4/sys/cxx/ipc_epiface](#)

16.100 L4::Epiface_t< Derived, IFACE, BASE, bool > Struct Template Reference

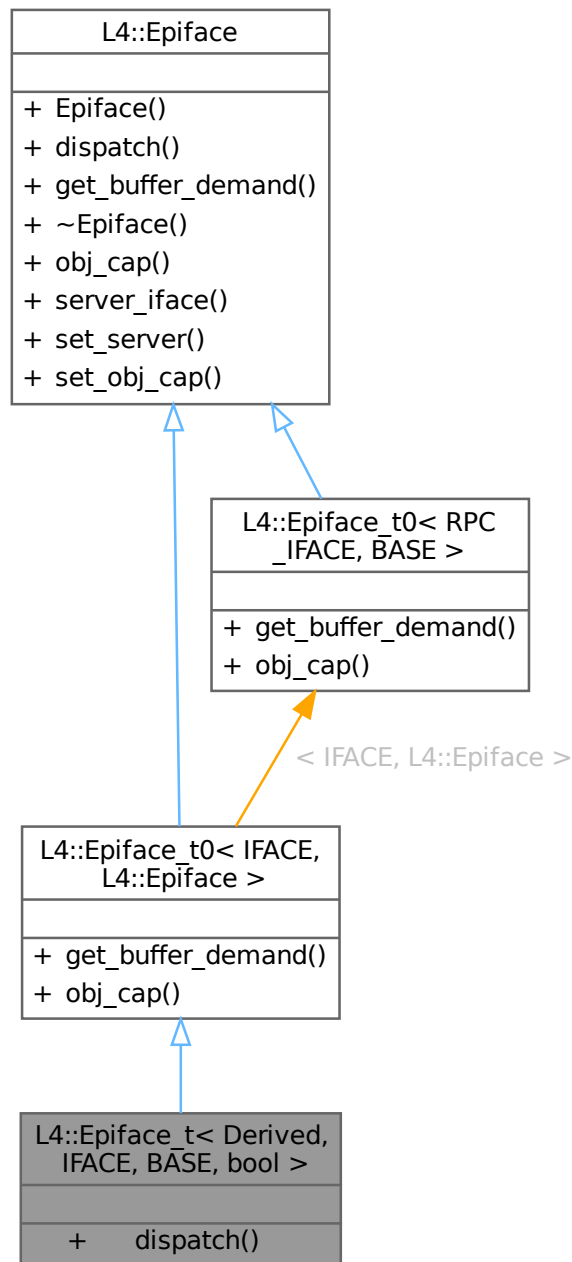
[Epiface](#) implementation for Kobject-based interface implementations.

```
#include <ipc_epiface>
```


Inheritance diagram for L4::Epiface_t< Derived, IFACE, BASE, bool >:



Collaboration diagram for L4::Epiface_t< Derived, IFACE, BASE, bool >:



Public Member Functions

- `l4_msgtag_t dispatch (l4_msgtag_t tag, unsigned rights, l4_utcb_t *utcb) final`
The abstract handler for client requests to the object.

Public Member Functions inherited from **L4::Epiface_t0< IFACE, L4::Epiface >**

- `Type_info::Demand get_buffer_demand () const`

Get the server-side buffer demand based in IFACE.

- [Cap](#)< IFACE > [obj_cap](#) () const

Get the (typed) capability to this object.

Public Member Functions inherited from [L4::Epiface](#)

- **Epiface** ()
Make a server object.
- virtual **~Epiface** ()=0
Destroy the object.
- Stored_cap [obj_cap](#) () const
Get the capability to the kernel object belonging to this object.
- [Server_iface](#) * [server_iface](#) () const
Get pointer to server interface at which the object is currently registered.
- int [set_server](#) ([Server_iface](#) *srv, [Cap](#)< void > cap, bool managed=false)
Set server registration info for the object.
- void **set_obj_cap** ([Cap](#)< void > const &cap)
Deprecated server registration function.

Additional Inherited Members

Public Types inherited from [L4::Epiface_t](#)< IFACE, [L4::Epiface](#) >

- typedef IFACE **Interface**
Data type of the IPC interface definition.

Public Types inherited from [L4::Epiface](#)

- typedef [lpc_svr::Server_iface](#) **Server_iface**
Type for abstract server interface.
- typedef [lpc_svr::Server_iface::Demand](#) **Demand**
Type for server-side receive buffer demand.

16.100.1 Detailed Description

```
template<typename Derived, typename IFACE, typename BASE = L4::Epiface, bool = cxx::is_↵
polymorphic<BASE>::value>
struct L4::Epiface_t< Derived, IFACE, BASE, bool >
```

[Epiface](#) implementation for Kobject-based interface implementations.

Template Parameters

<i>Derived</i>	Class providing the interface implementations.
<i>BASE</i>	Epiface base class.

Examples

[examples/clntsrv/src/server.cc](#).

Definition at line 503 of file [ipc_epiface](#).

16.100.2 Member Function Documentation

16.100.2.1 dispatch()

```
template<typename Derived, typename IFACE, typename BASE = L4::Epiface, bool = cxx::is_↵
polymorphic<BASE>::value>
l4_msgtag_t L4::Epiface_t< Derived, IFACE, BASE, bool >::dispatch (
    l4_msgtag_t tag,
    unsigned rights,
    l4_utcb_t * utcb) [inline], [final], [virtual]
```

The abstract handler for client requests to the object.

Parameters

<i>tag</i>	The message tag for this invocation.
<i>rights</i>	The rights bits in the invoked capability.
<i>utcb</i>	The UTCB used for the invocation.

Return values

<code>-L4_ENOREPLY</code>	No reply message is send.
<code>< 0</code>	Error, reply with error code.
<code>>= 0</code>	Success, reply with return value.

This function must be implemented by application specific server objects.

Implements [L4::Epiface](#).

Definition at line [506](#) of file [ipc_epiface](#).

The documentation for this struct was generated from the following file:

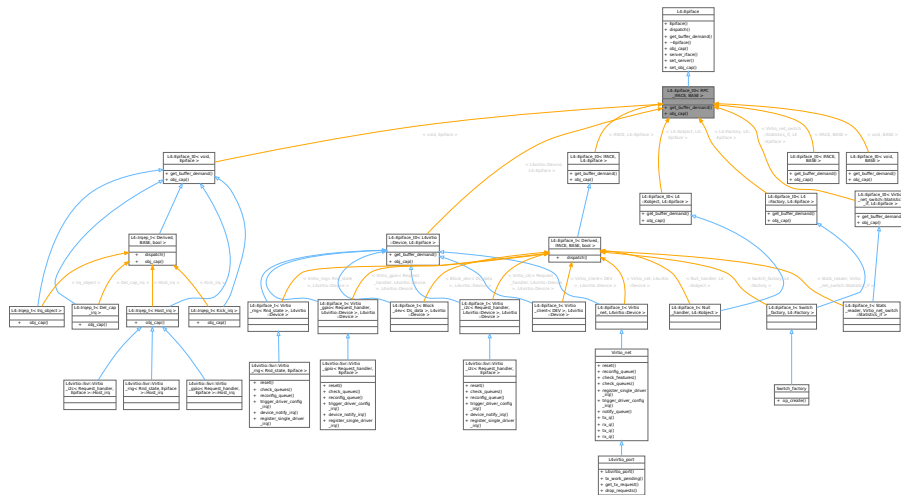
- `l4/sys/cxx/ipc_epiface`

16.101 L4::Epiface_t0< RPC_IFACE, BASE > Struct Template Reference

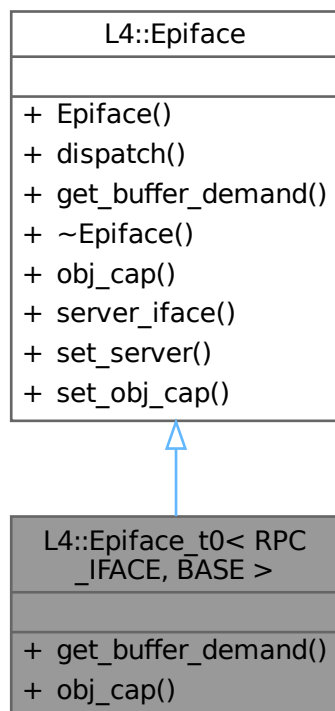
[Epiface](#) mixin for generic Kobject-based interfaces.

```
#include <ipc_epiface>
```

Inheritance diagram for L4::Epiface_t0< RPC_IFACE, BASE >:



Collaboration diagram for L4::Epiface_t0< RPC_IFACE, BASE >:



Public Types

- typedef **RPC_IFACE Interface**
Data type of the IPC interface definition.

Public Types inherited from [L4::Epiface](#)

- typedef [lpc_svr::Server_iface](#) **Server_iface**
Type for abstract server interface.
- typedef [lpc_svr::Server_iface::Demand](#) **Demand**
Type for server-side receive buffer demand.

Public Member Functions

- [Type_info::Demand](#) **get_buffer_demand** () const
Get the server-side buffer demand based in IFACE.
- [Cap](#)< [RPC_IFACE](#) > **obj_cap** () const
Get the (typed) capability to this object.

Public Member Functions inherited from [L4::Epiface](#)

- **Epiface** ()
Make a server object.
- virtual [l4_msgtag_t](#) **dispatch** ([l4_msgtag_t](#) tag, unsigned rights, [l4_utcb_t](#) *utcb)=0
The abstract handler for client requests to the object.
- virtual **~Epiface** ()=0
Destroy the object.
- Stored_cap **obj_cap** () const
Get the capability to the kernel object belonging to this object.
- [Server_iface](#) * **server_iface** () const
Get pointer to server interface at which the object is currently registered.
- int **set_server** ([Server_iface](#) *srv, [Cap](#)< void > cap, bool managed=false)
Set server registration info for the object.
- void **set_obj_cap** ([Cap](#)< void > const &cap)
Deprecated server registration function.

16.101.1 Detailed Description

```
template<typename RPC_IFACE, typename BASE = Epiface>
struct L4::Epiface_t0< RPC_IFACE, BASE >
```

[Epiface](#) mixin for generic Kobject-based interfaces.

Template Parameters

<i>RPC_IFACE</i>	Data type of the IPC interface definition.
<i>BASE</i>	Base Epiface class.

Definition at line 256 of file [ipc_epiface](#).

16.101.2 Member Function Documentation

16.101.2.1 obj_cap()

```
template<typename RPC_IFACE, typename BASE = Epiface>
Cap< RPC_IFACE > L4::Epiface_t0< RPC_IFACE, BASE >::obj_cap () const [inline]
```

Get the (typed) capability to this object.

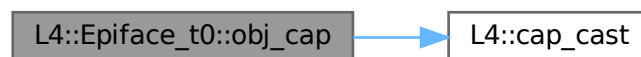
Returns

Capability for the kernel object behind the server.

Definition at line 269 of file [ipc_epiface](#).

References [L4::cap_cast\(\)](#).

Here is the call graph for this function:



The documentation for this struct was generated from the following file:

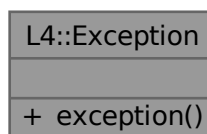
- `l4/sys/cxx/ipc_epiface`

16.102 L4::Exception Class Reference

[Exception](#) interface.

```
#include <exception>
```

Collaboration diagram for L4::Exception:



Public Member Functions

- [l4_msgtag_t exception](#) ([L4::lpc::In_out](#)< [l4_exc_regs_t](#) * > *regs*, [L4::lpc::Rcv_fpage](#) *rwin*, [L4::lpc::Opt](#)< [L4::lpc::Snd_fpage](#) & > *fp*)
Exception call.

16.102.1 Detailed Description

[Exception](#) interface.

This class defines the interface for handling exception IPC. When an exception occurs during program execution, for example due to a division by zero, the kernel will synthesise an exception IPC and send it to the thread's exception handler, who can then handle it.

The exception handler is set with the [L4::Thread::control](#) interface.

Definition at line 31 of file [exception](#).

16.102.2 Member Function Documentation

16.102.2.1 exception()

```
l4\_msgtag\_t L4::Exception::exception (
    L4::Ipc::In\_out< l4\_exc\_regs\_t * > regs,
    L4::Ipc::Rcv\_fpage rwin,
    L4::Ipc::Opt< L4::Ipc::Snd\_fpage & > fp)
```

[Exception](#) call.

Parameters

	<i>regs</i>	Register state of the faulting thread.
	<i>rwin</i>	Receive window in the address space.
out	<i>fp</i>	Optional flexpage to resolve the exception.

Returns

Message tag containing error code.

The documentation for this class was generated from the following file:

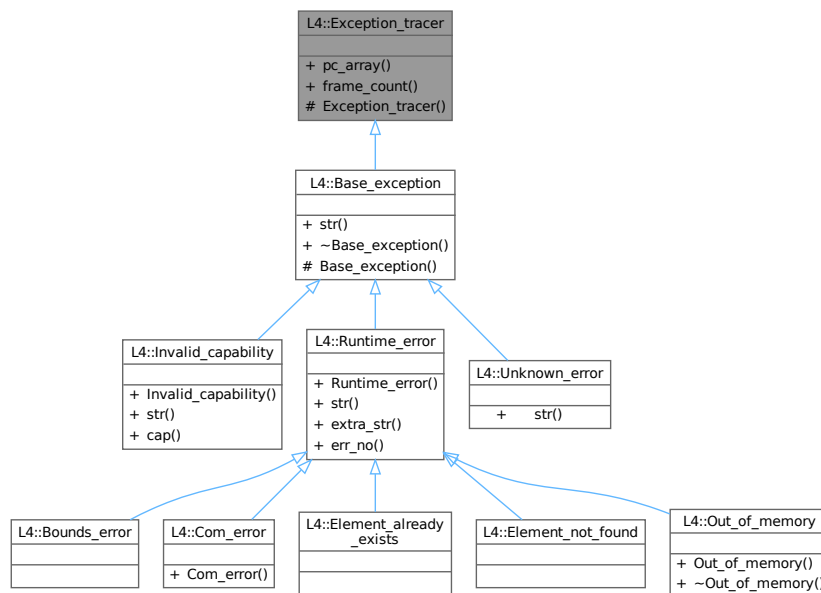
- [l4/sys/exception](#)

16.103 L4::Exception_tracer Class Reference

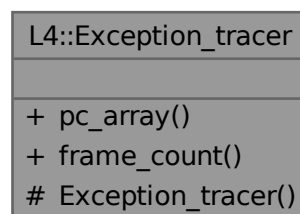
Back-trace support for exceptions.

```
#include <l4/cxx/exceptions>
```

Inheritance diagram for L4::Exception_tracer:



Collaboration diagram for L4::Exception_tracer:



Public Member Functions

- void const *const * **pc_array** () const noexcept
Get the array containing the call trace.
- int **frame_count** () const noexcept
Get the number of entries that are valid in the call trace.

Protected Member Functions

- **Exception_tracer** () noexcept
Create a back trace.

16.103.1 Detailed Description

Back-trace support for exceptions.

This class holds an array of at most [L4_CXX_EXCEPTION_BACKTRACE](#) instruction pointers containing the call trace at the instant when an exception was thrown.

Definition at line 51 of file [exceptions](#).

The documentation for this class was generated from the following file:

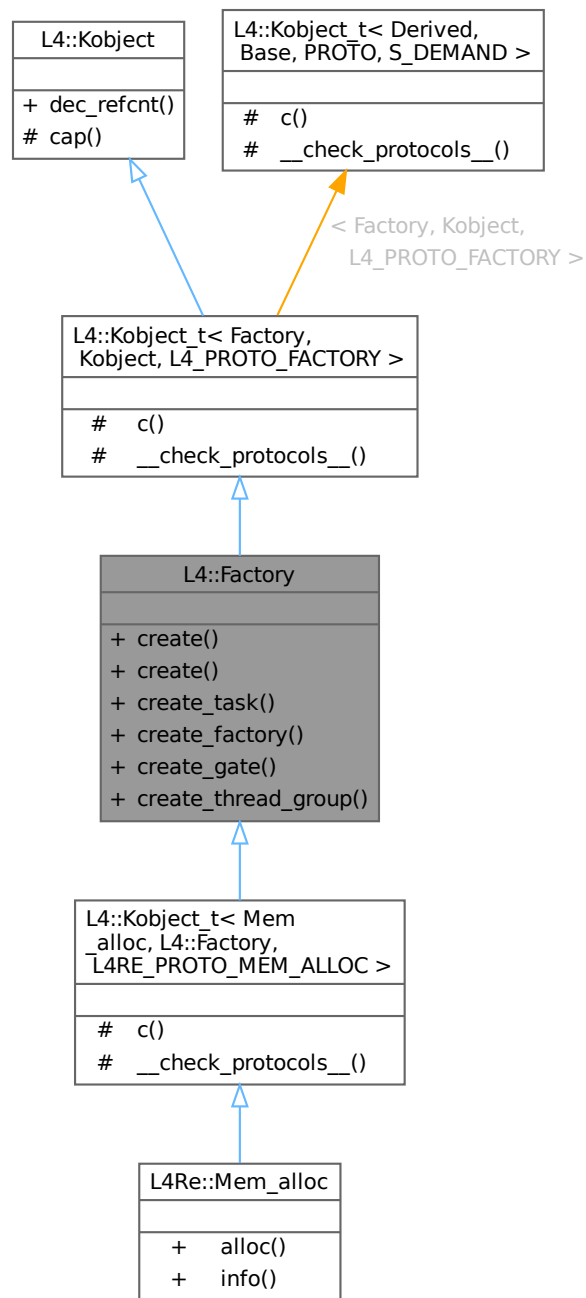
- [l4/cxx/exceptions](#)

16.104 L4::Factory Class Reference

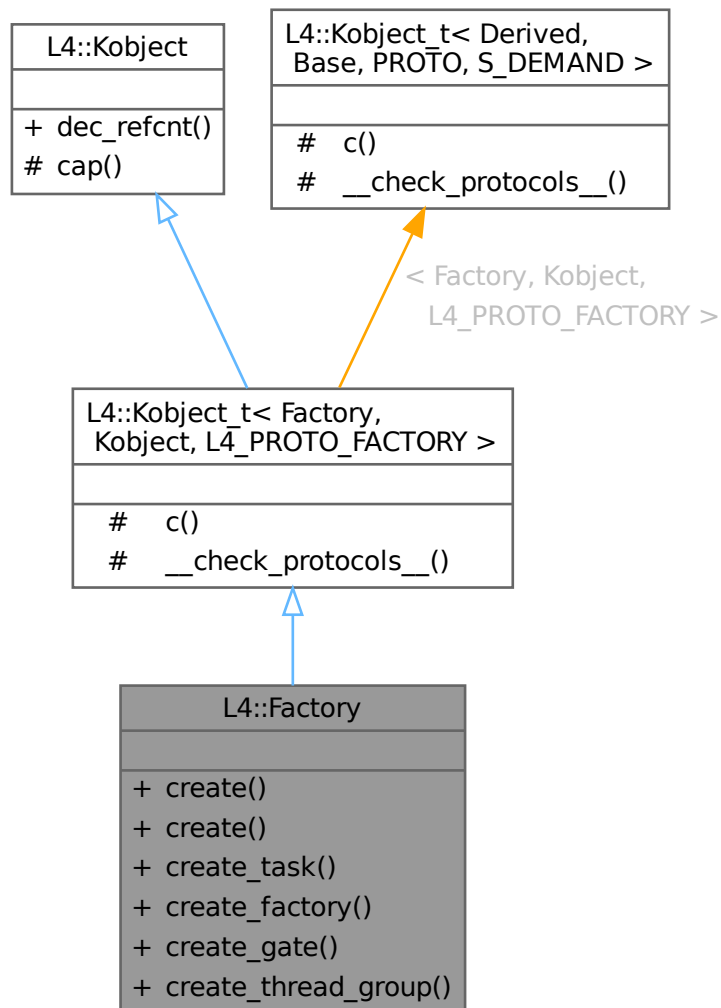
C++ Factory interface, see [Factory](#) for the C interface.

```
#include <factory>
```

Inheritance diagram for L4::Factory:



Collaboration diagram for L4::Factory:



Data Structures

- struct [Nil](#)
Special type to add a void argument into the factory create stream.
- struct [Lstr](#)
Special type to add a pascal string into the factory create stream.
- class [S](#)
Stream class for the [create\(\)](#) argument stream.

Public Member Functions

- [S create](#) ([Cap](#)< void > target, long obj, [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept
Generic create call to the factory.

- `template<typename OBJ>`
`S create (Cap< OBJ > target, l4_utcb_t *utcb=l4_utcb()) noexcept`
Create call for typed capabilities.
- `l4_msgtag_t create_task (Cap< Task > const &target_cap, l4_fpage_t *utcb_area, l4_utcb_t *utcb=l4_utcb()) noexcept`
Create a new task.
- `l4_msgtag_t create_factory (Cap< Factory > const &target_cap, unsigned long limit, l4_utcb_t *utcb=l4_utcb()) noexcept`
Create a new factory.
- `l4_msgtag_t create_gate (Cap< void > const &target_cap, Cap< Snd_destination > const &snd_dst_cap, l4_umword_t label, l4_utcb_t *utcb=l4_utcb()) noexcept`
Create a new IPC gate, optionally bound to a send destination (a thread or thread group).
- `l4_msgtag_t create_thread_group (Cap< Thread_group > const &target_cap, unsigned policy, l4_utcb_t *utcb=l4_utcb()) noexcept`
Create a new thread group.

Public Member Functions inherited from L4::Kobject

- `l4_msgtag_t dec_refcnt (l4_mword_t diff, l4_utcb_t *utcb=l4_utcb())`
Decrement the in kernel reference counter for the object.

Additional Inherited Members

Protected Types inherited from L4::Kobject_t< Factory, Kobject, L4_PROTO_FACTORY >

- `typedef Factory Class`
The target interface type (inheriting from Kobject_t).
- `typedef Typeid::Iface< PROTO, Factory > __iface`
The interface description for the derived class.
- `typedef Typeid::Merge_list< Typeid::Iface_list< __iface >, typename Kobject::__iface_list > __iface_list`
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from L4::Kobject_t< Factory, Kobject, L4_PROTO_FACTORY >

- `L4::Cap< Class > c () const noexcept`
Get the capability to ourselves.

Protected Member Functions inherited from L4::Kobject

- `l4_cap_idx_t cap () const noexcept`
Return capability selector.

Static Protected Member Functions inherited from L4::Kobject_t< Factory, Kobject, L4_PROTO_FACTORY >

- `static void __check_protocols__ () noexcept`
Helper to check for protocol conflicts.

16.104.1 Detailed Description

C++ Factory interface, see [Factory](#) for the C interface.

Factories provide an interface to create objects which are accessed via capabilities.

For additional information about which objects can be created via this interface, see server-specific information in [Kernel Factory](#) and [L4Re Servers](#).

Include File

```
#include <l4/sys/factory>
```

For the C interface refer to [Factory](#).

Definition at line 38 of file [factory](#).

16.104.2 Member Function Documentation

16.104.2.1 create() [1/2]

```
template<typename OBJ>
S L4::Factory::create (
    Cap< OBJ > target,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
```

Create call for typed capabilities.

Template Parameters

<i>OBJ</i>	Capability type of the object to be created.
------------	--

Parameters

out	<i>target</i>	Capability of type OBJ.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

A create stream that allows additional arguments to be passed to the `create()` call via the left-shift (`<<`) operator (see `#S::operator <<`).

This method does not directly invoke the factory. The factory is invoked when the create stream returned by this method is converted to an [l4_msgtag_t](#) (see `S::operator l4_msgtag_t()`), or otherwise when the stream goes out of scope (not recommended; see `#S::~~S()`).

Precondition

The invoked [Factory](#) capability must have the permission [L4_CAP_FPAGE_S](#), otherwise the later factory IPC will fail with [L4_EPERM](#) (see [S::operator l4_msgtag_t\(\)](#)).

Note

The create stream uses the UTCB to store parameters for the service call. During the lifetime of a create stream or, until it is converted to an [l4_msgtag_t](#), other UTCB-using operations must not be used.

Usage:

```
L4::Cap<L4Re::Dataspace> ds = L4Re::Util::cap_alloc.alloc<L4Re::Dataspace>();
factory->create(ds) << l4_mword_t(size_in_bytes);
```

Definition at line 329 of file [factory](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:

**16.104.2.2 create() [2/2]**

```
S L4::Factory::create (
    Cap< void > target,
    long obj,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
```

Generic create call to the factory.

Parameters

out	<i>target</i>	Capability selector for the new object. The caller must allocate the capability slot. The kernel stores the new objects's capability into this slot.
	<i>obj</i>	The protocol ID that specifies which kind of object shall be created.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

A create stream that allows additional arguments to be passed to the `create()` call via the left-shift (`<<`) operator (see [#S::operator <<](#)).

This method does not directly invoke the factory. The factory is invoked when the create stream returned by this method is converted to an [l4_msgtag_t](#) (see [S::operator l4_msgtag_t\(\)](#)), or otherwise when the stream goes out of scope (not recommended; see [#S::~~S\(\)](#)).

Precondition

The invoked [Factory](#) capability must have the permission [L4_CAP_FPAGE_S](#), otherwise the later factory IPC will fail with [L4_EPERM](#) (see [S::operator l4_msgtag_t\(\)](#)).

Note

The create stream uses the UTCB to store parameters for the service call. During the lifetime of a create stream or, until it is converted to an [l4_msgtag_t](#), other UTCB-using operations must not be used.

See also

[create\(Cap<OBJ>, l4_utcb_t *\)](#)

Definition at line [292](#) of file [factory](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:

**16.104.2.3 create_factory()**

```

l4_msgtag_t L4::Factory::create_factory (
    Cap< Factory > const & target_cap,
    unsigned long limit,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Create a new factory.

Parameters

out	<i>target_cap</i>	The kernel stores the new factory's capability into this slot.
	<i>limit</i>	Limit for the new factory in bytes.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

Syscall return tag

Return values

<code>L4_EOK</code>	No error occurred.
<code>-L4_EPERM</code>	Insufficient permissions; see precondition.
<code><0</code>	Error code.

Precondition

The invoked [Factory](#) capability must have the permission [L4_CAP_FPAGE_S](#).

Note

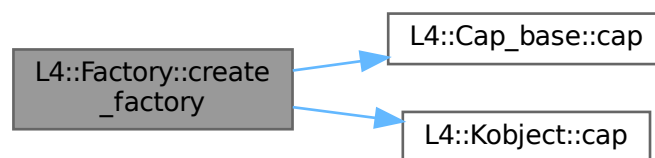
In addition to memory needed for internal data structures, the `limit` (quota) of the new factory is counted towards the quota of the creating factory. The `limit` must be within $1 \leq \text{limit} \leq 2^{(8 * \text{sizeof}(\text{l4_umword_t}) - 1) - 2}$ otherwise the behavior is undefined.

This method is only guaranteed to work with the [Kernel Factory](#). For other services, use the generic [create\(\)](#) method and consult the service documentation for information on the arguments that need to be passed to the create stream.

Definition at line [404](#) of file [factory](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:

**16.104.2.4 create_gate()**

```

l4_msgtag_t L4::Factory::create_gate (
    Cap< void > const & target_cap,
    Cap< Snd_destination > const & snd_dst_cap,
    l4_umword_t label,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Create a new IPC gate, optionally bound to a send destination (a thread or thread group).

Parameters

out	<code>target_cap</code>	The kernel stores the new IPC gate's capability into this slot.
-----	-------------------------	---

	<i>snd_dst_cap</i>	Optional capability selector of a thread or thread group to bind the gate to. Use L4_INVALID_CAP to create an unbound IPC gate.
	<i>label</i>	Optional label of the gate (precisely used if <i>snd_dst_cap</i> is valid). If <i>snd_dst_cap</i> is valid, <i>label</i> must be present.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

Syscall return tag containing one of the following return codes.

Return values

<i>L4_EOK</i>	No error occurred.
<i>-L4_ENOMEM</i>	Out-of-memory during allocation of the lpc_gate object.
<i>-L4_EINVAL</i>	<i>snd_dst_cap</i> is void or points to something that is not a thread or thread group.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.

Precondition

The invoked [Factory](#) capability must have the permission [L4_CAP_FPAGE_S](#). Also *snd_dst_cap* (if *valid*) must have the permission [L4_CAP_FPAGE_S](#).

An unbound IPC gate can be bound to a thread or thread group using [L4::lpc_gate::bind_thread\(\)](#) or [bind_snd_destination\(\)](#).

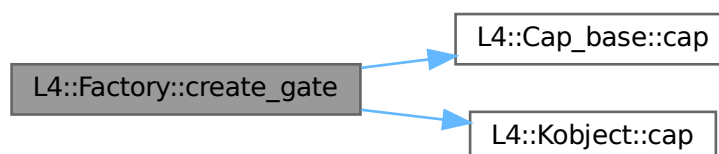
See also

[L4::lpc_gate](#)

Definition at line 440 of file [factory](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



16.104.2.5 create_task()

```
l4_msgtag_t L4::Factory::create_task (
    Cap< Task > const & target_cap,
    l4_fpage_t * utcb_area,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
```

Create a new task.

Parameters

out	<i>target_cap</i>	The kernel stores the new task's capability into this slot.
in, out	<i>utcb_area</i>	Flexpage that describes an area in the address space of the new task, where the kernel should map the kernel-allocated kernel-user memory to. The kernel uses the kernel-user memory to store UTCBs and vCPU state-save-areas of the new task.

On systems without MMU, the flexpage is adjusted to reflect the actually allocated physical address.

Parameters

<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .
-------------	--

Returns

Syscall return tag

Return values

<i>L4_EOK</i>	No error occurred.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i><0</i>	Error code.

Precondition

The invoked [Factory](#) capability must have the permission [L4_CAP_FPAGE_S](#).

Note

The size of the UTCB area specifies indirectly the number of UTCBs available for this task. Refer to [L4::Task::add_ku_mem](#) for adding more of this type of memory.

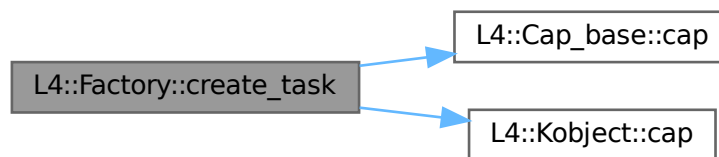
See also

[L4::Task](#)

Definition at line 370 of file [factory](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



16.104.2.6 create_thread_group()

```

l4_msgtag_t L4::Factory::create_thread_group (
    Cap< Thread_group > const & target_cap,
    unsigned policy,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Create a new thread group.

An IPC endpoint can be bound to a thread group. When a message arrives at the IPC endpoint, a specific thread of the thread group is selected to actually receive the message. A thread group is a send destination for an IPC endpoint.

Parameters

out	<i>target_cap</i>	The kernel stores the new thread group's capability into this slot.
	<i>policy</i>	Policy parameter for the thread group. See L4_thread_group_policy for a list of supported values.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

Syscall return tag containing one of the following return codes.

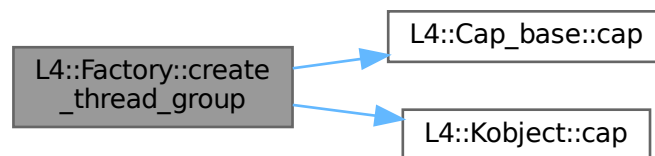
Return values

<i>L4_EOK</i>	No error occurred.
<i>-L4_ENOMEM</i>	Out-of-memory during allocation of the Thread_group object.
<i>-L4_EINVAL</i>	Invalid policy parameter.
<i>-L4_EPERM</i>	The factory instance requires L4_CAP_FPAGE_S rights on the invoked capability and L4_CAP_FPAGE_S is not present.

Definition at line 475 of file [factory](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

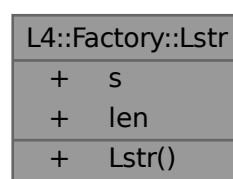
- [l4/sys/factory](#)

16.105 L4::Factory::Lstr Struct Reference

Special type to add a pascal string into the factory create stream.

```
#include <factory>
```

Collaboration diagram for L4::Factory::Lstr:



Public Member Functions

- [Lstr](#) (char const *[s](#), unsigned [len](#)) noexcept

Data Fields

- char const * **s**
The character buffer.
- unsigned **len**
The number of characters in the buffer.

16.105.1 Detailed Description

Special type to add a pascal string into the factory create stream.

This encapsulates a string that has an explicit length.

Definition at line [54](#) of file [factory](#).

16.105.2 Constructor & Destructor Documentation

16.105.2.1 Lstr()

```
L4::Factory::Lstr::Lstr (  
    char const * s,  
    unsigned len) [inline], [noexcept]
```

Parameters

<i>s</i>	Pointer to the c-style string.
<i>len</i>	Length in number of characters of the string <i>s</i> .

Definition at line [70](#) of file [factory](#).

References [len](#), and [s](#).

The documentation for this struct was generated from the following file:

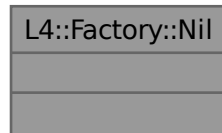
- [l4/sys/factory](#)

16.106 L4::Factory::Nil Struct Reference

Special type to add a void argument into the factory create stream.

```
#include <factory>
```

Collaboration diagram for L4::Factory::Nil:



16.106.1 Detailed Description

Special type to add a void argument into the factory create stream.

Definition at line 47 of file [factory](#).

The documentation for this struct was generated from the following file:

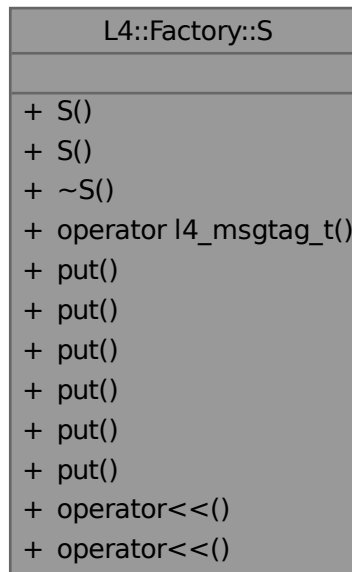
- [l4/sys/factory](#)

16.107 L4::Factory::S Class Reference

Stream class for the [create\(\)](#) argument stream.

```
#include <factory>
```

Collaboration diagram for L4::Factory::S:



Public Member Functions

- `S (S &&o) noexcept`
Move constructor.
- `S (l4_cap_idx_t f, long obj, L4::Cap< void > target, l4_utcb_t *utcb) noexcept`
Create a stream for a specific `create()` call.
- `~S () noexcept`
Commit the `create()` operation if not already done explicitly via `operator l4_msgtag_t()`.
- `operator l4_msgtag_t () noexcept`
Explicitly commits the operation and returns the result.
- `void put (l4_mword_t i) noexcept`
Put a single `l4_mword_t` as next argument.
- `void put (l4_umword_t i) noexcept`
Put a single `l4_umword_t` as next argument.
- `void put (char const *s) &noexcept`
Add a zero-terminated string as next argument.
- `void put (Lstr const &s) &noexcept`
Add a pascal string as next argument.
- `void put (Nil) &noexcept`
Add an empty argument.
- `void put (l4_fpage_t d) &noexcept`
Add a flexpage as next argument.
- `template<typename T>`
`S & operator<< (T const &d) &noexcept`
Add next argument.
- `template<typename T>`
`S && operator<< (T const &d) &&noexcept`
Add next argument.

16.107.1 Detailed Description

Stream class for the [create\(\)](#) argument stream.

This stream allows a variable number of arguments to be added to a [create\(\)](#) call.

Definition at line 79 of file [factory](#).

16.107.2 Constructor & Destructor Documentation

16.107.2.1 S() [1/2]

```
L4::Factory::S::S (
    S && o) [inline], [noexcept]
```

Move constructor.

Parameters

<i>o</i>	Instance of S to move.
----------	--

Definition at line 98 of file [factory](#).

16.107.2.2 S() [2/2]

```
L4::Factory::S::S (
    l4_cap_idx_t f,
    long obj,
    L4::Cap< void > target,
    l4_utcb_t * utcb) [inline], [noexcept]
```

Create a stream for a specific [create\(\)](#) call.

Parameters

	<i>f</i>	The capability for the factory object (L4::Factory).
	<i>obj</i>	The protocol ID to describe the type of the object that shall be created.
out	<i>target</i>	The capability selector for the new object. The caller must allocate the capability slot. The kernel stores the new object's capability into this slot.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Precondition

The capability *f* must have the permission [L4_CAP_FPAGE_S](#), otherwise the later factory IPC will fail with [L4_EPERM](#).

Definition at line 125 of file [factory](#).

16.107.2.3 ~S()

```
L4::Factory::S::~~S () [inline], [noexcept]
```

Commit the [create\(\)](#) operation if not already done explicitly via [operator l4_msgtag_t\(\)](#).

Warning

If the commit is deferred until destruction, potential errors are silently ignored. It is therefore recommended to commit explicitly via [operator l4_msgtag_t\(\)](#) and check the return value.

Definition at line 139 of file [factory](#).

16.107.3 Member Function Documentation

16.107.3.1 operator l4_msgtag_t()

```
L4::Factory::S::operator l4_msgtag_t () [inline], [noexcept]
```

Explicitly commits the operation and returns the result.

Returns

The result of the [create\(\)](#) operation.

Return values

<i>L4_EOK</i>	No error occurred.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i><0</i>	Error code.

Precondition

The invoked [Factory](#) capability must have the permission [L4_CAP_FPAGE_S](#).

Definition at line 157 of file [factory](#).

16.107.3.2 operator<<() [1/2]

```
template<typename T>
S && L4::Factory::S::operator<< (
    T const & d) && [inline], [noexcept]
```

Add next argument.

Template Parameters

<i>T</i>	The argument type. Compilation succeeds only if it is a possible argument type for <code>S::put()</code> .
----------	--

Parameters

<i>d</i>	The value to add as next argument.
----------	------------------------------------

Definition at line 252 of file [factory](#).

References [put\(\)](#).

Here is the call graph for this function:



16.107.3.3 operator<<() [2/2]

```

template<typename T>
S & L4::Factory::S::operator<< (
    T const & d) & [inline], [noexcept]
  
```

Add next argument.

Template Parameters

<i>T</i>	The argument type. Compilation succeeds only if it is a possible argument type for <code>S::put()</code> .
----------	--

Parameters

<i>d</i>	The value to add as next argument.
----------	------------------------------------

Definition at line 237 of file [factory](#).

References [put\(\)](#).

Here is the call graph for this function:



16.107.3.4 put() [1/5]

```
void L4::Factory::S::put (
    char const * s) & [inline], [noexcept]
```

Add a zero-terminated string as next argument.

Parameters

<i>s</i>	The string to add as next argument.
----------	-------------------------------------

The string will be added with the zero-terminator.

Definition at line 191 of file [factory](#).

16.107.3.5 put() [2/5]

```
void L4::Factory::S::put (
    l4_fpage_t d) & [inline], [noexcept]
```

Add a flexpage as next argument.

Parameters

<i>d</i>	The flexpage to add (there will be no map operation).
----------	---

Definition at line 223 of file [factory](#).

16.107.3.6 put() [3/5]

```
void L4::Factory::S::put (
    l4_mword_t i) [inline], [noexcept]
```

Put a single [l4_mword_t](#) as next argument.

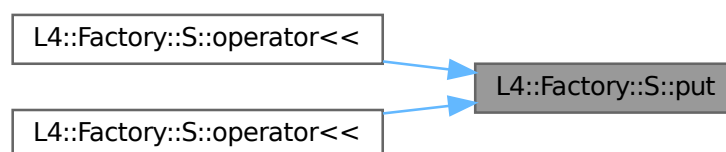
Parameters

<i>i</i>	The value to add as next argument.
----------	------------------------------------

Definition at line 169 of file [factory](#).

Referenced by [operator<<\(\)](#), and [operator<<\(\)](#).

Here is the caller graph for this function:



16.107.3.7 put() [4/5]

```
void L4::Factory::S::put (
    l4_umword_t i) [inline], [noexcept]
```

Put a single [l4_umword_t](#) as next argument.

Parameters

<i>i</i>	The value to add as next argument.
----------	------------------------------------

Definition at line [179](#) of file [factory](#).

16.107.3.8 put() [5/5]

```
void L4::Factory::S::put (
    Lstr const & s) & [inline], [noexcept]
```

Add a pascal string as next argument.

Parameters

<i>s</i>	The string to add as next argument.
----------	-------------------------------------

The string will be added with the exact length given. It is the responsibility of the caller to make sure that the string is zero- terminated when that is required by the server.

Definition at line [205](#) of file [factory](#).

The documentation for this class was generated from the following file:

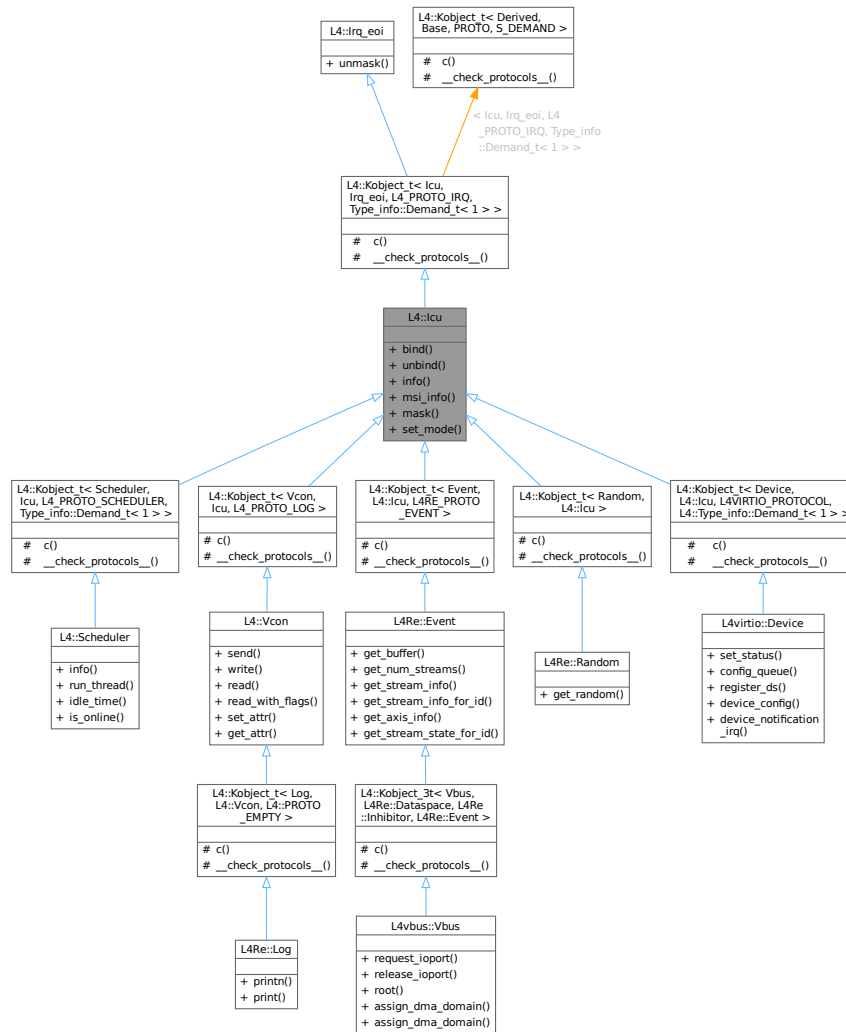
- [l4/sys/factory](#)

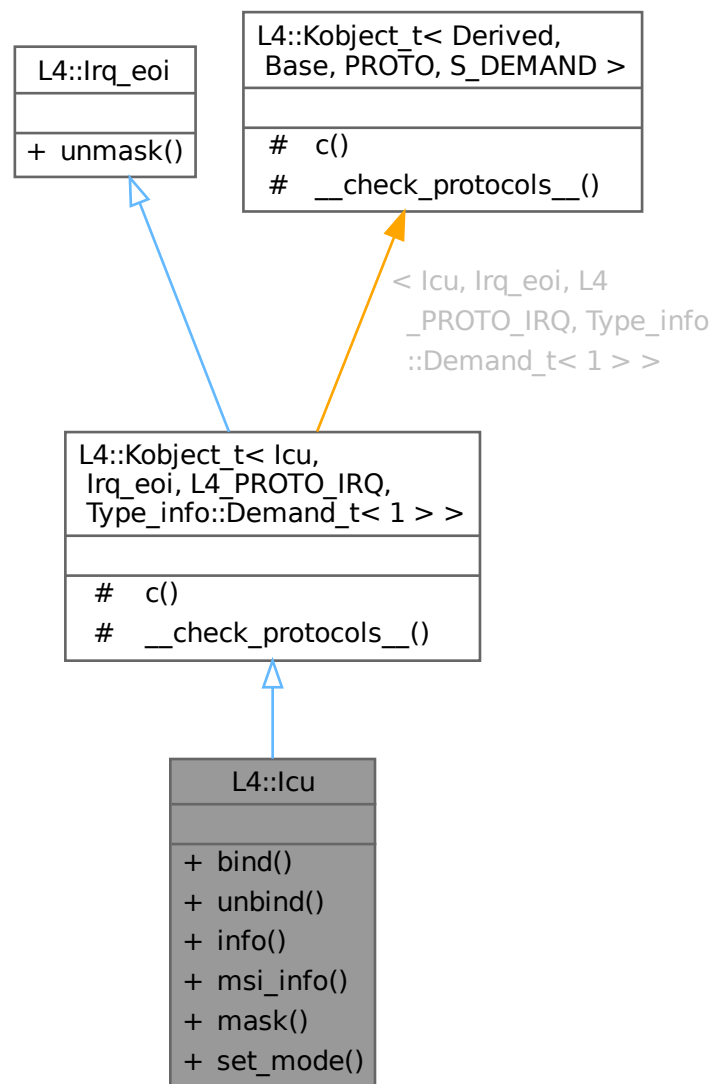
16.108 L4::lcu Class Reference

C++ [lcu](#) interface, see [Interrupt controller](#) for the C interface.

```
#include <irq>
```

Inheritance diagram for L4::Icu:





Data Structures

- class Info

This class encapsulates information about an ICU.

Public Member Functions

- `l4_msgtag_t bind` (unsigned irqnum, `L4::Cap< Triggerable > irq`, `l4_utcb_t *utcb=l4_utcb()`) noexcept
Bind an interrupt line of an interrupt controller to an interrupt object.
- `l4_msgtag_t unbind` (unsigned irqnum, `L4::Cap< Triggerable > irq`, `l4_utcb_t *utcb=l4_utcb()`) noexcept
Remove binding of an interrupt line from the interrupt controller object.

- `l4_msgtag_t info` (`l4_icu_info_t *info`, `l4_utcb_t *utcb=l4_utcb()`) noexcept
Get information about the ICU features.
- `l4_msgtag_t msi_info` (`l4_umword_t irqnum`, `l4_uint64_t source`, `l4_icu_msi_info_t *msi_info`)
Get MSI info about IRQ.
- `l4_msgtag_t mask` (`unsigned irqnum`, `l4_umword_t *label=0`, `l4_timeout_t to=L4_IPC_NEVER`, `l4_utcb_t *utcb=l4_utcb()`) noexcept
Mask an IRQ line.
- `l4_msgtag_t set_mode` (`unsigned irqnum`, `l4_umword_t mode`, `l4_utcb_t *utcb=l4_utcb()`) noexcept
Set interrupt mode.

Public Member Functions inherited from `L4::Irq_eoi`

- `l4_msgtag_t unmask` (`unsigned irqnum`, `l4_umword_t *label=0`, `l4_timeout_t to=L4_IPC_NEVER`, `l4_utcb_t *utcb=l4_utcb()`) noexcept
Unmask the given interrupt line.

Additional Inherited Members

Protected Types inherited from

`L4::Kobject_t< Icu, Irq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > >`

- typedef `Icu Class`
The target interface type (inheriting from `Kobject_t`).
- typedef `Typeid::Iface< PROTO, Icu > __Iface`
The interface description for the derived class.
- typedef `Typeid::Merge_list< Typeid::Iface_list< __Iface >, typename Irq_eoi::__Iface_list > __Iface_list`
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from

`L4::Kobject_t< Icu, Irq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > >`

- `L4::Cap< Class > c` () const noexcept
Get the capability to ourselves.

Static Protected Member Functions inherited from

`L4::Kobject_t< Icu, Irq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > >`

- static void `__check_protocols` () noexcept
Helper to check for protocol conflicts.

16.108.1 Detailed Description

C++ [Icu](#) interface, see [Interrupt controller](#) for the C interface.

Note

"ICU" is short for "interrupt control unit".

This class defines the interface for interrupt controllers. It defines functions for binding [L4::Irq](#) objects to interrupt lines and other interrupt sources, as well as functions for masking and unmasking of interrupts.

To setup an interrupt line the following steps are required:

1. [set_mode\(\)](#) (optional if interrupt has a default mode)
2. [L4::Rcv_endpoint::bind_thread\(\)](#) or [L4::Rcv_endpoint::bind_snd_destination\(\)](#) to attach the [L4::Irq](#) object to a send destination.
3. [bind\(\)](#)
4. [unmask\(\)](#) to receive the first interrupt

For certain interrupt sources only some of these steps are necessary and supported, see [L4::Scheduler](#) and [L4::Vcon](#).

At most one [L4::Irq](#) object can be bound to a certain interrupt source and a certain [L4::Irq](#) object can be bound to at most one interrupt source.

Include File

```
#include <l4/sys/icu>
```

Definition at line 250 of file [irq](#).

16.108.2 Member Function Documentation

16.108.2.1 bind()

```
l4_msgtag_t L4::Icu::bind (
    unsigned irqnum,
    L4::Cap< Triggerable > irq,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
```

Bind an interrupt line of an interrupt controller to an interrupt object.

Parameters

<i>irqnum</i>	IRQ line at the ICU.
<i>irq</i>	IRQ object for the given IRQ line to bind to this ICU.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Returns

Syscall return tag. The caller should check the return value using [l4_error\(\)](#) to check for errors and to identify the correct method for unmasking the interrupt. Return values < 0 indicate an error. A return value of 0 means a direct unmask via the IRQ object using [L4::Irq::unmask](#). A return value of 1 means that the interrupt has to be unmasked via the ICU using [L4::Icu::unmask](#).

Return values

-L4_EINVAL	<code>irq</code> is bound to an interrupt source.
-L4_EPERM	Insufficient permissions; see precondition.

Precondition

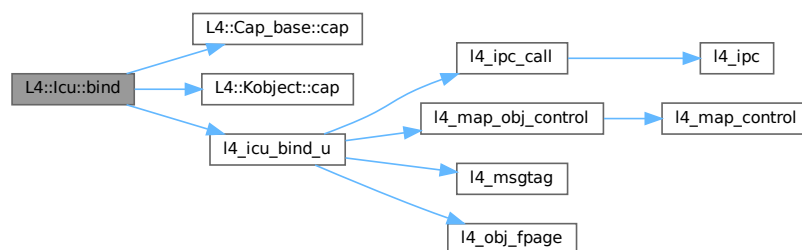
The capability `irq` must have the permission [L4_CAP_FPAGE_W](#).

In case the `irq` is already bound to an interrupt source, it is unbound first. In case the `irq` is bound and the interrupt source is bound to a different [L4::Irq](#) object, only the unbinding happens. An [L4::Irq](#) object that is bound to an interrupt source will get unbound if the [L4::Irq](#) object is deleted.

Definition at line 310 of file [irq](#).

References [L4::Cap_base::cap\(\)](#), [L4::Kobject::cap\(\)](#), and [l4_icu_bind_u\(\)](#).

Here is the call graph for this function:



16.108.2.2 info()

```

l4_msgtag_t L4::Icu::info (
    l4_icu_info_t * info,
    l4_utcb_t * utcb = l4_utcb())  [inline], [noexcept]

```

Get information about the ICU features.

Parameters

out	info	Info structure to be filled with information.
	utcb	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Returns

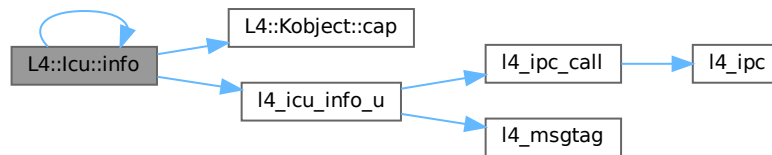
Syscall return tag

Definition at line 345 of file [irq](#).

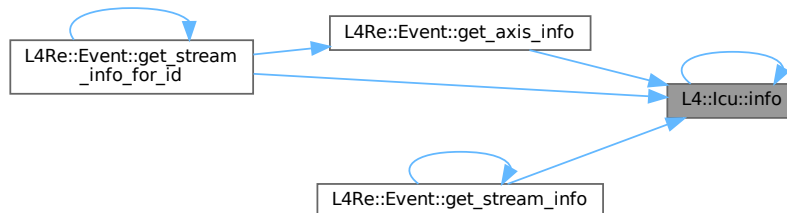
References [L4::Kobject::cap\(\)](#), [info\(\)](#), and [l4_icu_info_u\(\)](#).

Referenced by [L4Re::Event::get_axis_info\(\)](#), [L4Re::Event::get_stream_info\(\)](#), [L4Re::Event::get_stream_info_for_id\(\)](#), and [info\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.108.2.3 mask()

```

l4_msgtag_t L4::Icu::mask (
    unsigned irqnum,
    l4_umword_t * label = 0,
    l4_timeout_t to = L4_IPC_NEVER,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Mask an IRQ line.

Parameters

<i>irqnum</i>	IRQ line at the ICU.
---------------	----------------------

<i>label</i>	If NULL, this function is a send-only message to the ICU. If not NULL, this function will enter an open wait after sending the mask message and the received label is returned here.
<i>to</i>	The timeout-pair (send and receive) that shall be used for this operation. The receive timeout is used with a non-NULL <code>label</code> only.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

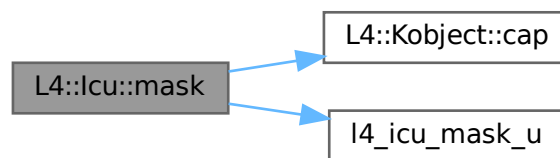
Returns

Syscall return tag. If `label` is NULL, this function performs an IPC send-only operation and there is no return value except [L4_MSGTAG_ERROR](#) indicating success or failure of the send operation. In this case use [l4_ipc_error\(\)](#) to check for errors and **do not** use [l4_error\(\)](#).

Definition at line 393 of file [irq](#).

References [L4::Kobject::cap\(\)](#), [l4_icu_mask_u\(\)](#), and [L4_IPC_NEVER](#).

Here is the call graph for this function:



16.108.2.4 msi_info()

```

l4_msgtag_t L4::Icu::msi_info (
    l4_umword_t irqnum,
    l4_uint64_t source,
    l4_icu_msi_info_t * msi_info)
  
```

Get MSI info about IRQ.

Parameters

	<i>irqnum</i>	IRQ line at the ICU.
	<i>source</i>	Platform dependent requester ID for MSIs. On IA32 we use a 20bit source filter value as described in the Intel IRQ remapping specification.
out	<i>msi_info</i>	A l4_icu_msi_info_t structure receiving the address and the data value to trigger this MSI.

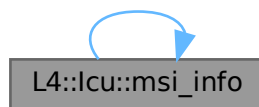
Returns

Syscall return tag

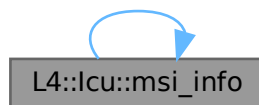
References [msi_info\(\)](#).

Referenced by [msi_info\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**16.108.2.5 set_mode()**

```
l4_msgtag_t L4::Icu::set_mode (  
    unsigned irqnum,  
    l4_unword_t mode,  
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
```

Set interrupt mode.

Parameters

<i>irqnum</i>	IRQ line at the ICU.
<i>mode</i>	Mode, see L4_irq_mode .
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

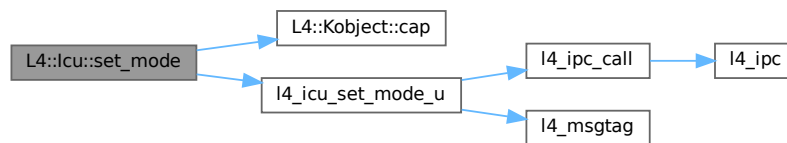
Returns

Syscall return tag

Definition at line 421 of file [irq](#).

References [L4::Kobject::cap\(\)](#), and [l4_icu_set_mode_u\(\)](#).

Here is the call graph for this function:

**16.108.2.6 unbind()**

```

l4_msgtag_t L4::Icu::unbind (
    unsigned irqnum,
    L4::Cap< Triggerable > irq,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Remove binding of an interrupt line from the interrupt controller object.

Parameters

<i>irqnum</i>	IRQ line at the ICU.
<i>irq</i>	IRQ object to remove from the ICU.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

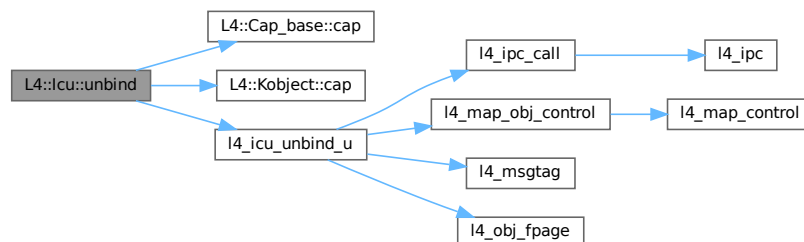
Returns

Syscall return tag

Definition at line 328 of file [irq](#).

References [L4::Cap_base::cap\(\)](#), [L4::Kobject::cap\(\)](#), and [l4_icu_unbind_u\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

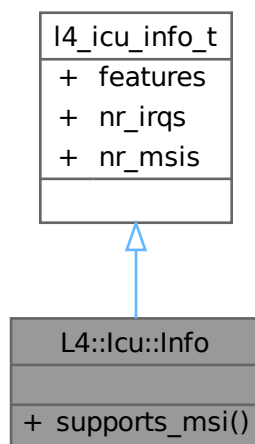
- [l4/sys/irq](#)

16.109 L4::lcu::Info Class Reference

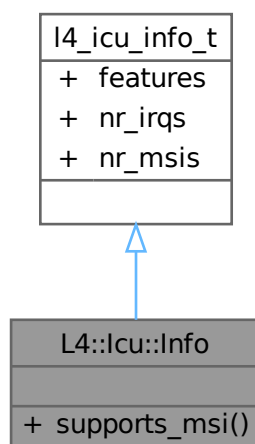
This class encapsulates information about an ICU.

```
#include <irq>
```

Inheritance diagram for L4::lcu::Info:



Collaboration diagram for L4::lcu::Info:



Public Member Functions

- bool **supports_msi** () const noexcept
True, if the ICU has support for MSIs.

Additional Inherited Members

Data Fields inherited from [l4_icu_info_t](#)

- unsigned [features](#)
Feature flags.
- unsigned **nr_irqs**
The number of IRQ lines supported by the ICU,.
- unsigned **nr_msis**
The number of MSI vectors supported by the ICU,.

16.109.1 Detailed Description

This class encapsulates information about an ICU.

Definition at line [277](#) of file [irq](#).

The documentation for this class was generated from the following file:

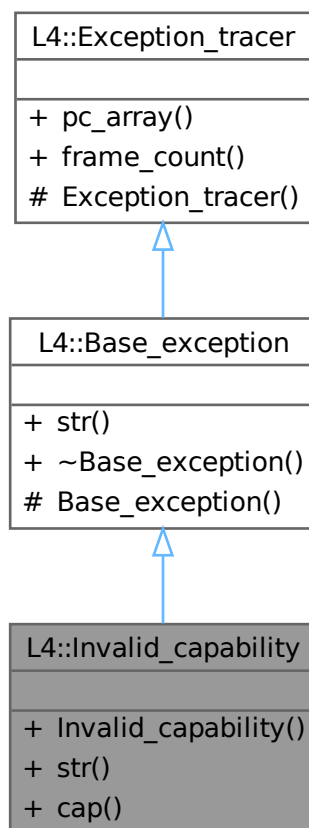
- [l4/sys/irq](#)

16.110 L4::Invalid_capability Class Reference

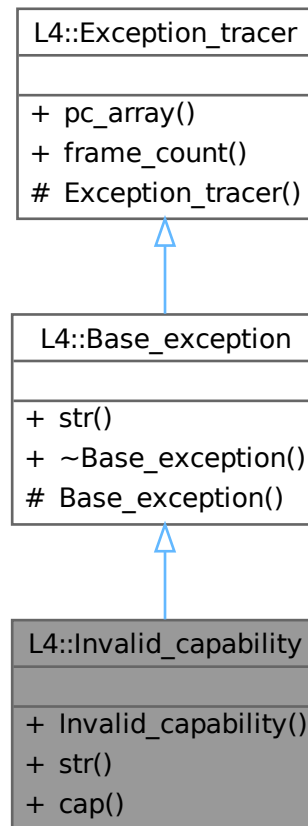
Indicates that an invalid object was invoked.

```
#include <l4/cxx/exceptions>
```


Inheritance diagram for L4::Invalid_capability:



Collaboration diagram for L4::Invalid_capability:



Public Member Functions

- `Invalid_capability (Cap< void > const &o) noexcept`
Create an *Invalid_object* exception for the Object *o*.
- `char const * str () const noexcept` override
Return a human readable string for the exception.
- `Cap< void > const & cap () const noexcept`
Get the object that caused the error.

Public Member Functions inherited from `L4::Base_exception`

- `virtual ~Base_exception () noexcept`
Destruction.

Public Member Functions inherited from `L4::Exception_tracer`

- `void const *const * pc_array () const noexcept`
Get the array containing the call trace.
- `int frame_count () const noexcept`
Get the number of entries that are valid in the call trace.

Additional Inherited Members

Protected Member Functions inherited from [L4::Base_exception](#)

- **Base_exception** () noexcept
Create a base exception.

Protected Member Functions inherited from [L4::Exception_tracer](#)

- **Exception_tracer** () noexcept
Create a back trace.

16.110.1 Detailed Description

Indicates that an invalid object was invoked.

An Object is invalid if it has L4_INVALID_ID as server [L4](#) UID, or if the server does not know the object ID.

Definition at line [234](#) of file [exceptions](#).

16.110.2 Constructor & Destructor Documentation

16.110.2.1 Invalid_capability()

```
L4::Invalid_capability::Invalid_capability (
    Cap< void > const & o) [inline], [explicit], [noexcept]
```

Create an Invalid_object exception for the Object o.

Parameters

<i>o</i>	The object that caused the server side error.
----------	---

Definition at line [244](#) of file [exceptions](#).

16.110.3 Member Function Documentation

16.110.3.1 cap()

```
Cap< void > const & L4::Invalid_capability::cap () const [inline], [noexcept]
```

Get the object that caused the error.

Returns

The object that caused the error on invocation.

Definition at line [253](#) of file [exceptions](#).

The documentation for this class was generated from the following file:

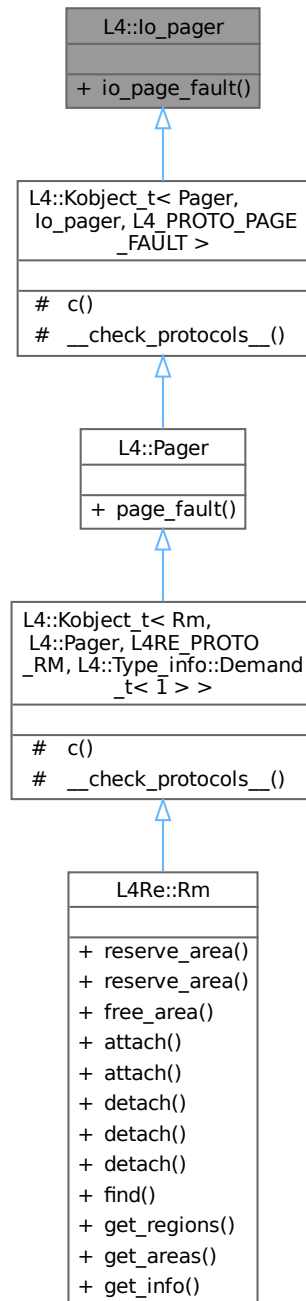
- [l4/cxx/exceptions](#)

16.111 L4::lo_pager Class Reference

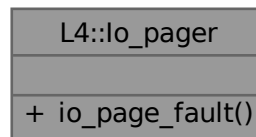
[lo_pager](#) interface.

```
#include <pager>
```

Inheritance diagram for L4::lo_pager:



Collaboration diagram for L4::io_pager:



Public Member Functions

- [l4_msgtag_t](#) [io_page_fault](#) ([l4_fpage_t](#) io_pfa, [l4_umword_t](#) pc, [L4::lpc::Rcv_fpage](#) rwin, [L4::lpc::Opt<L4::lpc::Snd_fpage](#) & > fp)

IO page fault protocol message.

16.111.1 Detailed Description

[io_pager](#) interface.

Note

This interface is IA32 specific.

This class defines the interface for handling IO page faults. IO page faults happen when a thread tries to access an IO port that it does not currently have access to.

Depending on the microkernel's implementation, IO page faults can be handled in two ways.

If the microkernel does not support IO page faults, this IO pagefault interface is not used. Instead, the microkernel sends an exception IPC to the thread's exception handler ([L4::Exception](#)), indicating a GP (exception number 13). The exception handler must consult the faulting instruction to determine the cause of the exception. This is the default in Fiasco.OC.

In contrast, if the microkernel supports IO page faults, the microkernel will generate an IO page fault message and send it to the thread's page fault handler (pager). The page fault handler can implement this interface to handle the IO page faults.

Note

A program may use this mechanism to implement a lazy IO port access scheme.

The page fault and exception handlers are set with the [L4::Thread::control](#) interface.

Definition at line 50 of file [pager](#).

16.111.2 Member Function Documentation

16.111.2.1 io_page_fault()

```
l4_msgtag_t L4::Io_pager::io_page_fault (
    l4_fpage_t io_pfa,
    l4_umword_t pc,
    L4::Ipc::Rcv_fpage rwin,
    L4::Ipc::Opt< L4::Ipc::Snd_fpage & > fp)
```

IO page fault protocol message.

Parameters

	<i>io_pfa</i>	Flexpage describing the faulting IO-port.
	<i>pc</i>	Faulting program counter.
	<i>rwin</i>	The receive window for a flexpage mapping.
out	<i>fp</i>	Optional: flexpage descriptor to send to the task raising the page fault.

Returns

System call message tag; use [l4_error\(\)](#) to check for errors.

IO-port fault messages are usually generated by the kernel and an IO-page-fault handler needs to be in place to handle such faults and generate a reply, potentially filling in `fp`.

The documentation for this class was generated from the following file:

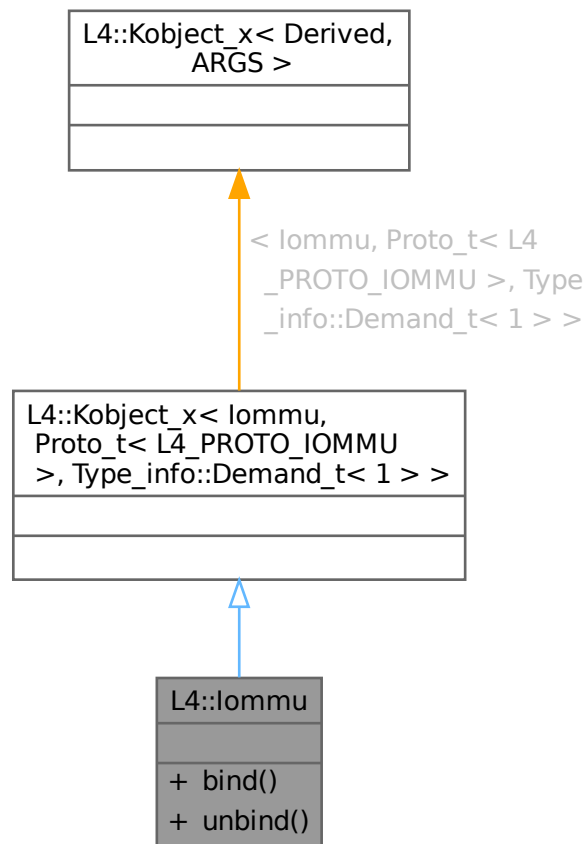
- [l4/sys/pager](#)

16.112 L4::lommu Class Reference

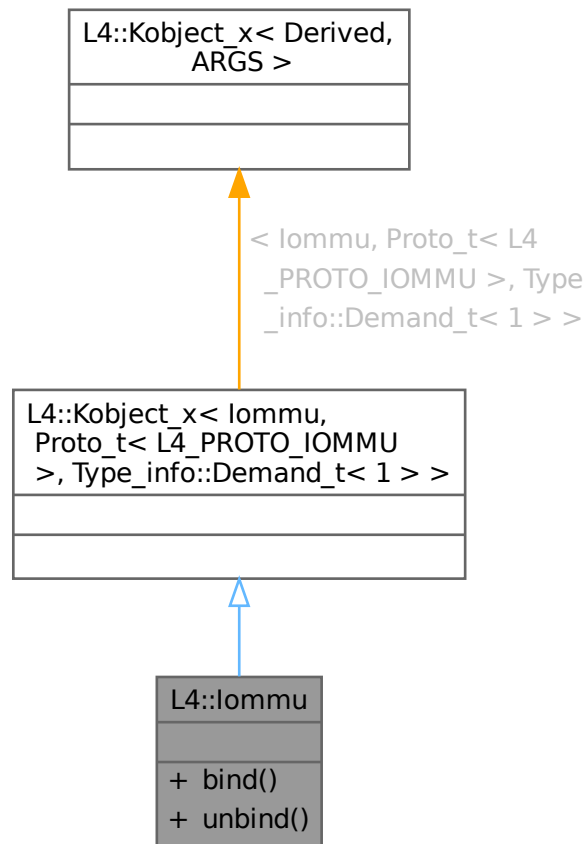
Interface for IO-MMUs used for DMA remapping.

```
#include <iommu>
```

Inheritance diagram for L4::lomu:



Collaboration diagram for L4::lommu:



Public Member Functions

- `l4_msgtag_t bind (l4_uint64_t src_id, lpc::Cap< Task > dma_space)`
Associate *dma_space* with the set of device(s) specified by *src_id*.
- `l4_msgtag_t unbind (l4_uint64_t src_id, lpc::Cap< Task > dma_space)`
Remove the association of the given DMA address space from the device(s) specified by *src_id*.

16.112.1 Detailed Description

Interface for IO-MMUs used for DMA remapping.

Note

This interface is only present in the kernel if the kernel detected an IOMMU during boot.

This interface allows to associate a DMA address space with a platform dependent set of devices. The kernel automatically keeps the memory spaces of associated DMA spaces in sync with the respective page table structures in the IOMMU.

Definition at line 21 of file [iommu](#).

16.112.2 Member Function Documentation

16.112.2.1 bind()

```
l4_msgtag_t L4::Iommu::bind (
    l4_uint64_t src_id,
    Ipc::Cap< Task > dma_space)
```

Associate `dma_space` with the set of device(s) specified by `src_id`.

Updates the respective page table structures in the IOMMU and keeps them in sync when memory is mapped to the `dma_space` or revoked from it.

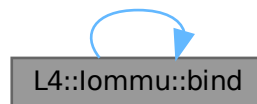
Parameters

<i>src_id</i>	Platform dependent source ID specifying the set of devices that shall use <code>dma_space</code> for DMA remapping.
<i>dma_space</i>	The DMA space (L4::Task created with <code>L4_PROTO_DMA_SPACE</code>) providing the mappings that shall be used for the device(s).

References [bind\(\)](#).

Referenced by [bind\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.112.2.2 unbind()

```
l4_msgtag_t L4::Iommu::unbind (
    l4_uint64_t src_id,
    Ipc::Cap< Task > dma_space)
```

Remove the association of the given DMA address space from the device(s) specified by `src_id`.

Clear the respective page stable structures in the IOMMU.

Parameters

<i>src_id</i>	Platform dependent source ID specifying the set of devices that shall no longer use <code>dma_space</code> for DMA remapping.
<i>dma_space</i>	The DMA space formerly associated with bind() .

References [unbind\(\)](#).

Referenced by [unbind\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

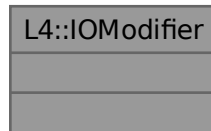
- `l4/sys/iommu`

16.113 L4::IOModifier Class Reference

Modifier class for the IO stream.

```
#include <basic_ostream>
```

Collaboration diagram for L4::IOModifier:



16.113.1 Detailed Description

Modifier class for the IO stream.

An IO Modifier can be used to change properties of an IO stream for example the number format.

Definition at line 22 of file [basic_ostream](#).

The documentation for this class was generated from the following file:

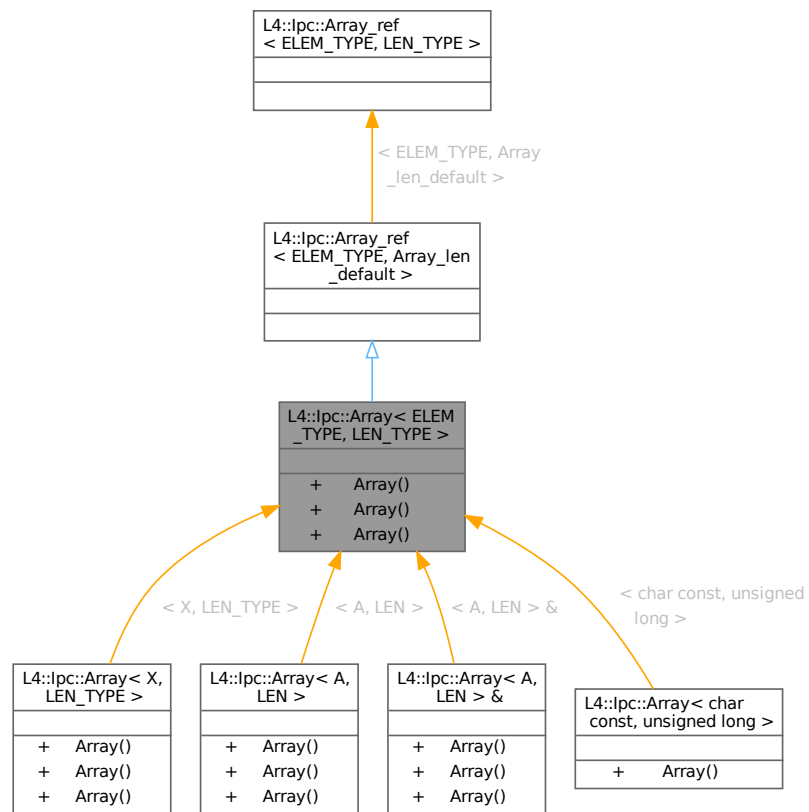
- [l4/cxx/basic_ostream](#)

16.114 L4::lpc::Array< ELEM_TYPE, LEN_TYPE > Struct Template Reference

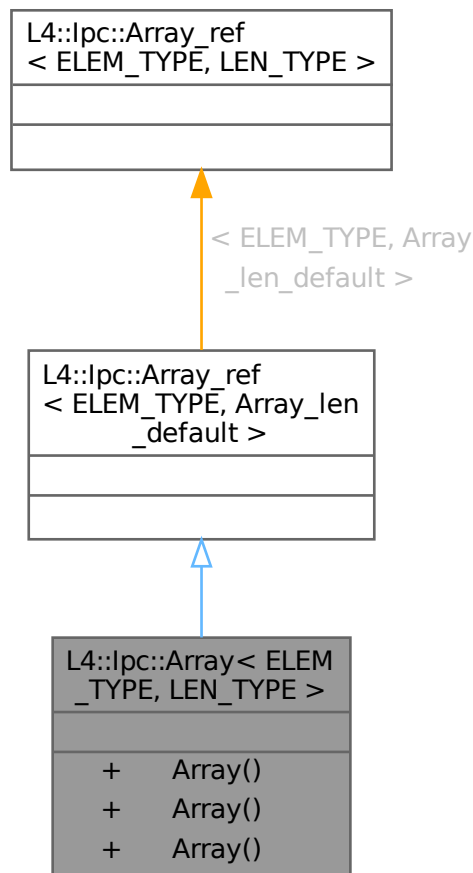
[Array](#) data type for dynamically sized arrays in RPCs.

```
#include <ipc_array>
```

Inheritance diagram for L4::lpc::Array< ELEM_TYPE, LEN_TYPE >:



Collaboration diagram for L4::lpc::Array< ELEM_TYPE, LEN_TYPE >:



Public Member Functions

- **Array ()**
Make array.
- **Array (LEN_TYPE length, ELEM_TYPE *data)**
Make array from length and data pointer.
- **Array (typename Non_const< ELEM_TYPE >::type const &other)**
Make a const array from a non-const array.

16.114.1 Detailed Description

```
template<typename ELEM_TYPE, typename LEN_TYPE = Array_len_default>
struct L4::lpc::Array< ELEM_TYPE, LEN_TYPE >
```

[Array](#) data type for dynamically sized arrays in RPCs.

Template Parameters

<i>ELEM_TYPE</i>	The data type of an array element, should be 'const' when used as input.
<i>LEN_TYPE</i>	Data type used to store the number of elements in the array.

An [Array](#) generally encapsulates a data pointer and a length (number of elements). [Array](#) does *not* provide any storage for the data itself. The storage is either provided by a client-side caller or in the case of [Array_ref](#) is the message itself.

Arrays can be used as input or as output arguments, when used as input *ELEM_TYPE* should be qualified *const*, when used as output a reference to an array must be used and the *ELEM_TYPE* must *not* be qualified *const*. It is the caller's responsibility to provide an array buffer of sufficient length. If a message from the server is too large it will be silently truncated.

If backward compatibility with `lpc::Stream` is required, then *LEN_TYPE* must be `unsigned long`.

Definition at line 81 of file [ipc_array](#).

The documentation for this struct was generated from the following file:

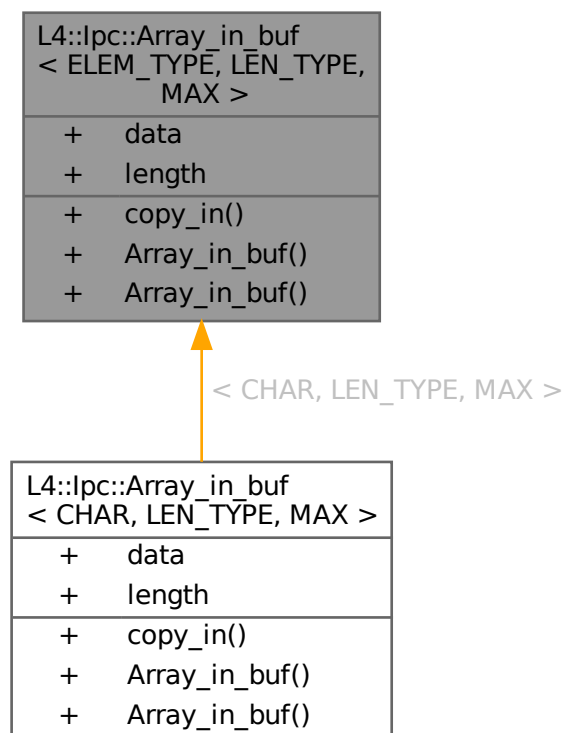
- `l4/sys/cxx/ipc_array`

16.115 L4::lpc::Array_in_buf< ELEM_TYPE, LEN_TYPE, MAX > Struct Template Reference

Server-side copy in buffer for [Array](#).

```
#include <ipc_array>
```

Inheritance diagram for `L4::lpc::Array_in_buf< ELEM_TYPE, LEN_TYPE, MAX >`:



Collaboration diagram for L4::lpc::Array_in_buf< ELEM_TYPE, LEN_TYPE, MAX >:

L4::lpc::Array_in_buf < ELEM_TYPE, LEN_TYPE, MAX >	
+	data
+	length
+	copy_in()
+	Array_in_buf()
+	Array_in_buf()

Public Member Functions

- void **copy_in** ([const_array](#) a)
copy in data from a source array
- **Array_in_buf** ([const_array](#) a)
Make [Array_in_buf](#) from a const array.
- **Array_in_buf** ([array](#) a)
Make [Array_in_buf](#) from a non-const array.

Data Fields

- ELEM_TYPE **data** [MAX]
The data elements.
- LEN_TYPE **length**
The length of the array.

16.115.1 Detailed Description

```
template<typename ELEM_TYPE, typename LEN_TYPE = Array_len_default, LEN_TYPE MAX = (L4_↔
UTCB_GENERIC_DATA_SIZE * sizeof(I4_umword_t)) / sizeof(ELEM_TYPE)>
struct L4::lpc::Array_in_buf< ELEM_TYPE, LEN_TYPE, MAX >
```

Server-side copy in buffer for [Array](#).

Template Parameters

<i>ELEM_TYPE</i>	Data type of an array element.
<i>LEN_TYPE</i>	Data type for the number of elements in the array.

MAX	The maximum number of elements in the buffer. If the actual message is longer than the buffer, it will be silently truncated.
------------	---

This type is assignment compatible to [Array_ref<ELEM_TYPE, LEN_TYPE>](#) and provides a transparent server-side copy-in mechanism for array parameters. The [Array_in_buf](#) provides the storage for the array data and receives a copy of the data passed to the server-function.

Definition at line 126 of file [ipc_array](#).

The documentation for this struct was generated from the following file:

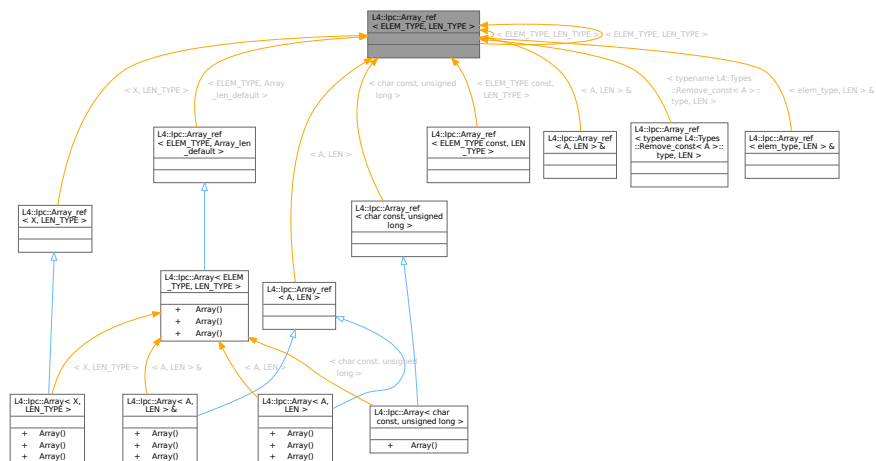
- [l4/sys/cxx/ipc_array](#)

16.116 L4::lpc::Array_ref< ELEM_TYPE, LEN_TYPE > Struct Template Reference

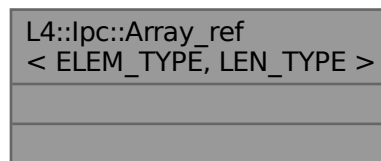
[Array](#) reference data type for arrays located in the message.

```
#include <ipc_array>
```

Inheritance diagram for L4::lpc::Array_ref< ELEM_TYPE, LEN_TYPE >:



Collaboration diagram for L4::lpc::Array_ref< ELEM_TYPE, LEN_TYPE >:



16.116.1 Detailed Description

```
template<typename ELEM_TYPE, typename LEN_TYPE = Array_len_default>
struct L4::lpc::Array_ref< ELEM_TYPE, LEN_TYPE >
```

[Array](#) reference data type for arrays located in the message.

Note

Use [Array](#) for normal RPC interfaces, [Array_ref](#) is usually used as server-side argument, see [Array](#).

Template Parameters

<i>ELEM_TYPE</i>	The data type of an array element, should be 'const' when used as input.
<i>LEN_TYPE</i>	Data type used to store the number of elements in the array.

Definition at line 28 of file [ipc_array](#).

The documentation for this struct was generated from the following file:

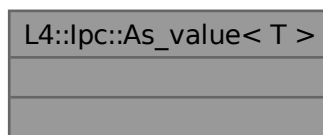
- l4/sys/cxx/ipc_array

16.117 L4::lpc::As_value< T > Struct Template Reference

Pass the argument as plain data value.

```
#include <ipc_types>
```

Collaboration diagram for L4::lpc::As_value< T >:



16.117.1 Detailed Description

```
template<typename T>
struct L4::lpc::As_value< T >
```

Pass the argument as plain data value.

Template Parameters

<i>T</i>	The type of the original argument.
----------	------------------------------------

[As_value<T>](#) is used when *T* would be otherwise interpreted specially, for example as flexpage. When using [As_value<>](#) then the argument is transmitted as plain data element.

Definition at line 116 of file [ipc_types](#).

The documentation for this struct was generated from the following file:

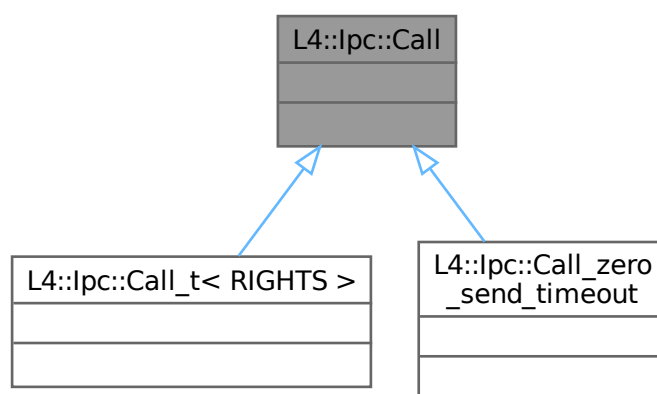
- [l4/sys/cxx/ipc_types](#)

16.118 L4::lpc::Call Struct Reference

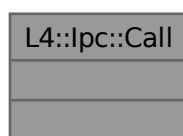
RPC attribute for a standard RPC call.

```
#include <ipc_iface>
```

Inheritance diagram for L4::lpc::Call:



Collaboration diagram for L4::lpc::Call:



16.118.1 Detailed Description

RPC attribute for a standard RPC call.

This is the default for the *FLAGS* parameter for `L4::lpc::Msg::Rpc_call` `L4::lpc::Msg::Rpc_inline_call` templates and declares the RPC to have default call semantics and timeouts.

Examples:

```
L4_RPC(long, send, (unsigned value), L4::Ipc::Call);
```

which is equivalent to:

```
L4_RPC(long, send, (unsigned value));
```

Definition at line 239 of file [ipc_iface](#).

The documentation for this struct was generated from the following file:

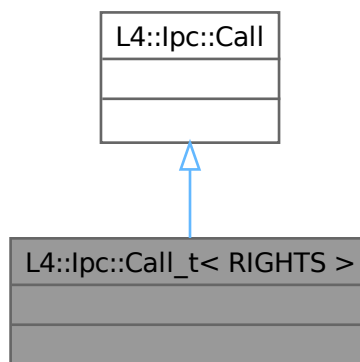
- [l4/sys/cxx/ipc_iface](#)

16.119 L4::lpc::Call_t< RIGHTS > Struct Template Reference

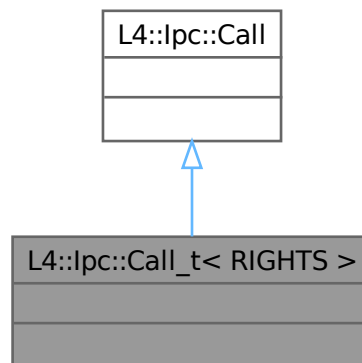
RPC attribute for an RPC call with required rights.

```
#include <ipc_iface>
```

Inheritance diagram for `L4::lpc::Call_t< RIGHTS >`:



Collaboration diagram for L4::Ipc::Call_t< RIGHTS >:



16.119.1 Detailed Description

```
template<unsigned RIGHTS>
struct L4::Ipc::Call_t< RIGHTS >
```

RPC attribute for an RPC call with required rights.

Template Parameters

<i>RIGHTS</i>	The capability rights required for this call. L4_CAP_FPAGE_W and L4_CAP_FPAGE_S are checked within the server (and -L4_EPERM shall be returned if the caller has insufficient rights). L4_CAP_FPAGE_R is always on but might be specified for documentation purposes. Other rights cannot be used in this context, because they cannot be checked at the server side.
---------------	---

Examples:

```
L4_RPC(long, func, (unsigned value), L4::Ipc::Call_t<L4_CAP_FPAGE_RW>);
```

Definition at line 270 of file [ipc_iface](#).

The documentation for this struct was generated from the following file:

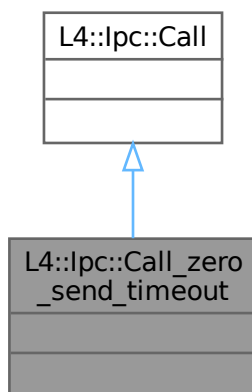
- [l4/sys/cxx/ipc_iface](#)

16.120 L4::Ipc::Call_zero_send_timeout Struct Reference

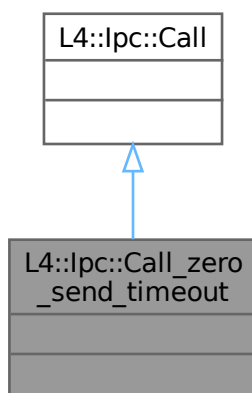
RPC attribute for an RPC call, with zero send timeout.

```
#include <ipc_iface>
```

Inheritance diagram for L4::lpc::Call_zero_send_timeout:



Collaboration diagram for L4::lpc::Call_zero_send_timeout:



16.120.1 Detailed Description

RPC attribute for an RPC call, with zero send timeout.

Definition at line 249 of file [ipc_iface](#).

The documentation for this struct was generated from the following file:

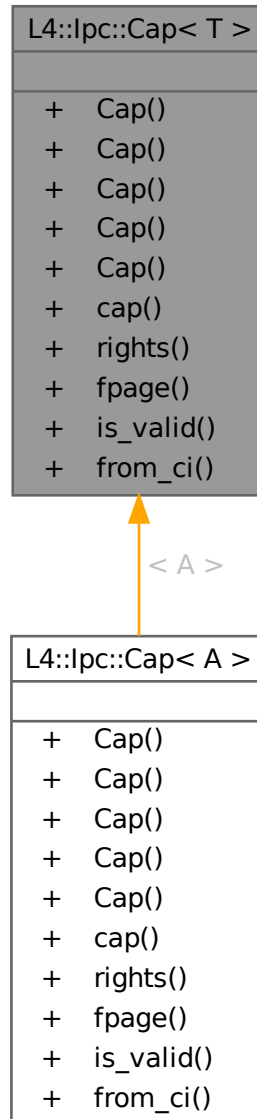
- [l4/sys/cxx/ipc_iface](#)

16.121 L4::lpc::Cap< T > Class Template Reference

Capability type for RPC interfaces (see [L4::Cap<T>](#)).

```
#include <ipc_types>
```

Inheritance diagram for L4::lpc::Cap< T >:



Collaboration diagram for L4::lpc::Cap< T >:

L4::lpc::Cap< T >
<ul style="list-style-type: none"> + Cap() + Cap() + Cap() + Cap() + Cap() + cap() + rights() + fpage() + is_valid() + from_ci()

Public Types

- enum { [Rights_mask](#) = 0xff , [Cap_mask](#) = L4_CAP_MASK }

Public Member Functions

- template<typename O>
Cap ([Cap](#)< O > const &o) noexcept
Make copy with conversion.
- Cap** ([L4::Cap](#)< T > [cap](#)) noexcept
Make a [Cap](#) from [L4::Cap<T>](#), with minimal rights.
- template<typename O>
Cap ([L4::Cap](#)< O > [cap](#)) noexcept
Make IPC [Cap](#) from [L4::Cap](#) with conversion (and minimal rights).
- Cap** () noexcept
Make an invalid cap.
- Cap** ([L4::Cap](#)< T > [cap](#), unsigned char [rights](#)) noexcept
Make a [Cap](#) from [L4::Cap<T>](#) with the given rights.
- [L4::Cap](#)< T > **cap** () const noexcept
Return the [L4::Cap<T>](#) of this [Cap](#).
- unsigned **rights** () const noexcept
Return the rights bits stored in this IPC cap.
- [L4::lpc::Snd_fpage](#) **fpage** () const noexcept
Return the send flexpage for this [Cap](#) (see [l4_fpage_t](#)).
- bool **is_valid** () const noexcept
Return true if this [Cap](#) is valid.

Static Public Member Functions

- static [Cap from_ci](#) ([l4_cap_idx_t](#) c) noexcept
Create an IPC capability from a C capability index plus rights.

16.121.1 Detailed Description

```
template<typename T>
class L4::lpc::Cap< T >
```

Capability type for RPC interfaces (see [L4 : :Cap<T>](#)).

Template Parameters

<i>T</i>	type of the interface referenced by the capability.
----------	---

In contrast to [L4 : :Cap<T>](#) this type additionally stores a rights mask that shall be used when the capability is transferred to the receiver. This allows to apply restrictions to the transferred capability in the form of a subset of the rights possessed by the sender.

See also

[L4::lpc::make_cap\(\)](#)

Definition at line 698 of file [ipc_types](#).

16.121.2 Member Enumeration Documentation

16.121.2.1 anonymous enum

```
template<typename T>
anonymous enum
```

Enumerator

Rights_mask	Mask for rights bits stored internally. L4_FPAGE_RIGHTS_MASK L4_FPAGE_C_NO_REF_CNT L4_FPAGE_C_OBJ_RIGHTS).
Cap_mask	Mask for significant capability bits. (incl. the invalid bit to support invalid caps)

Definition at line 704 of file [ipc_types](#).

16.121.3 Constructor & Destructor Documentation

16.121.3.1 Cap()

```
template<typename T>
L4::lpc::Cap< T >::Cap (
    L4::Cap< T > cap,
    unsigned char rights) [inline], [noexcept]
```

Make a [Cap](#) from [L4::Cap<T>](#) with the given rights.

Parameters

<i>cap</i>	Capability to be sent.
<i>rights</i>	Rights to be sent. Consists of L4_fpage_rights and L4_obj_fpage_ctl .

Definition at line [750](#) of file [ipc_types](#).

16.121.4 Member Function Documentation

16.121.4.1 from_ci()

```
template<typename T>
Cap L4::Ipc::Cap< T >::from_ci (
    l4_cap_idx_t c) [inline], [static], [noexcept]
```

Create an IPC capability from a C capability index plus rights.

Parameters

<i>c</i>	C capability index with the lowest 8 bits used as rights for the map operation (see L4_fpage_rights).
----------	--

Definition at line [758](#) of file [ipc_types](#).

The documentation for this class was generated from the following file:

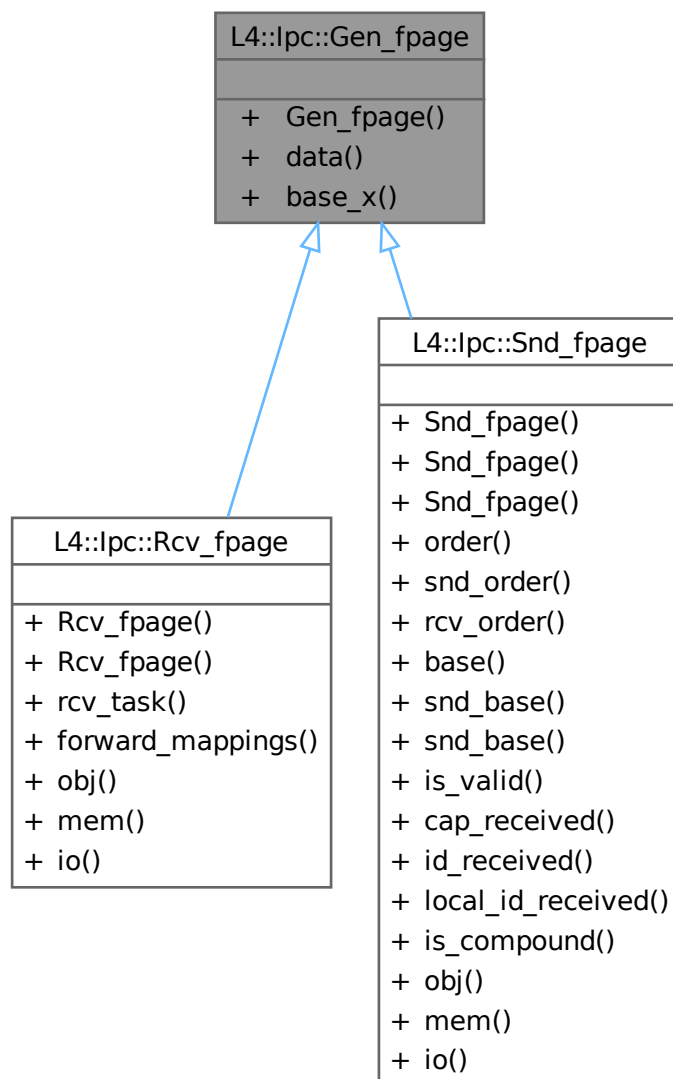
- [l4/sys/cxx/ipc_types](#)

16.122 L4::Ipc::Gen_fpage Class Reference

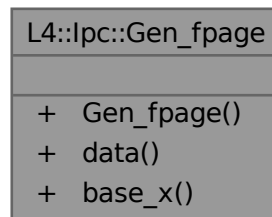
Generic RPC base for typed message items.

```
#include <ipc_types>
```

Inheritance diagram for L4::lpc::Gen_fpage:



Collaboration diagram for L4::Ipc::Gen_fpage:



Public Types

- enum [Type](#) { [Special](#) = L4_FPAGE_SPECIAL << 4 , [Memory](#) = L4_FPAGE_MEMORY << 4 , [Io](#) = L4_FPAGE_IO << 4 , [Obj](#) = L4_FPAGE_OBJ << 4 }
- Type of mapping object, see [L4_fpage_type](#).*

Public Member Functions

- **Gen_fpage** ([l4_umword_t](#) base, [l4_umword_t](#) data) noexcept
Construct from raw values.
- [l4_umword_t](#) **data** () const noexcept
Return the raw flexpage descriptor.
- [l4_umword_t](#) **base_x** () const noexcept
Return the raw base descriptor.

16.122.1 Detailed Description

Generic RPC base for typed message items.

Definition at line 286 of file [ipc_types](#).

16.122.2 Member Enumeration Documentation

16.122.2.1 Type

enum [L4::Ipc::Gen_fpage::Type](#)

[Type](#) of mapping object, see [L4_fpage_type](#).

Enumerator

Special	Special flexpage, either l4_fpage_invalid() or l4_fpage_all() ; only supported by selected interfaces.
Memory	Flexpage for memory spaces.
Io	Flexpage for I/O port spaces.
Obj	Flexpage for object spaces.

Definition at line [290](#) of file [ipc_types](#).

The documentation for this class was generated from the following file:

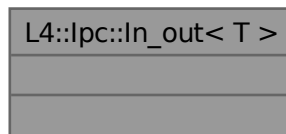
- [l4/sys/cxx/ipc_types](#)

16.123 L4::lpc::ln_out< T > Struct Template Reference

Mark an argument as in-out argument.

```
#include <ipc_types>
```

Collaboration diagram for L4::lpc::ln_out< T >:



16.123.1 Detailed Description

```
template<typename T>
struct L4::lpc::ln_out< T >
```

Mark an argument as in-out argument.

Template Parameters

<i>T</i>	The original argument type, usually a pointer or a reference.
----------	---

[ln_out<>](#) is used when an otherwise output-only value shall also be used as input value.

Definition at line [41](#) of file [ipc_types](#).

The documentation for this struct was generated from the following file:

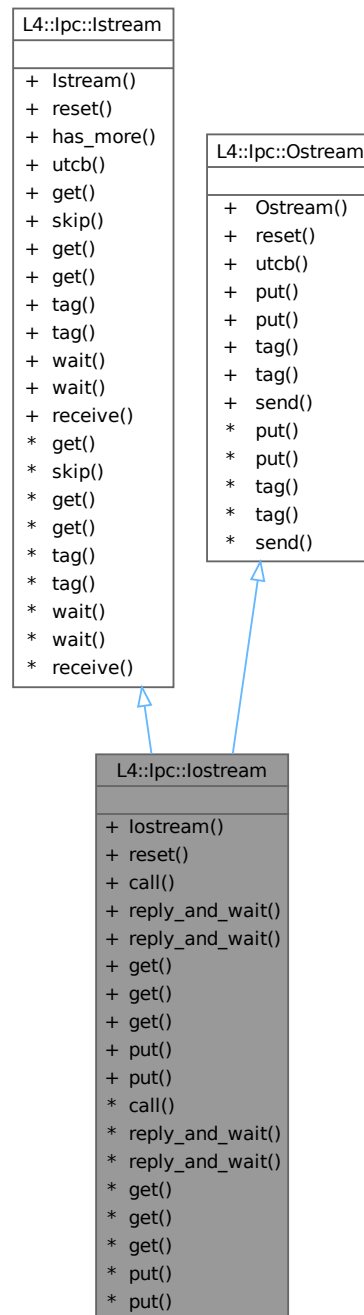
- [l4/sys/cxx/ipc_types](#)

16.124 L4::lpc::lostream Class Reference

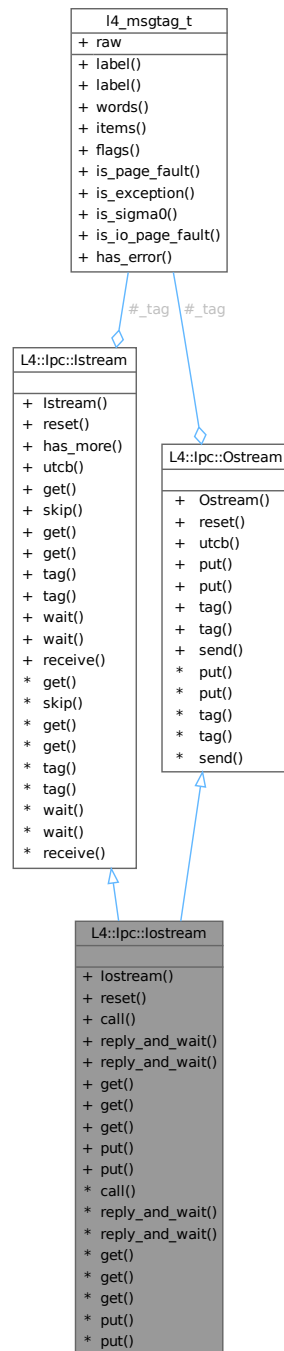
Input/Output stream for IPC [un]marshalling.

```
#include <ipc_stream>
```

Inheritance diagram for L4::lpc::lostream:



Collaboration diagram for L4::lpc::lostream:



Public Member Functions

- `lostream (l4_utcb_t *utcb)`
Create an IPC IO stream with a single message buffer.
- `void reset ()`
Reset the stream to its initial state.

IPC operations.

- `l4_msgtag_t call (l4_cap_idx_t dst, l4_timeout_t timeout, long proto=0)`
Do an IPC call using the message in the output stream and receive the reply in the input stream.
- `l4_msgtag_t reply_and_wait (l4_umword_t *src_dst, long proto=0)`
Do an IPC reply and wait.
- `l4_msgtag_t reply_and_wait (l4_umword_t *src_dst, l4_timeout_t timeout, long proto=0)`
Do an IPC reply and wait.

Get/Put functions.

These functions are basically used to implement the insertion operators (<<) and should not be called directly.

- `template<typename T>`
`unsigned long get (T *buf, unsigned long elems)`
Copy out an array of type T with size elements.
- `template<typename T>`
`unsigned long get (Msg_ptr< T > const &buf, unsigned long elems=1)`
Read one size elements of type T from the stream and return a pointer.
- `template<typename T>`
`bool get (T &v)`
Extract a single element of type T from the stream.
- `template<typename T>`
`bool put (T *buf, unsigned long size)`
Put an array with size elements of type T into the stream.
- `template<typename T>`
`bool put (T const &v)`
Insert an element of type T into the stream.

Public Member Functions inherited from `L4::lpc::lstream`

- `lstream (l4_utcb_t *utcb)`
Create an input stream for the given message buffer.
- `void reset ()`
Reset the stream to empty, and ready for `receive()/wait()`.
- `template<typename T>`
`bool has_more (unsigned long count=1)`
Check whether a value of type T can be obtained from the stream.
- `l4_utcb_t * utcb () const`
Return utcb pointer.
- `template<typename T>`
`unsigned long get (T *buf, unsigned long elems)`
Copy out an array of type T with size elements.
- `template<typename T>`
`void skip (unsigned long elems)`
Skip size elements of type T in the stream.
- `template<typename T>`
`unsigned long get (Msg_ptr< T > const &buf, unsigned long elems=1)`
Read one size elements of type T from the stream and return a pointer.
- `template<typename T>`
`bool get (T &v)`
Extract a single element of type T from the stream.
- `l4_msgtag_t tag () const`

Get the message tag of a received IPC.

- [l4_msgtag_t](#) & [tag](#) ()

Get the message tag of a received IPC.

- [l4_msgtag_t](#) wait ([l4_umword_t](#) *src)

Wait for an incoming message from any sender.

- [l4_msgtag_t](#) wait ([l4_umword_t](#) *src, [l4_timeout_t](#) timeout)

Wait for an incoming message from any sender.

- [l4_msgtag_t](#) receive ([l4_cap_idx_t](#) src)

Wait for a message from the specified sender.

Public Member Functions inherited from [L4::lpc::Ostream](#)

- **Ostream** ([l4_utcb_t](#) *utcb)

Create an IPC output stream using the given message buffer [utcb](#).

- void **reset** ()

Reset the stream to empty, same state as a newly created stream.

- [l4_utcb_t](#) * **utcb** () const

Return utcb pointer.

- template<typename T>
bool **put** (T *buf, unsigned long size)

*Put an array with *size* elements of type *T* into the stream.*

- template<typename T>
bool **put** (T const &v)

*Insert an element of type *T* into the stream.*

- [l4_msgtag_t](#) **tag** () const

Extract the [L4](#) message tag from the stream.

- [l4_msgtag_t](#) & **tag** ()

Extract a reference to the [L4](#) message tag from the stream.

- [l4_msgtag_t](#) **send** ([l4_cap_idx_t](#) dst, long proto=0, unsigned flags=0)

Send the message via IPC to the given receiver.

16.124.1 Detailed Description

Input/Output stream for IPC [un]marshalling.

The [lpc::lostream](#) is part of the AW Env IPC framework as well as [lpc::lstream](#) and [lpc::Ostream](#). In particular an [lpc::lostream](#) is a combination of an [lpc::lstream](#) and an [lpc::Ostream](#). It can use either a single message buffer for receiving and sending messages or a pair of a receive and a send buffer. The stream also supports combined IPC operations such as [call\(\)](#) and [reply_and_wait\(\)](#), which can be used to implement RPC functionality.

Examples

[examples/libs/l4re/c++/shared_ds/ds_srv.cc](#), [examples/libs/l4re/streammap/client.cc](#), and [examples/libs/l4re/streammap/server.cc](#).

Definition at line 789 of file [ipc_stream](#).

16.124.2 Constructor & Destructor Documentation

16.124.2.1 `Iostream()`

```
L4::Ipc::Iostream::Iostream (  
    l4_utcb_t * utcb) [inline], [explicit]
```

Create an IPC IO stream with a single message buffer.

Parameters

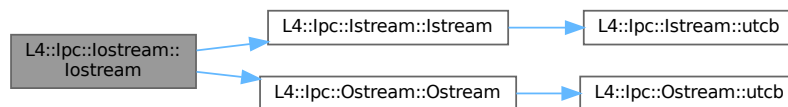
<i>utcb</i>	The message buffer used as backing store.
-------------	---

The created IO stream uses the same message buffer for sending and receiving IPC messages.

Definition at line 801 of file [ipc_stream](#).

References [L4::ipc::Istream::Istream\(\)](#), and [L4::ipc::Ostream::Ostream\(\)](#).

Here is the call graph for this function:



16.124.3 Member Function Documentation

16.124.3.1 call()

```

l4_msgtag_t L4::Ipc::Iostream::call (
    l4_cap_idx_t dst,
    l4_timeout_t timeout,
    long proto = 0) [inline]

```

Do an IPC call using the message in the output stream and receive the reply in the input stream.

Parameters

<i>dst</i>	The destination to call.
<i>timeout</i>	The IPC timeout for the call.
<i>proto</i>	The protocol value to use in the message tag.

Returns

The result tag of the IPC operation.

This is a combined IPC operation consisting of a send and a receive to/from the given destination *dst*.

A call is usually used by clients for RPCs to a server.

Examples

[examples/libs/l4re/streammap/client.cc](#).

Definition at line 966 of file [ipc_stream](#).

References [l4_ipc_call\(\)](#), and [L4::ipc::Istream::tag\(\)](#).

Here is the call graph for this function:



16.124.3.2 `get()` [1/3]

```

template<typename T>
unsigned long L4::IpC::Istream::get (
    Msg_ptr< T > const & buf,
    unsigned long elems = 1) [inline]
  
```

Read one size elements of type T from the stream and return a pointer.

Parameters

<i>buf</i>	A Msg_ptr that is actually set to point to the element in the stream.
<i>elems</i>	Number of elements to extract (default is 1).

Returns

The number of elements extracted.

In contrast to a normal `get`, this version does actually not copy the data but returns a pointer to the data.

See [operator>>\(\)](#)

Definition at line 439 of file [ipc_stream](#).

16.124.3.3 `get()` [2/3]

```

template<typename T>
bool L4::IpC::Istream::get (
    T & v) [inline]
  
```

Extract a single element of type T from the stream.

Parameters

<i>out</i>	<i>v</i>	The element.
------------	----------	--------------

Return values

<i>true</i>	An element was successfully extracted.
<i>false</i>	An element could not be extracted.

See [operator>>\(\)](#)

Definition at line 464 of file [ipc_stream](#).

16.124.3.4 get() [3/3]

```
template<typename T>
unsigned long L4::Ipc::Istream::get (
    T * buf,
    unsigned long elems) [inline]
```

Copy out an array of type T with *size* elements.

Parameters

<i>buf</i>	Pointer to a buffer for <i>size</i> elements of type T.
<i>elems</i>	Number of elements of type T to copy out.

Returns

The number of elements copied out.

See [operator>>\(\)](#)

Definition at line 394 of file [ipc_stream](#).

16.124.3.5 put() [1/2]

```
template<typename T>
bool L4::Ipc::Ostream::put (
    T * buf,
    unsigned long size) [inline]
```

Put an array with *size* elements of type T into the stream.

Parameters

<i>buf</i>	A pointer to the array to insert into the buffer.
------------	---

<i>size</i>	The number of elements in the array.
-------------	--------------------------------------

Definition at line 660 of file [ipc_stream](#).

16.124.3.6 put() [2/2]

```
template<typename T>
bool L4::IpC::Ostream::put (
    T const & v) [inline]
```

Insert an element of type T into the stream.

Parameters

<i>v</i>	The element to insert.
----------	------------------------

Definition at line 678 of file [ipc_stream](#).

16.124.3.7 reply_and_wait() [1/2]

```
l4_msgtag_t L4::IpC::Iostream::reply_and_wait (
    l4_umword_t * src_dst,
    l4_timeout_t timeout,
    long proto = 0) [inline]
```

Do an IPC reply and wait.

Parameters

<i>in, out</i>	<i>src_dst</i>	Input: the destination for the send operation. Output: the source of the received message.
	<i>timeout</i>	Timeout used for IPC.
	<i>proto</i>	Protocol to use.

Returns

The result tag of the IPC operation.

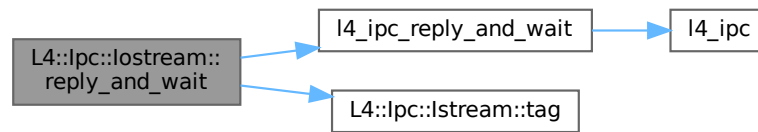
This is a combined IPC operation consisting of a send operation and an open wait for any message.

A reply and wait is usually used by servers that reply to a client and wait for the next request by any other client.

Definition at line 981 of file [ipc_stream](#).

References [l4_ipc_reply_and_wait\(\)](#), and [L4::IpC::Istream::tag\(\)](#).

Here is the call graph for this function:



16.124.3.8 reply_and_wait() [2/2]

```

l4_msgtag_t L4::Ipc::Iostream::reply_and_wait (
    l4_umword_t * src_dst,
    long proto = 0) [inline]
  
```

Do an IPC reply and wait.

Parameters

<i>in, out</i>	<i>src_dst</i>	Input: the destination for the send operation. Output: the source of the received message.
	<i>proto</i>	Protocol to use.

Returns

The result tag of the IPC operation.

This is a combined IPC operation consisting of a send operation and an open wait for any message.

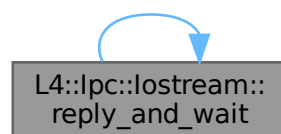
A reply and wait is usually used by servers that reply to a client and wait for the next request by any other client.

Definition at line 874 of file [ipc_stream](#).

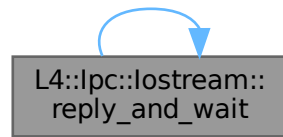
References [L4_IPC_SEND_TIMEOUT_0](#), and [reply_and_wait\(\)](#).

Referenced by [reply_and_wait\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.124.3.9 reset()

```
void L4::Ipc::Iostream::reset () [inline]
```

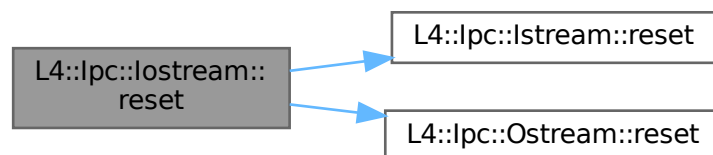
Reset the stream to its initial state.

Input as well as the output stream are reset.

Definition at line 815 of file [ipc_stream](#).

References [L4::lpc::Istream::reset\(\)](#), and [L4::lpc::Ostream::reset\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

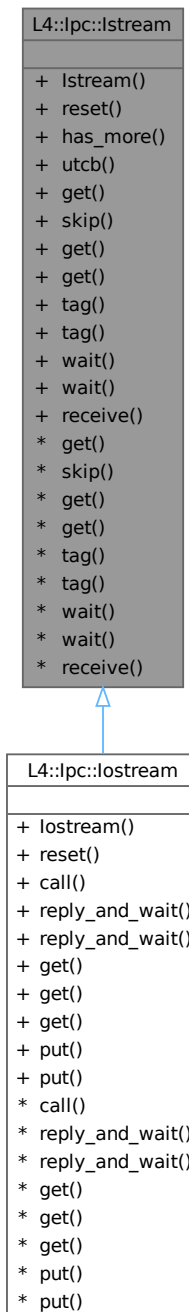
- [l4/cxx/ipc_stream](#)

16.125 L4::lpc::Istream Class Reference

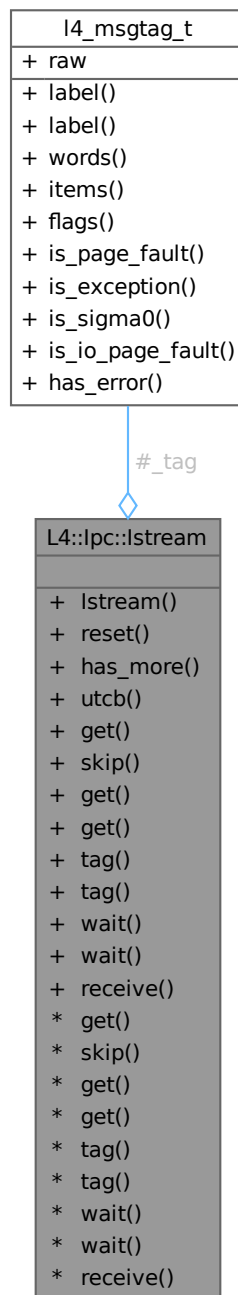
Input stream for IPC unmarshalling.

```
#include <ipc_stream>
```

Inheritance diagram for L4::lpc::Istream:



Collaboration diagram for L4::lpc::lstream:



Public Member Functions

- [lstream](#) ([l4_utcb_t](#) *utcb)
Create an input stream for the given message buffer.
- void [reset](#) ()
Reset the stream to empty, and ready for [receive\(\)](#)/[wait\(\)](#).

- `template<typename T>`
`bool has_more (unsigned long count=1)`
Check whether a value of type T can be obtained from the stream.
- `l4_utcb_t * utcb () const`
Return utcb pointer.

Get/Put Functions.

- `template<typename T>`
`unsigned long get (T *buf, unsigned long elems)`
Copy out an array of type T with size elements.
- `template<typename T>`
`void skip (unsigned long elems)`
Skip size elements of type T in the stream.
- `template<typename T>`
`unsigned long get (Msg_ptr< T > const &buf, unsigned long elems=1)`
Read one size elements of type T from the stream and return a pointer.
- `template<typename T>`
`bool get (T &v)`
Extract a single element of type T from the stream.
- `l4_msgtag_t tag () const`
Get the message tag of a received IPC.
- `l4_msgtag_t & tag ()`
Get the message tag of a received IPC.

IPC operations.

- `l4_msgtag_t wait (l4_umword_t *src)`
Wait for an incoming message from any sender.
- `l4_msgtag_t wait (l4_umword_t *src, l4_timeout_t timeout)`
Wait for an incoming message from any sender.
- `l4_msgtag_t receive (l4_cap_idx_t src)`
Wait for a message from the specified sender.

16.125.1 Detailed Description

Input stream for IPC unmarshalling.

[lpc::Istream](#) is part of the dynamic IPC marshalling infrastructure, as well as [lpc::Ostream](#) and [lpc::Iostream](#).

[lpc::Istream](#) is an input stream supporting extraction of values from an IPC message buffer. A received IPC message can be unmarshalled using the usual extraction operator (>>).

There exist some special wrapper classes to extract arrays (see `lpc_buf_cp_in` and `lpc_buf_in`) and indirect strings (see `Msg_in_buffer` and `Msg_io_buffer`).

Definition at line 334 of file [ipc_stream](#).

16.125.2 Constructor & Destructor Documentation

16.125.2.1 Istream()

```
L4::Ipc::Istream::Istream (
    l4_utcb_t * utcb) [inline]
```

Create an input stream for the given message buffer.

The given message buffer is used for IPC operations [wait\(\)](#)/[receive\(\)](#) and received data can be extracted using the >> operator afterwards. In the case of indirect message parts a buffer of type `Msg_in_buffer` must be inserted into the stream before the IPC operation and contains received data afterwards.

Parameters

<i>utcb</i>	The message buffer to receive IPC messages.
-------------	---

Definition at line 348 of file [ipc_stream](#).

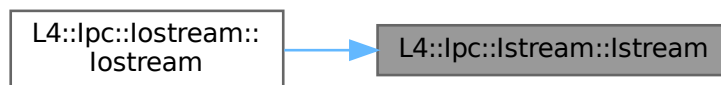
References [utcb\(\)](#).

Referenced by [L4::lpc::lstream::lstream\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.125.3 Member Function Documentation

16.125.3.1 `get()` [1/3]

```

template<typename T>
unsigned long L4::lpc::lstream::get (
    Msg\_ptr< T > const & buf,
    unsigned long elems = 1) [inline]
  
```

Read one size elements of type T from the stream and return a pointer.

Parameters

<i>buf</i>	A Msg_ptr that is actually set to point to the element in the stream.
<i>elems</i>	Number of elements to extract (default is 1).

Returns

The number of elements extracted.

In contrast to a normal get, this version does actually not copy the data but returns a pointer to the data.

See [operator>>\(\)](#)

Definition at line 439 of file [ipc_stream](#).

References [has_more\(\)](#), and [L4_UNLIKELY](#).

Here is the call graph for this function:

**16.125.3.2 get() [2/3]**

```

template<typename T>
bool L4::lpc::Istream::get (
    T & v) [inline]
  
```

Extract a single element of type T from the stream.

Parameters

out	v	The element.
-----	---	--------------

Return values

<i>true</i>	An element was successfully extracted.
<i>false</i>	An element could not be extracted.

See [operator>>\(\)](#)

Definition at line 464 of file [ipc_stream](#).

References [has_more\(\)](#), and [L4_UNLIKELY](#).

Here is the call graph for this function:



16.125.3.3 `get()` [3/3]

```
template<typename T>
unsigned long L4::Ipc::Istream::get (
    T * buf,
    unsigned long elems) [inline]
```

Copy out an array of type `T` with `size` elements.

Parameters

<i>buf</i>	Pointer to a buffer for <code>size</code> elements of type <code>T</code> .
<i>elems</i>	Number of elements of type <code>T</code> to copy out.

Returns

The number of elements copied out.

See [operator>>\(\)](#)

Definition at line 394 of file [ipc_stream](#).

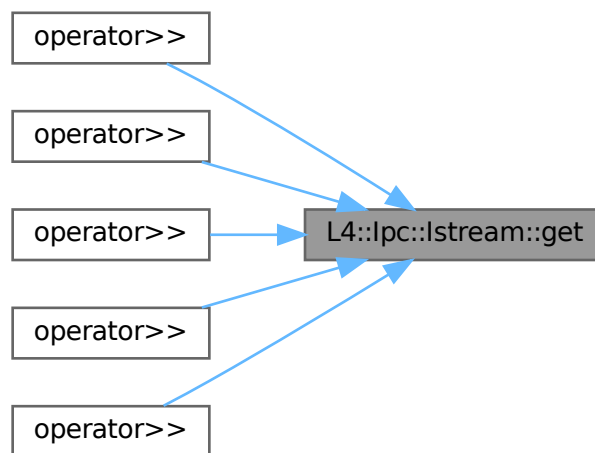
References [has_more\(\)](#), and [L4_UNLIKELY](#).

Referenced by [operator>>\(\)](#), [operator>>\(\)](#), [operator>>\(\)](#), [operator>>\(\)](#), and [operator>>\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.125.3.4 receive()

```
l4_msgtag_t L4::Ipc::Istream::receive (
    l4_cap_idx_t src) [inline]
```

Wait for a message from the specified sender.

Parameters

<code>src</code>	The sender id to receive from.
------------------	--------------------------------

Returns

The IPC result tag (`l4_msgtag_t`).

This is commonly known as 'closed wait'.

Definition at line 572 of file `ipc_stream`.

References `L4_IPC_NEVER`, and `receive()`.

Referenced by `receive()`.

Here is the call graph for this function:



Here is the caller graph for this function:



16.125.3.5 reset()

```
void L4::Ipc::Istream::reset () [inline]
```

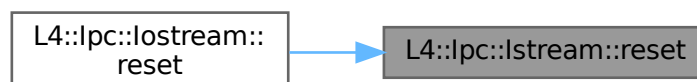
Reset the stream to empty, and ready for [receive\(\)/wait\(\)](#).

The stream is reset to the same state as on its creation.

Definition at line [358](#) of file [ipc_stream](#).

Referenced by [L4::ipc::Istream::reset\(\)](#).

Here is the caller graph for this function:



16.125.3.6 skip()

```
template<typename T>
void L4::Ipc::Istream::skip (
    unsigned long elems) [inline]
```

Skip size elements of type T in the stream.

Parameters

<i>elems</i>	Number of elements to skip.
--------------	-----------------------------

Definition at line 414 of file [ipc_stream](#).

References [has_more\(\)](#), and [L4_UNLIKELY](#).

Here is the call graph for this function:



16.125.3.7 tag() [1/2]

```
l4\_msgtag\_t & L4::Ipc::Istream::tag () [inline]
```

Get the message tag of a received IPC.

Returns

A reference to the [L4](#) message tag for the received IPC.

This is in particular useful for handling page faults or exceptions.

See [operator>>\(\)](#)

Definition at line 517 of file [ipc_stream](#).

16.125.3.8 tag() [2/2]

```
l4\_msgtag\_t L4::Ipc::Istream::tag () const [inline]
```

Get the message tag of a received IPC.

Returns

The [L4](#) message tag for the received IPC.

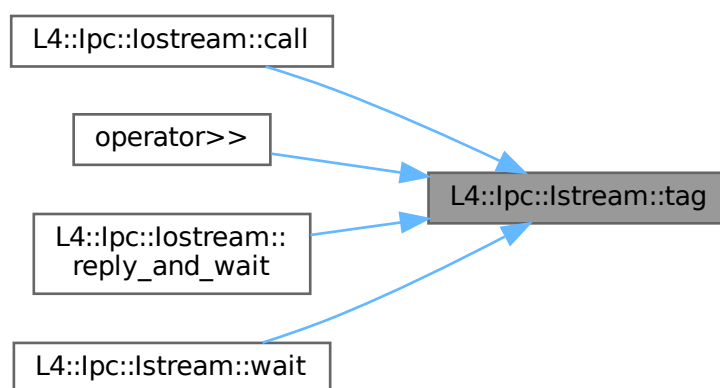
This is in particular useful for handling page faults or exceptions.

See [operator>>\(\)](#)

Definition at line 505 of file [ipc_stream](#).

Referenced by [L4::lpc::lostream::call\(\)](#), [operator>>\(\)](#), [L4::lpc::lostream::reply_and_wait\(\)](#), and [wait\(\)](#).

Here is the caller graph for this function:

**16.125.3.9 wait() [1/2]**

```
l4_msgtag_t L4::Ipc::Istream::wait (
    l4_umword_t * src) [inline]
```

Wait for an incoming message from any sender.

Parameters

out	src	Contains the sender after a successful IPC operation.
-----	-----	---

Returns

Syscall return tag.

This wait is actually known as 'open wait'.

Definition at line 548 of file [ipc_stream](#).

References [L4_IPC_NEVER](#), and [wait\(\)](#).

Referenced by [wait\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**16.125.3.10 wait() [2/2]**

```

l4_msgtag_t L4::Ipc::Istream::wait (
    l4_umword_t * src,
    l4_timeout_t timeout) [inline]
  
```

Wait for an incoming message from any sender.

Parameters

out	src	Contains the sender after a successful IPC operation.
	timeout	Timeout used for IPC.

Returns

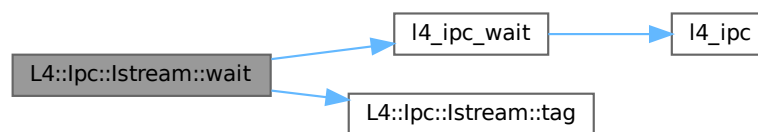
The IPC result tag ([l4_msgtag_t](#)).

This wait is actually known as 'open wait'.

Definition at line 1013 of file [ipc_stream](#).

References [l4_ipc_wait\(\)](#), and [tag\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

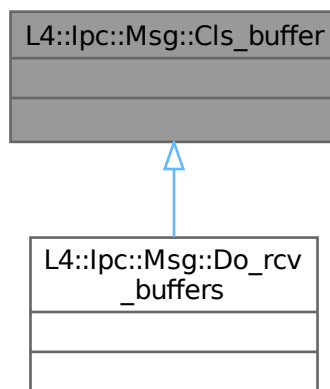
- [l4/cxx/ipc_stream](#)

16.126 L4::lpc::Msg::Cls_buffer Struct Reference

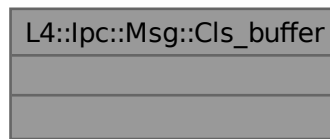
Marker type for receive buffer values.

```
#include <ipc_basics>
```

Inheritance diagram for `L4::lpc::Msg::Cls_buffer`:



Collaboration diagram for L4::lpc::Msg::Cls_buffer:



16.126.1 Detailed Description

Marker type for receive buffer values.

Definition at line 154 of file [ipc_basics](#).

The documentation for this struct was generated from the following file:

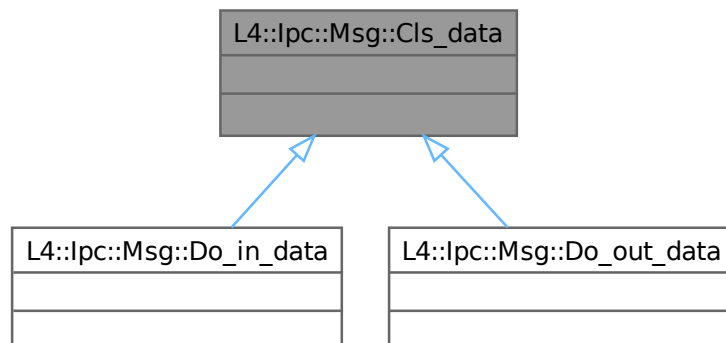
- [l4/sys/cxx/ipc_basics](#)

16.127 L4::lpc::Msg::Cls_data Struct Reference

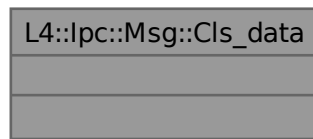
Marker type for data values.

```
#include <ipc_basics>
```

Inheritance diagram for L4::lpc::Msg::Cls_data:



Collaboration diagram for L4::lpc::Msg::Cls_data:



16.127.1 Detailed Description

Marker type for data values.

Definition at line 150 of file [ipc_basics](#).

The documentation for this struct was generated from the following file:

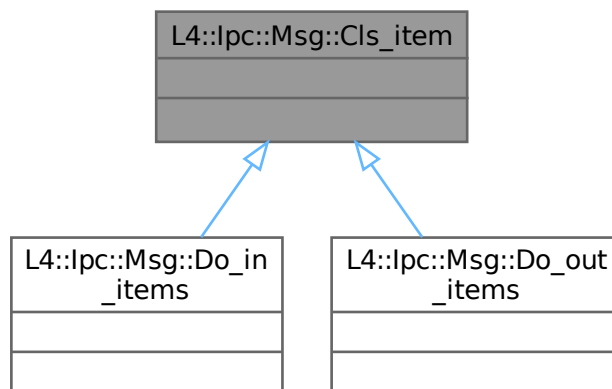
- l4/sys/cxx/ipc_basics

16.128 L4::lpc::Msg::Cls_item Struct Reference

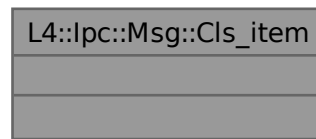
Marker type for item values.

```
#include <ipc_basics>
```

Inheritance diagram for L4::lpc::Msg::Cls_item:



Collaboration diagram for L4::lpc::Msg::Cls_item:



16.128.1 Detailed Description

Marker type for item values.

Definition at line 152 of file [ipc_basics](#).

The documentation for this struct was generated from the following file:

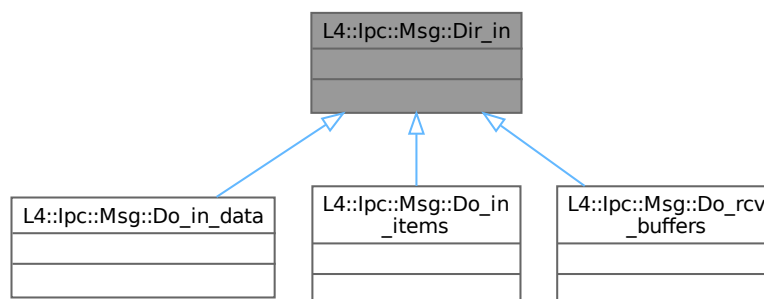
- `l4/sys/cxx/ipc_basics`

16.129 L4::lpc::Msg::Dir_in Struct Reference

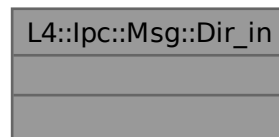
Marker type for input values.

```
#include <ipc_basics>
```

Inheritance diagram for L4::lpc::Msg::Dir_in:



Collaboration diagram for L4::lpc::Msg::Dir_in:



16.129.1 Detailed Description

Marker type for input values.

Definition at line 145 of file [ipc_basics](#).

The documentation for this struct was generated from the following file:

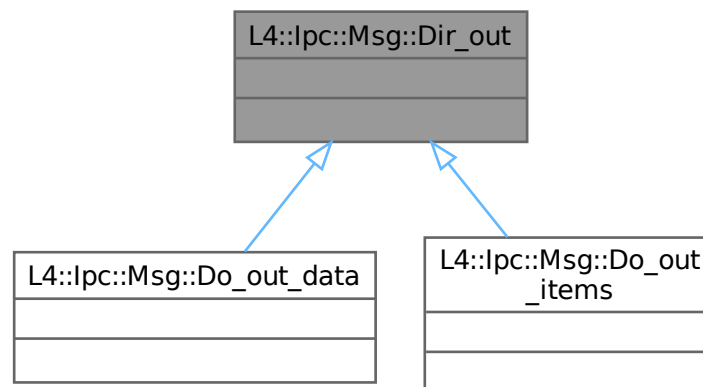
- `l4/sys/cxx/ipc_basics`

16.130 L4::lpc::Msg::Dir_out Struct Reference

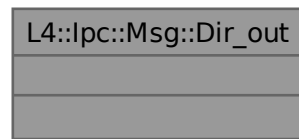
Marker type for output values.

```
#include <ipc_basics>
```

Inheritance diagram for L4::lpc::Msg::Dir_out:



Collaboration diagram for L4::lpc::Msg::Dir_out:



16.130.1 Detailed Description

Marker type for output values.

Definition at line 147 of file [ipc_basics](#).

The documentation for this struct was generated from the following file:

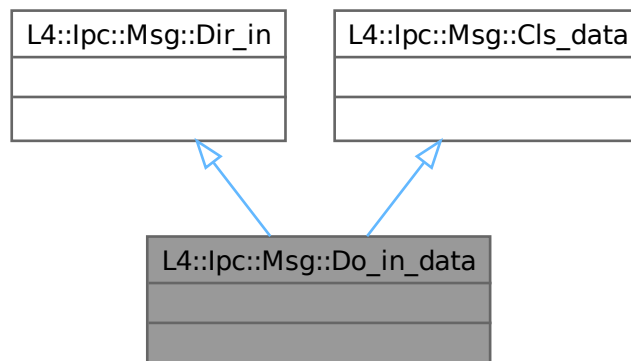
- l4/sys/cxx/ipc_basics

16.131 L4::lpc::Msg::Do_in_data Struct Reference

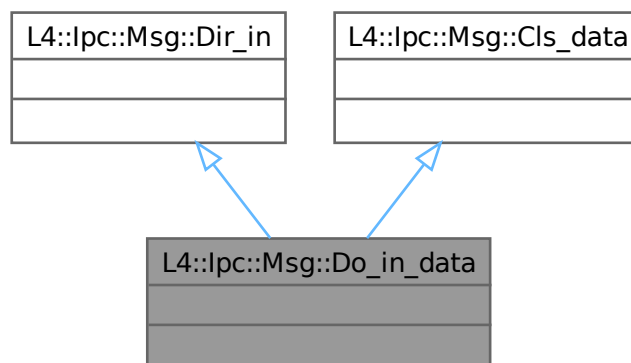
Marker for Input data.

```
#include <ipc_basics>
```

Inheritance diagram for L4::lpc::Msg::Do_in_data:



Collaboration diagram for L4::lpc::Msg::Do_in_data:



16.131.1 Detailed Description

Marker for Input data.

Definition at line 158 of file [ipc_basics](#).

The documentation for this struct was generated from the following file:

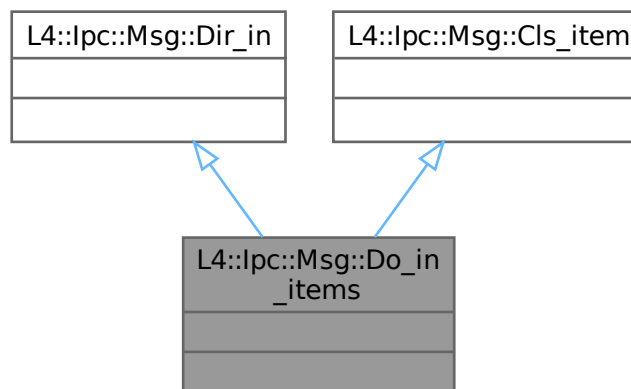
- `l4/sys/cxx/ipc_basics`

16.132 L4::lpc::Msg::Do_in_items Struct Reference

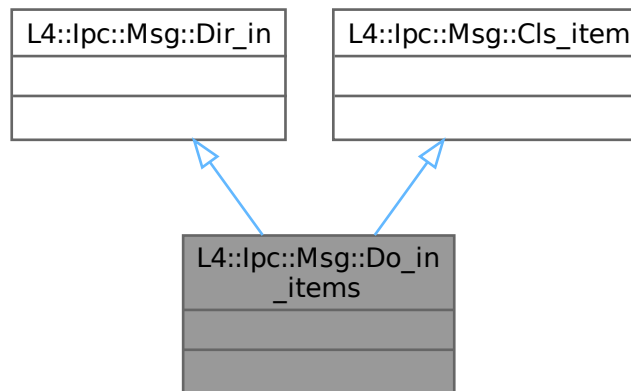
Marker for Input items.

```
#include <ipc_basics>
```

Inheritance diagram for L4::lpc::Msg::Do_in_items:



Collaboration diagram for L4::lpc::Msg::Do_in_items:



16.132.1 Detailed Description

Marker for Input items.

Definition at line 162 of file [ipc_basics](#).

The documentation for this struct was generated from the following file:

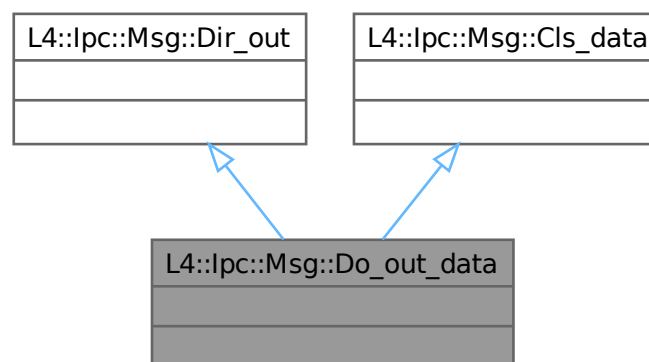
- `l4/sys/cxx/ipc_basics`

16.133 L4::lpc::Msg::Do_out_data Struct Reference

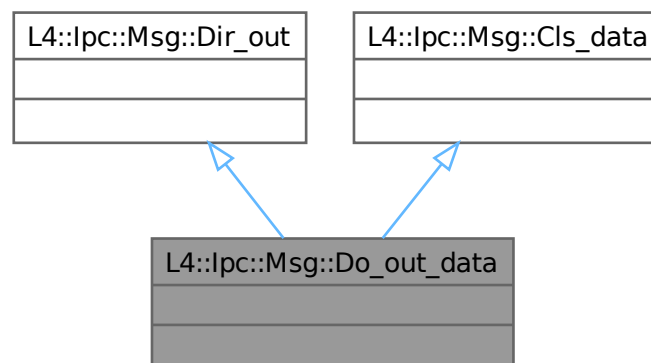
Marker for Output data.

```
#include <ipc_basics>
```

Inheritance diagram for L4::lpc::Msg::Do_out_data:



Collaboration diagram for L4::lpc::Msg::Do_out_data:



16.133.1 Detailed Description

Marker for Output data.

Definition at line 160 of file [ipc_basics](#).

The documentation for this struct was generated from the following file:

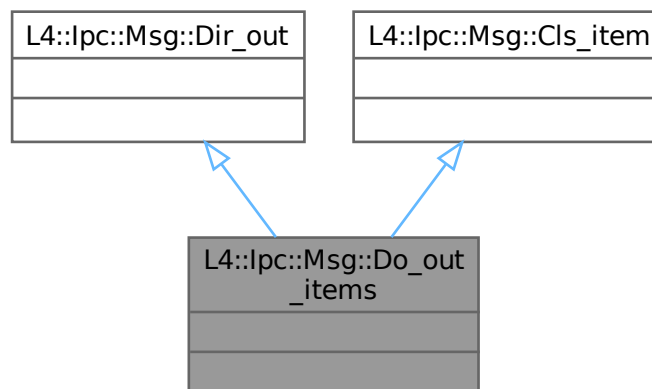
- l4/sys/cxx/ipc_basics

16.134 L4::lpc::Msg::Do_out_items Struct Reference

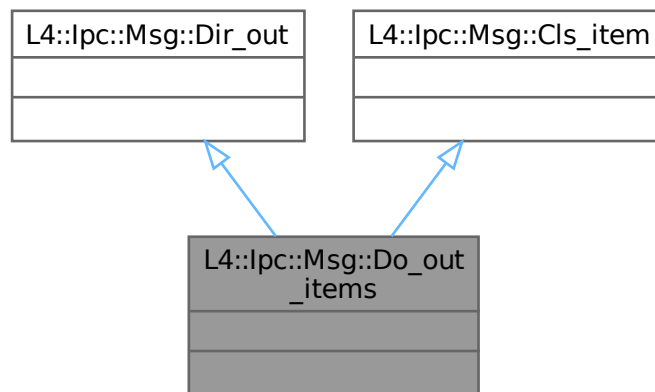
Marker for Output items.

```
#include <ipc_basics>
```

Inheritance diagram for L4::lpc::Msg::Do_out_items:



Collaboration diagram for L4::lpc::Msg::Do_out_items:



16.134.1 Detailed Description

Marker for Output items.

Definition at line 164 of file [ipc_basics](#).

The documentation for this struct was generated from the following file:

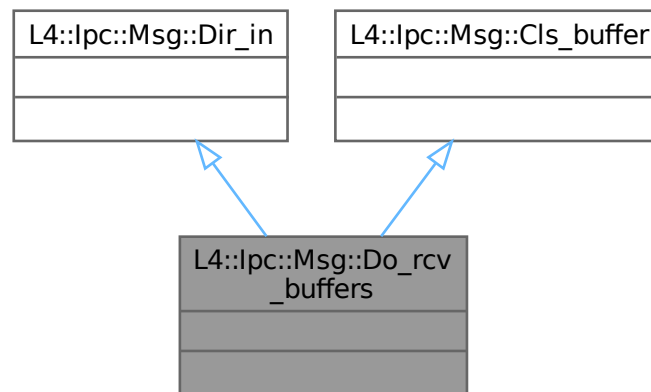
- `l4/sys/cxx/ipc_basics`

16.135 L4::lpc::Msg::Do_rcv_buffers Struct Reference

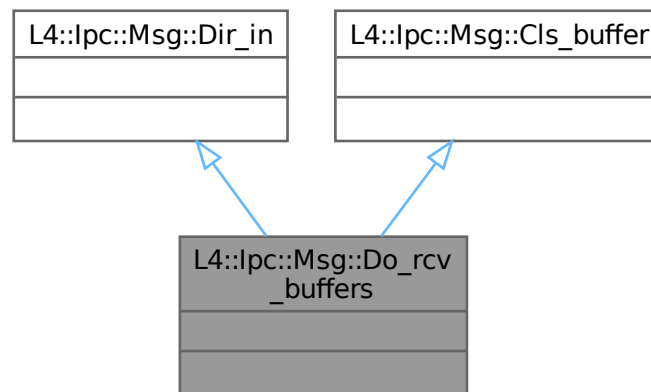
Marker for receive buffers.

```
#include <ipc_basics>
```

Inheritance diagram for L4::lpc::Msg::Do_rcv_buffers:



Collaboration diagram for L4::lpc::Msg::Do_rcv_buffers:



16.135.1 Detailed Description

Marker for receive buffers.

Definition at line 166 of file [ipc_basics](#).

The documentation for this struct was generated from the following file:

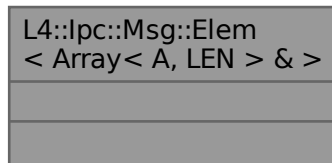
- `l4/sys/cxx/ipc_basics`

16.136 L4::lpc::Msg::Elem< Array< A, LEN > & > Struct Template Reference

[Array](#) as output argument.

```
#include <ipc_array>
```

Collaboration diagram for L4::lpc::Msg::Elem< Array< A, LEN > & >:



Public Types

- typedef [Array](#)< A, LEN > & **arg_type**
[Array](#)<> & at the interface.
- typedef [Array_ref](#)< A, LEN > **svr_type**
[Array_ref](#)<> as server storage type.
- typedef [svr_type](#) & **svr_arg_type**
[Array_ref](#)<> & at the server side.

16.136.1 Detailed Description

```
template<typename A, typename LEN>
struct L4::lpc::Msg::Elem< Array< A, LEN > & >
```

[Array](#) as output argument.

Definition at line 170 of file [ipc_array](#).

The documentation for this struct was generated from the following file:

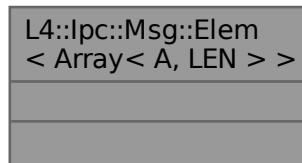
- `I4/sys/cxx/ipc_array`

16.137 L4::lpc::Msg::Elem< Array< A, LEN > > Struct Template Reference

[Array](#) as input arguments.

```
#include <ipc_array>
```

Collaboration diagram for L4::lpc::Msg::Elem< Array< A, LEN > >:



Public Types

- typedef [Array](#)< A, LEN > **arg_type**
[Array](#)<> as argument at the interface.
- typedef [Array_ref](#)< A, LEN > **svr_type**
[Array_ref](#)<> at the server side.

16.137.1 Detailed Description

```
template<typename A, typename LEN>
struct L4::lpc::Msg::Elem< Array< A, LEN > >
```

[Array](#) as input arguments.

Definition at line 158 of file [ipc_array](#).

The documentation for this struct was generated from the following file:

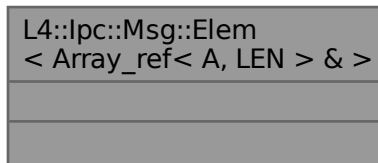
- `l4/sys/cxx/ipc_array`

16.138 L4::lpc::Msg::Elem< Array_ref< A, LEN > & > Struct Template Reference

[Array_ref](#) as output argument.

```
#include <ipc_array>
```

Collaboration diagram for L4::lpc::Msg::Elem< Array_ref< A, LEN > & >:



Public Types

- typedef [Array_ref](#)< A, LEN > & **arg_type**
[Array_ref](#)<> at the interface.
- typedef [Array_ref](#)< typename L4::Types::Remove_const< A >::type, LEN > **svr_type**
[Array_ref](#)<> as server storage.
- typedef **svr_type** & **svr_arg_type**
[Array_ref](#)<> & as server argument.

16.138.1 Detailed Description

```
template<typename A, typename LEN>
struct L4::lpc::Msg::Elem< Array_ref< A, LEN > & >
```

[Array_ref](#) as output argument.

Definition at line 183 of file [ipc_array](#).

The documentation for this struct was generated from the following file:

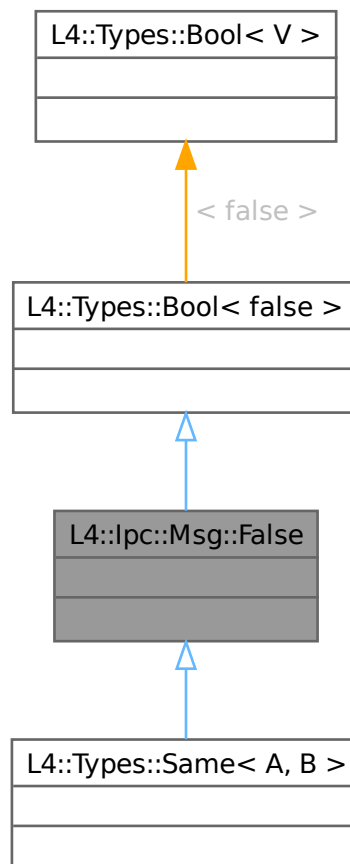
- I4/sys/cxx/ipc_array

16.139 L4::lpc::Msg::False Struct Reference

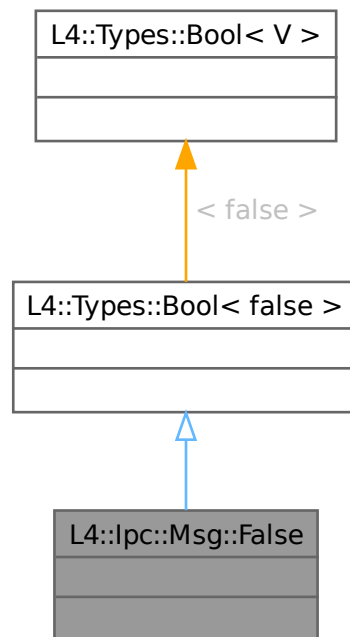
False meta value.

```
#include <types>
```

Inheritance diagram for L4::lpc::Msg::False:



Collaboration diagram for L4::lpc::Msg::False:



Additional Inherited Members

Public Types inherited from [L4::Types::Bool< false >](#)

- typedef [Bool< V >](#) **type**
The meta type itself.

16.139.1 Detailed Description

[False](#) meta value.

Definition at line [296](#) of file [types](#).

The documentation for this struct was generated from the following file:

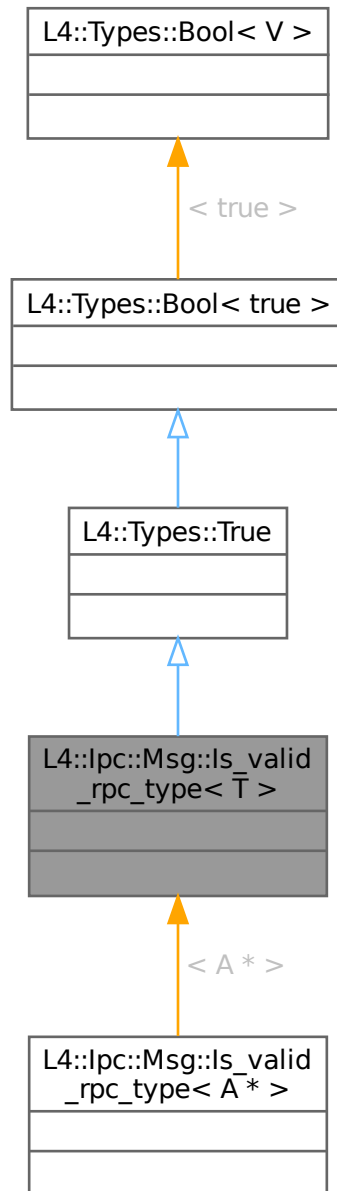
- [l4/sys/cxx/types](#)

16.140 L4::lpc::Msg::ls_valid_rpc_type< T > Struct Template Reference

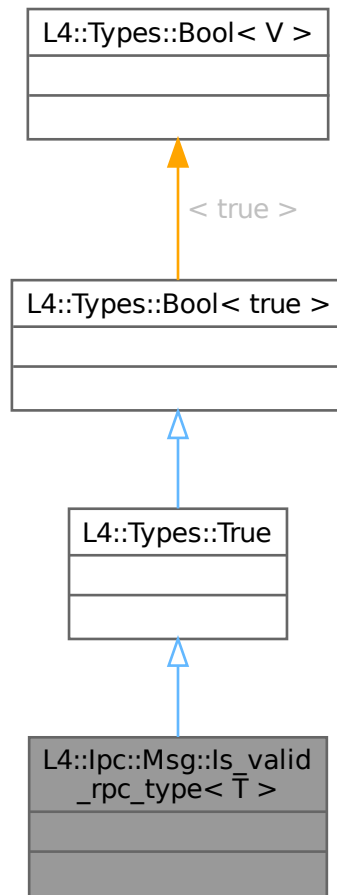
Type trait defining a valid RPC parameter type.

```
#include <ipc_basics>
```

Inheritance diagram for L4::lpc::Msg::ls_valid_rpc_type< T >:



Collaboration diagram for `L4::lpc::Msg::ls_valid_rpc_type< T >`:



Additional Inherited Members

Public Types inherited from `L4::Types::Bool< true >`

- typedef `Bool< V >` type
The meta type itself.

16.140.1 Detailed Description

```
template<typename T>
struct L4::lpc::Msg::ls_valid_rpc_type< T >
```

Type trait defining a valid RPC parameter type.

Definition at line 339 of file `ipc_basics`.

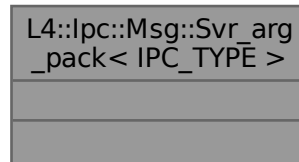
The documentation for this struct was generated from the following file:

- `I4/sys/cxx/ipc_basics`

16.141 L4::lpc::Msg::Svr_arg_pack< IPC_TYPE > Struct Template Reference

Server-side RPC arguments data structure used to provide arguments to the server-side implementation of an RPC function.

Collaboration diagram for L4::lpc::Msg::Svr_arg_pack< IPC_TYPE >:



16.141.1 Detailed Description

```
template<typename IPC_TYPE>
struct L4::lpc::Msg::Svr_arg_pack< IPC_TYPE >
```

Server-side RPC arguments data structure used to provide arguments to the server-side implementation of an RPC function.

Definition at line 144 of file [ipc_server](#).

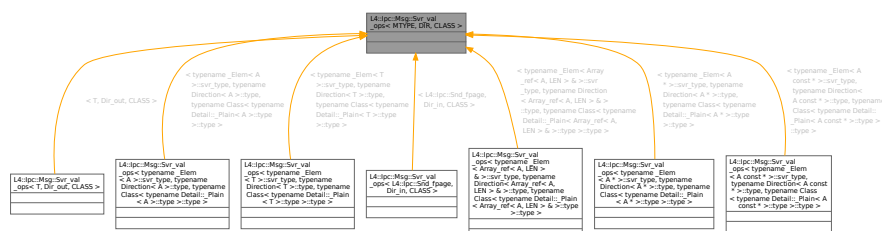
The documentation for this struct was generated from the following file:

- [l4/sys/cxx/ipc_server](#)

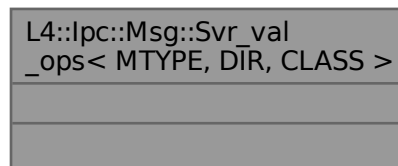
16.142 L4::lpc::Msg::Svr_val_ops< MTYPE, DIR, CLASS > Struct Template Reference

Defines client-side handling of `MTYPE' as RPC argument.

Inheritance diagram for L4::lpc::Msg::Svr_val_ops< MTYPE, DIR, CLASS >:



Collaboration diagram for L4::lpc::Msg::Svr_val_ops< MTYPE, DIR, CLASS >:



16.142.1 Detailed Description

```
template<typename MTYPE, typename DIR, typename CLASS>
struct L4::lpc::Msg::Svr_val_ops< MTYPE, DIR, CLASS >
```

Defines client-side handling of `MTYPE` as RPC argument.

Template Parameters

<i>MTYPE</i>	Elem<T>::arg_type (where T is the type used in the RPC definition)
<i>DIR</i>	Dir_in (client -> server), or Dir_out (server -> client)
<i>CLASS</i>	Cls_data , Cls_item , or Cls_buffer */ template<typename MTYPE, typename DIR, typename CLASS> struct Clnt_val_ops;

```
template<typename T> struct Clnt_noops { template<typename A, typename B> static constexpr int to_msg(char
*, unsigned offset, unsigned, T, A, B) noexcept { return offset; } }
```

```
/ copy data from the message to the client reference template<typename A, typename B> static constexpr int
from_msg(char *, unsigned offset, unsigned, long, T const &, A, B) noexcept { return offset; } };
```

```
template<typename T> struct Svr_noops { template<typename A, typename B> static constexpr int from_svr(char
*, unsigned offset, unsigned, long, T, A, B) noexcept { return offset; } }
```

```
/ copy data from the message to the client reference template<typename A, typename B> static constexpr int
to_svr(char *, unsigned offset, unsigned, T, A, B) noexcept { return offset; } };
```

```
template<typename MTYPE, typename CLASS> struct Clnt_val_ops<MTYPE, Dir_in, CLASS> : Clnt_noops<↵
MTYPE> { using Clnt_noops<MTYPE>::to_msg; / Copy a T into the message static int to_msg(char *msg, un-
signed offset, unsigned limit, MTYPE arg, Dir_in, CLASS) noexcept { return msg_add<MTYPE>(msg, offset, limit,
arg); } };
```

```
template<typename MTYPE, typename CLASS> struct Clnt_val_ops<MTYPE, Dir_out, CLASS> : Clnt_↵
noops<MTYPE> { using Clnt_noops<MTYPE>::from_msg; / copy data from the message to the client reference
static int from_msg(char *msg, unsigned offset, unsigned limit, long, MTYPE &arg, Dir_out, CLASS) noexcept {
return msg_get<MTYPE>(msg, offset, limit, arg); } };
```

/** Defines server-side handling for MTYPE server arguments.

Template Parameters

<i>MTYPE</i>	Elem<T>::svr_type (where T is the type used in the RPC definition)
<i>DIR</i>	Dir_in (client -> server), or Dir_out (server -> client)
<i>CLASS</i>	Cls_data , Cls_item , or Cls_buffer

Definition at line [264](#) of file [ipc_basics](#).

The documentation for this struct was generated from the following file:

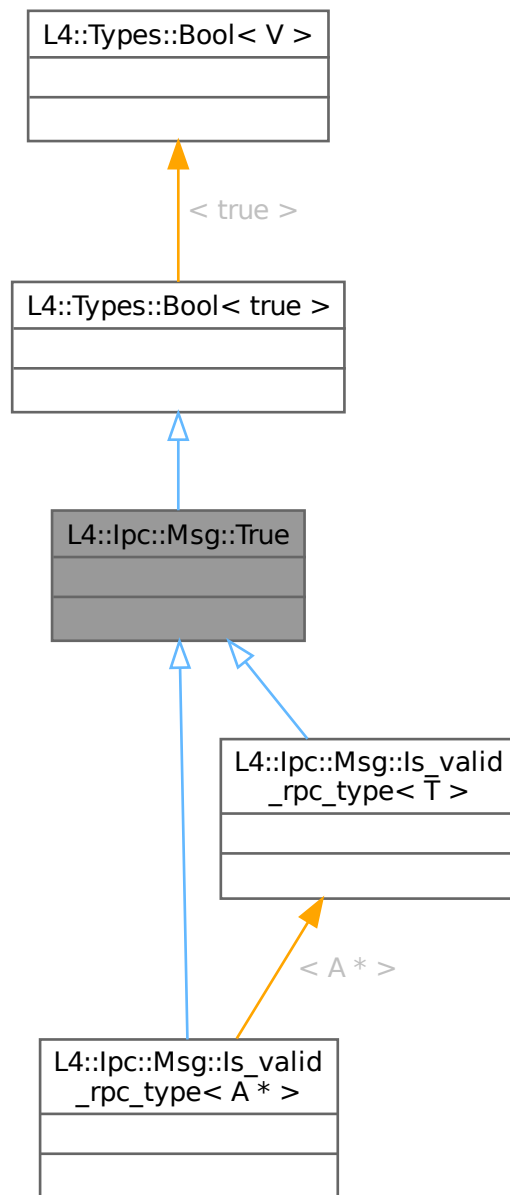
- [l4/sys/cxx/ipc_basics](#)

16.143 L4::lpc::Msg::True Struct Reference

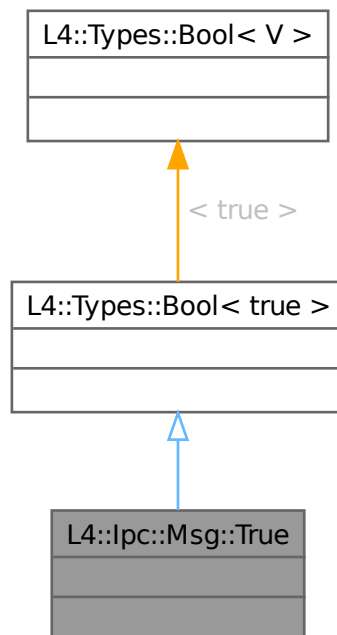
[True](#) meta value.

```
#include <types>
```

Inheritance diagram for L4::lpc::Msg::True:



Collaboration diagram for L4::lpc::Msg::True:



Additional Inherited Members

Public Types inherited from `L4::Types::Bool< true >`

- typedef `Bool< V > type`
The meta type itself.

16.143.1 Detailed Description

`True` meta value.

Definition at line 300 of file `types`.

The documentation for this struct was generated from the following file:

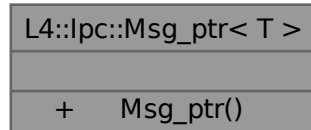
- `l4/sys/cxx/types`

16.144 L4::lpc::Msg_ptr< T > Class Template Reference

Pointer to an element of type T in an [lpc::lstream](#).

```
#include <ipc_stream>
```

Collaboration diagram for L4::lpc::Msg_ptr< T >:



Public Member Functions

- [Msg_ptr](#) (T *&p)

Create a [Msg_ptr](#) object that set pointer p to point into the message buffer.

16.144.1 Detailed Description

```
template<typename T>
class L4::lpc::Msg_ptr< T >
```

Pointer to an element of type T in an [lpc::lstream](#).

This wrapper can be used to extract an element of type T from an [lpc::lstream](#), whereas the data is not copied out, but a pointer into the message buffer itself is returned. With is mechanism it is possible to avoid an extra copy of large data structures from a received IPC message, instead the returned pointer gives direct access to the data in the message.

See [msg_ptr\(\)](#).

Definition at line 229 of file [ipc_stream](#).

16.144.2 Constructor & Destructor Documentation

16.144.2.1 Msg_ptr()

```
template<typename T>
L4::lpc::Msg_ptr< T >::Msg_ptr (
    T *& p) [inline], [explicit]
```

Create a [Msg_ptr](#) object that set pointer p to point into the message buffer.

Parameters

p The pointer that is adjusted to point into the message buffer.

Definition at line 240 of file [ipc_stream](#).

The documentation for this class was generated from the following file:

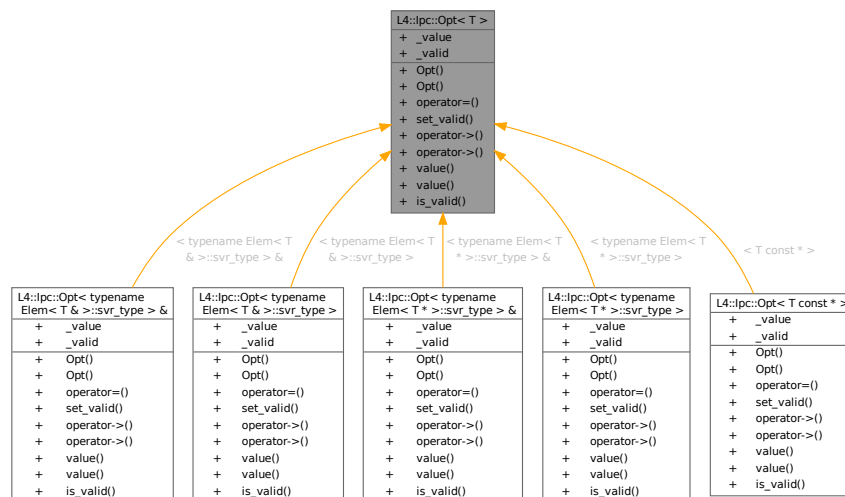
- [l4/cxx/ipc_stream](#)

16.145 L4::lpc::Opt< T > Struct Template Reference

Attribute for defining an optional RPC argument.

```
#include <ipc_types>
```

Inheritance diagram for L4::lpc::Opt< T >:



Collaboration diagram for L4::lpc::Opt< T >:

L4::lpc::Opt< T >
+ _value
+ _valid
+ Opt()
+ Opt()
+ operator=()
+ set_valid()
+ operator->()
+ operator->()
+ value()
+ value()
+ is_valid()

Public Member Functions

- **Opt** () noexcept
Make an absent optional argument.
- **Opt** (T **value**) noexcept
Make a present optional argument with the given value.
- **Opt** & **operator=** (T **value**) noexcept
Assign a value to the optional argument (makes the argument present).
- void **set_valid** (bool valid=true) noexcept
Set the argument to present or absent.
- T * **operator->** () noexcept
Get the pointer to the value.
- T const * **operator->** () const noexcept
Get the const pointer to the value.
- T **value** () const noexcept
Get the value.
- T & **value** () noexcept
Get the value.
- bool **is_valid** () const noexcept
Get true if present, false if not.

Data Fields

- T **_value**
The value.
- bool **_valid**
True if the optional argument is present, false else.

16.145.1 Detailed Description

```
template<typename T>  
struct L4::lpc::Opt< T >
```

Attribute for defining an optional RPC argument.

Definition at line 136 of file [ipc_types](#).

The documentation for this struct was generated from the following file:

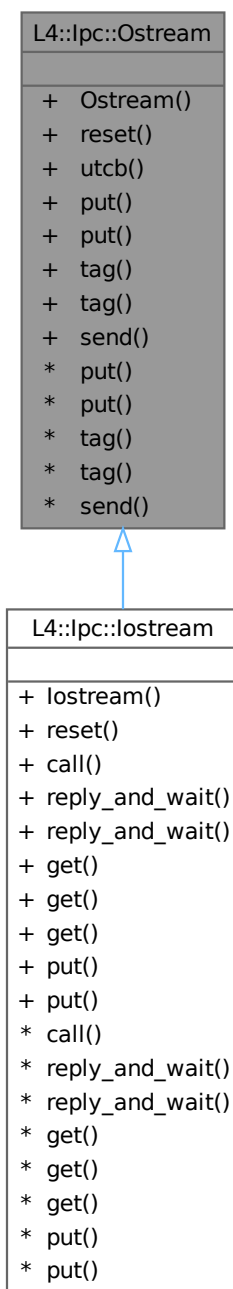
- [l4/sys/cxx/ipc_types](#)

16.146 L4::lpc::Ostream Class Reference

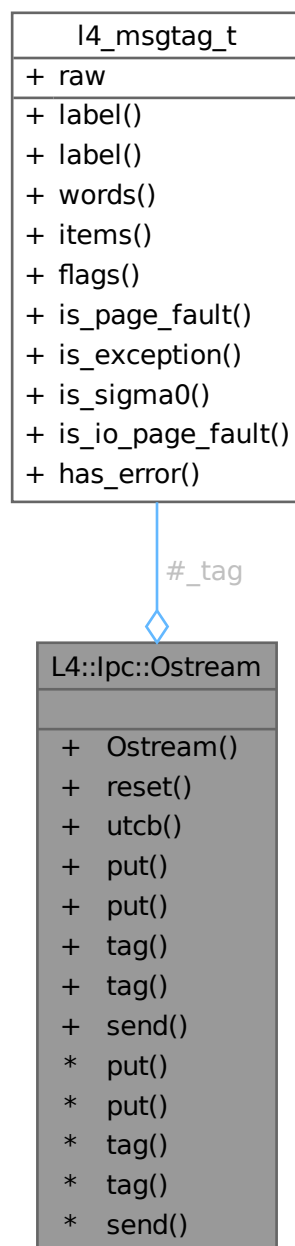
Output stream for IPC marshalling.

```
#include <ipc_stream>
```

Inheritance diagram for L4::ipc::Ostream:



Collaboration diagram for L4::lpc::Ostream:



Public Member Functions

- **Ostream** ([l4_utcb_t](#) *[utcb](#))
Create an IPC output stream using the given message buffer [utcb](#).
- void **reset** ()
Reset the stream to empty, same state as a newly created stream.
- [l4_utcb_t](#) * **utcb** () const

Return utcb pointer.

Get/Put functions.

These functions are basically used to implement the insertion operators (<<) and should not be called directly.

- `template<typename T>`
`bool put (T *buf, unsigned long size)`
Put an array with `size` elements of type `T` into the stream.
- `template<typename T>`
`bool put (T const &v)`
Insert an element of type `T` into the stream.
- `l4_msgtag_t tag () const`
Extract the `L4` message tag from the stream.
- `l4_msgtag_t & tag ()`
Extract a reference to the `L4` message tag from the stream.

IPC operations.

- `l4_msgtag_t send (l4_cap_idx_t dst, long proto=0, unsigned flags=0)`
Send the message via IPC to the given receiver.

16.146.1 Detailed Description

Output stream for IPC marshalling.

`lpc::Ostream` is part of the dynamic IPC marshalling infrastructure, as well as `lpc::Istream` and `lpc::lostream`.

`lpc::Ostream` is an output stream supporting insertion of values into an IPC message buffer. A IPC message can be marshalled using the usual insertion operator <<, see [IPC stream operators](#) .

There exist some special wrapper classes to insert arrays (see `lpc::Buf_cp_out`) and indirect strings (see `Msg_↔_out_buffer` and `Msg_io_buffer`).

Definition at line 623 of file [ipc_stream](#).

16.146.2 Member Function Documentation

16.146.2.1 put() [1/2]

```
template<typename T>
bool L4::Ipc::Ostream::put (
    T * buf,
    unsigned long size) [inline]
```

Put an array with `size` elements of type `T` into the stream.

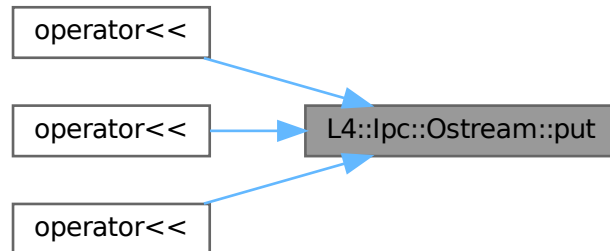
Parameters

<i>buf</i>	A pointer to the array to insert into the buffer.
<i>size</i>	The number of elements in the array.

Definition at line 660 of file [ipc_stream](#).

Referenced by [operator<<\(\)](#), [operator<<\(\)](#), and [operator<<\(\)](#).

Here is the caller graph for this function:



16.146.2.2 put() [2/2]

```
template<typename T>
bool L4::Ipc::Ostream::put (
    T const & v) [inline]
```

Insert an element of type T into the stream.

Parameters

<i>v</i>	The element to insert.
----------	------------------------

Definition at line 678 of file [ipc_stream](#).

16.146.2.3 send()

```
l4_msgtag_t L4::Ipc::Ostream::send (
    l4_cap_idx_t dst,
    long proto = 0,
    unsigned flags = 0) [inline]
```

Send the message via IPC to the given receiver.

Parameters

<i>dst</i>	The destination for the message.
<i>proto</i>	Protocol to use.

<i>flags</i>	Flags to use.
--------------	---------------

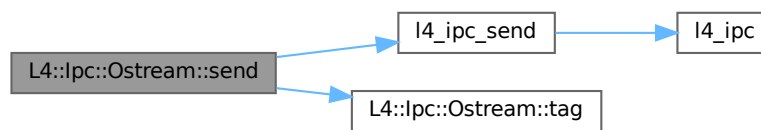
Returns

The syscall return tag.

Definition at line 959 of file [ipc_stream](#).

References [L4_IPC_NEVER](#), [l4_ipc_send\(\)](#), [L4_MSGTAG_FLAGS](#), and [tag\(\)](#).

Here is the call graph for this function:

**16.146.2.4 tag() [1/2]**

```
l4_msgtag_t & L4::Ipc::Ostream::tag () [inline]
```

Extract a reference to the [L4](#) message tag from the stream.

Returns

A reference to the [L4](#) message tag.

Definition at line 713 of file [ipc_stream](#).

16.146.2.5 tag() [2/2]

```
l4_msgtag_t L4::Ipc::Ostream::tag () const [inline]
```

Extract the [L4](#) message tag from the stream.

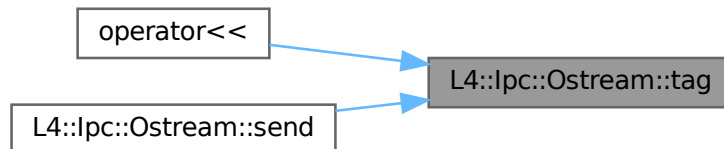
Returns

The extracted [L4](#) message tag.

Definition at line [706](#) of file [ipc_stream](#).

Referenced by [operator<<\(\)](#), and [send\(\)](#).

Here is the caller graph for this function:



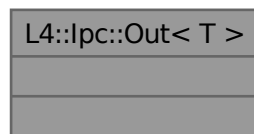
The documentation for this class was generated from the following file:

- [l4/cxx/ipc_stream](#)

16.147 L4::lpc::Out< T > Struct Template Reference

Mark an argument as a output value in an RPC signature.

Collaboration diagram for L4::lpc::Out< T >:



16.147.1 Detailed Description

```
template<typename T>
struct L4::lpc::Out< T >
```

Mark an argument as a output value in an RPC signature.

Template Parameters

<i>T</i>	The original type of the argument.
----------	------------------------------------

Note

The use of `Out<>` is usually not needed, because typical out-put data types in C++ (pointers to non-const objects or non-const references are interpreted as output values anyway. However, there are some data types, such as returned capabilities that can be marked as such by using `Out<>`.

Definition at line 31 of file [ipc_types](#).

The documentation for this struct was generated from the following file:

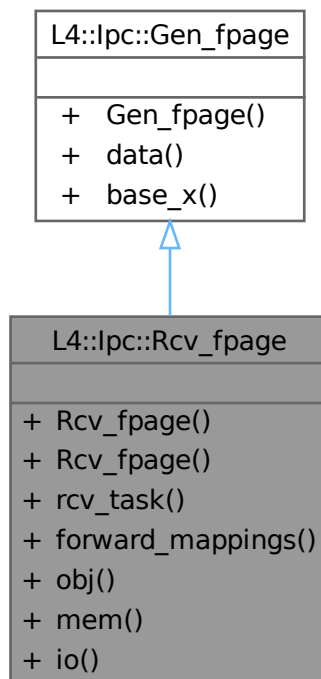
- [l4/sys/cxx/ipc_types](#)

16.148 L4::lpc::Rcv_fpage Class Reference

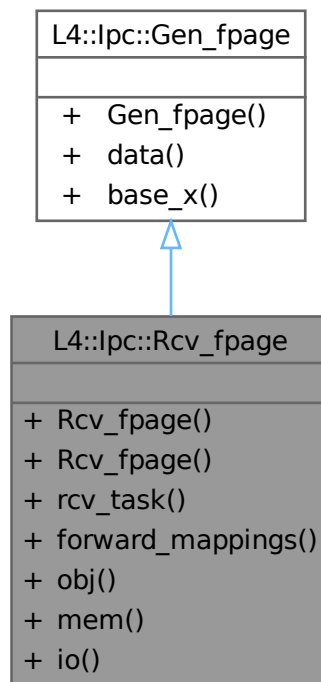
Non-small receive item.

```
#include <ipc_types>
```

Inheritance diagram for L4::lpc::Rcv_fpage:



Collaboration diagram for L4::lpc::Rcv_fpage:



Public Member Functions

- **Rcv_fpage** () noexcept
Construct a void receive item.
- **Rcv_fpage** (l4_fpage_t const &fp, l4_addr_t snd_base=0, l4_cap_idx_t rcv_task=L4_INVALID_CAP) noexcept
Construct a non-small receive item.
- **l4_cap_idx_t rcv_task** () const
Get the capability index of the destination task for received capabilities.
- **bool forward_mappings** () const noexcept
Check if rcv_task() shall be used as destination for received capabilities.

Public Member Functions inherited from L4::lpc::Gen_fpage

- **Gen_fpage** (l4_umword_t base, l4_umword_t data) noexcept
Construct from raw values.
- **l4_umword_t data** () const noexcept
Return the raw flexpage descriptor.
- **l4_umword_t base_x** () const noexcept
Return the raw base descriptor.

Static Public Member Functions

- static [Rcv_fpage obj](#) ([l4_cap_idx_t](#) base, int order, [l4_addr_t](#) snd_base=0, [L4::Cap](#)< void > [rcv_task](#)=[L4::Cap](#)< void >::Invalid) noexcept
Construct a non-small receive item for the object space.
- static [Rcv_fpage mem](#) ([l4_addr_t](#) base, int order, [l4_addr_t](#) snd_base=0, [L4::Cap](#)< void > [rcv_task](#)=[L4::Cap](#)< void >::Invalid) noexcept
Construct a receive item for the memory space.
- static [Rcv_fpage io](#) (unsigned long base, int order, [l4_addr_t](#) snd_base=0, [L4::Cap](#)< void > [rcv_task](#)=[L4::Cap](#)< void >::Invalid) noexcept
Construct a receive item for the I/O port space.

Additional Inherited Members

Public Types inherited from [L4::lpc::Gen_fpage](#)

- enum [Type](#) { [Special](#) = [L4_FPAGE_SPECIAL](#) << 4 , [Memory](#) = [L4_FPAGE_MEMORY](#) << 4 , [Io](#) = [L4_FPAGE_IO](#) << 4 , [Obj](#) = [L4_FPAGE_OBJ](#) << 4 }
- Type of mapping object, see [L4_fpage_type](#).*

16.148.1 Detailed Description

Non-small receive item.

This class represents a non-small receive item. A receive item is a message item in the buffer registers of the UTCB of the receiver (see [l4_utcb_br\(\)](#)).

Definition at line 544 of file [ipc_types](#).

16.148.2 Constructor & Destructor Documentation

16.148.2.1 [Rcv_fpage\(\)](#)

```
L4::Ipc::Rcv_fpage::Rcv_fpage (
    l4\_fpage\_t const & fp,
    l4\_addr\_t snd_base = 0,
    l4\_cap\_idx\_t rcv_task = L4\_INVALID\_CAP) [inline], [noexcept]
```

Construct a non-small receive item.

Parameters

<i>fp</i>	Flexpage defining where and which kind of capabilities may be received.
<i>snd_base</i>	Reserved; should be zero.
<i>rcv_task</i>	Optional destination task for received capabilities. If invalid, capabilities are received in the invoking task.

Definition at line 561 of file [ipc_types](#).

References [L4_INVALID_CAP](#), [L4_ITEM_MAP](#), [L4_RCV_ITEM_FORWARD_MAPPINGS](#), and [rcv_task\(\)](#).

Here is the call graph for this function:



16.148.3 Member Function Documentation

16.148.3.1 io()

```

Rcv_fpage L4::Ipc::Rcv_fpage::io (
    unsigned long base,
    int order,
    l4_addr_t snd_base = 0,
    L4::Cap< void > rcv_task = L4::Cap<void>::Invalid) [inline], [static], [noexcept]
  
```

Construct a receive item for the I/O port space.

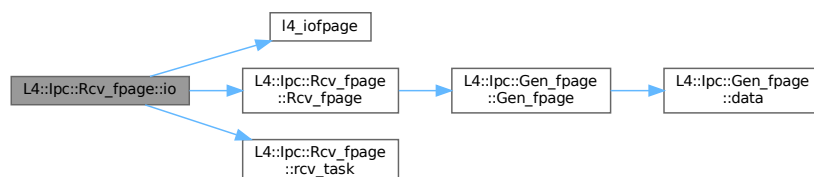
Parameters

<i>base</i>	Start of flexpage (see l4_iofpage()).
<i>order</i>	Log ₂ size of flexpage (see l4_iofpage()).
<i>snd_base</i>	Reserved; should be zero.
<i>rcv_task</i>	Optional destination task for received capabilities. If invalid, capabilities are received in the invoking task.

Definition at line 609 of file [ipc_types](#).

References [L4::Cap_base::Invalid](#), [l4_iofpage\(\)](#), [Rcv_fpage\(\)](#), and [rcv_task\(\)](#).

Here is the call graph for this function:



16.148.3.2 mem()

```
Rcv_fpage L4::Ipc::Rcv_fpage::mem (
    l4_addr_t base,
    int order,
    l4_addr_t snd_base = 0,
    L4::Cap< void > rcv_task = L4::Cap<void>::Invalid) [inline], [static], [noexcept]
```

Construct a receive item for the memory space.

Parameters

<i>base</i>	Start of flexpage (see l4_fpage()).
<i>order</i>	Log ₂ size of flexpage (see l4_fpage()).
<i>snd_base</i>	Reserved; should be zero.
<i>rcv_task</i>	Optional destination task for received capabilities. If invalid, capabilities are received in the invoking task.

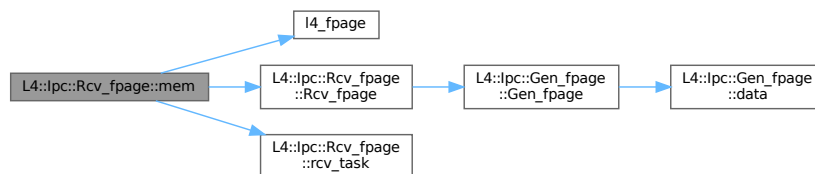
Examples

[examples/libs/l4re/streammap/client.cc](#).

Definition at line 594 of file [ipc_types](#).

References [L4::Cap_base::Invalid](#), [l4_fpage\(\)](#), [Rcv_fpage\(\)](#), and [rcv_task\(\)](#).

Here is the call graph for this function:



16.148.3.3 obj()

```
Rcv_fpage L4::Ipc::Rcv_fpage::obj (
    l4_cap_idx_t base,
    int order,
    l4_addr_t snd_base = 0,
    L4::Cap< void > rcv_task = L4::Cap<void>::Invalid) [inline], [static], [noexcept]
```

Construct a non-small receive item for the object space.

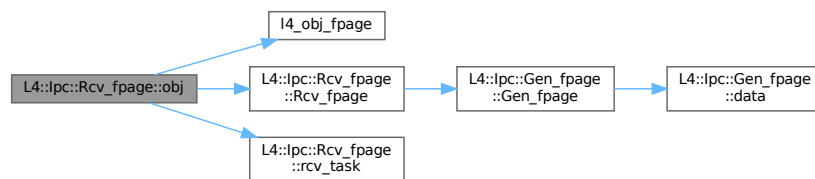
Parameters

<i>base</i>	Start of flexpage (see l4_obj_fpage()).
<i>order</i>	\log_2 size of flexpage (see l4_obj_fpage()).
<i>snd_base</i>	Reserved; should be zero.
<i>rcv_task</i>	Optional destination task for received capabilities. If invalid, capabilities are received in the invoking task.

Definition at line 578 of file [ipc_types](#).

References [L4::Cap_base::Invalid](#), [l4_obj_fpage\(\)](#), [Rcv_fpage\(\)](#), and [rcv_task\(\)](#).

Here is the call graph for this function:



16.148.3.4 rcv_task()

```
l4_cap_idx_t L4::lpc::Rcv_fpage::rcv_task () const [inline]
```

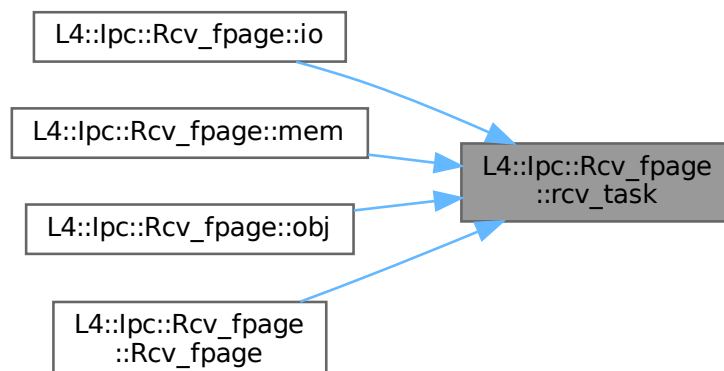
Get the capability index of the destination task for received capabilities.

Only relevant if [forward_mappings\(\)](#) is true.

Definition at line 620 of file [ipc_types](#).

Referenced by [io\(\)](#), [mem\(\)](#), [obj\(\)](#), and [Rcv_fpage\(\)](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

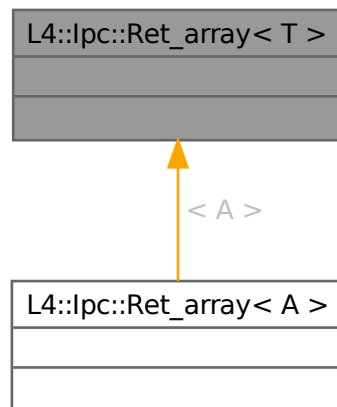
- [l4/sys/cxx/ipc_types](#)

16.149 L4::ipc::Ret_array< T > Struct Template Reference

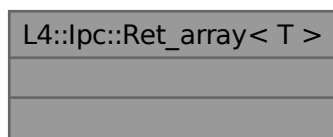
Dynamically sized output array of type T.

```
#include <ipc_ret_array>
```

Inheritance diagram for L4::ipc::Ret_array< T >:



Collaboration diagram for L4::ipc::Ret_array< T >:



16.149.1 Detailed Description

```
template<typename T>
struct L4::ipc::Ret_array< T >
```

Dynamically sized output array of type T.

Template Parameters

<i>T</i>	The data-type of each array element.
----------	--------------------------------------

[Ret_array<>](#) is a special dynamically sized output array where the number of transmitted elements is passed in the return value of the call (if positive)

Definition at line 23 of file [ipc_ret_array](#).

The documentation for this struct was generated from the following file:

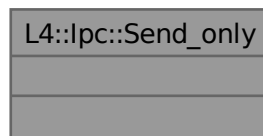
- [l4/sys/cxx/ipc_ret_array](#)

16.150 L4::lpc::Send_only Struct Reference

RPC attribute for a send-only RPC.

```
#include <ipc_iface>
```

Collaboration diagram for L4::lpc::Send_only:



16.150.1 Detailed Description

RPC attribute for a send-only RPC.

This class can be used as FLAGS parameter to [L4::lpc::Msg::Rpc_call](#) and [L4::lpc::Msg::Rpc_inline_call](#) templates and declares the RPC to use send-only semantics and timeouts.

Examples:

```
L4\_RPC(long, send, (unsigned value), L4::Ipc::Send_only);
```

Definition at line 287 of file [ipc_iface](#).

The documentation for this struct was generated from the following file:

- [l4/sys/cxx/ipc_iface](#)

16.151 L4::lpc::Small_buf Class Reference

A receive item for receiving a single object capability.

```
#include <ipc_types>
```

Collaboration diagram for L4::lpc::Small_buf:

L4::lpc::Small_buf
<ul style="list-style-type: none"> + Small_buf() + Small_buf() + raw()

Public Member Functions

- [Small_buf](#) ([L4::Cap](#)< void > cap, unsigned long flags=0) noexcept
Create a receive item from a C++ cap.
- [Small_buf](#) ([l4_cap_idx_t](#) cap, unsigned long flags=0) noexcept
Create a receive item from a C cap.
- [l4_umword_t raw](#) () const noexcept
Return the raw data.

16.151.1 Detailed Description

A receive item for receiving a single object capability.

This class is the main abstraction for receiving object capabilities via [lpc::lstream](#). To receive an object capability, an instance of [Small_buf](#) that refers to an empty capability slot must be inserted into the [lpc::lstream](#) before the receive operation.

Definition at line 257 of file [ipc_types](#).

16.151.2 Constructor & Destructor Documentation

16.151.2.1 Small_buf() [1/2]

```
L4::Ipc::Small_buf::Small_buf (
    L4::Cap< void > cap,
    unsigned long flags = 0) [inline], [explicit], [noexcept]
```

Create a receive item from a C++ cap.

Parameters

<i>cap</i>	Capability slot where to save the capability.
<i>flags</i>	Receive buffer flags, see l4_msg_item_consts_t . L4_RCV_ITEM_SINGLE_CAP will always be set.

Definition at line 267 of file [ipc_types](#).

References [L4_RCV_ITEM_SINGLE_CAP](#).

16.151.2.2 Small_buf() [2/2]

```
L4::Ipc::Small_buf::Small_buf (  
    l4\_cap\_idx\_t cap,  
    unsigned long flags = 0) [inline], [explicit], [noexcept]
```

Create a receive item from a C cap.

Parameters

<i>cap</i>	Capability slot where to save the capability.
<i>flags</i>	Receive buffer flags, see l4_msg_item_consts_t . L4_RCV_ITEM_SINGLE_CAP will always be set.

Definition at line 274 of file [ipc_types](#).

References [L4_RCV_ITEM_SINGLE_CAP](#).

The documentation for this class was generated from the following file:

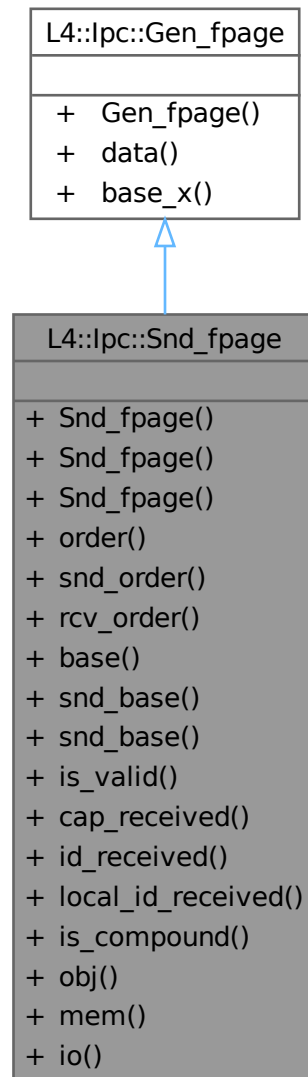
- [l4/sys/cxx/ipc_types](#)

16.152 L4::Ipc::Snd_fpage Class Reference

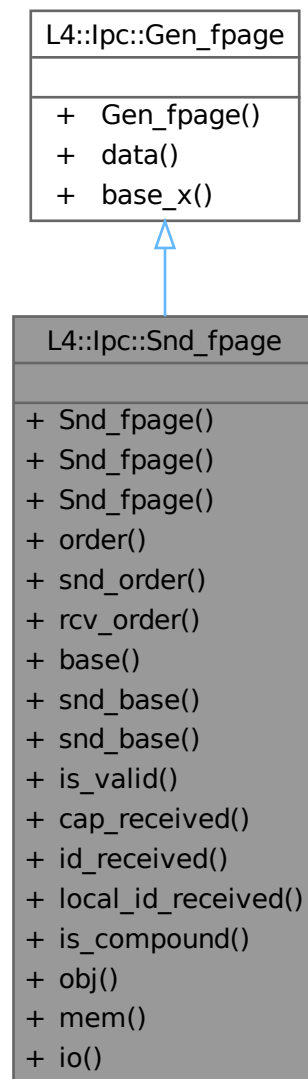
Send item or return item.

```
#include <ipc_types>
```

Inheritance diagram for L4::lpc::Snd_fpage:



Collaboration diagram for L4::lpc::Snd_fpage:



Public Types

- enum [Map_type](#) { [Map](#) = L4_MAP_ITEM_MAP , [Grant](#) = L4_MAP_ITEM_GRANT }
(Defined for send items only.) Kind of mapping.
- enum [Cacheopt](#) { [None](#) = 0 , [Cached](#) = L4_FPAGE_CACHEABLE << 4 , [Buffered](#) = L4_FPAGE_BUFFERABLE << 4 , [Uncached](#) = L4_FPAGE_UNCACHEABLE << 4 }
(Defined for memory send items only.) Caching options, see [l4_fpage_cacheability_opt_t](#).
- enum [Continue](#) { [Single](#) = 0 , [Last](#) = 0 , [More](#) = L4_ITEM_CONT , [Compound](#) = L4_ITEM_CONT }
Specify if the following item is associated with the same receive item as this one, see [L4_ITEM_CONT](#).

Public Types inherited from [L4::lpc::Gen_fpage](#)

- enum [Type](#) { [Special](#) = L4_FPAGE_SPECIAL << 4 , [Memory](#) = L4_FPAGE_MEMORY << 4 , [Io](#) = L4_FPAGE_IO << 4 , [Obj](#) = L4_FPAGE_OBJ << 4 }

Type of mapping object, see [L4_fpage_type](#).

Public Member Functions

- [Snd_fpage](#) ([l4_umword_t](#) base=0, [l4_umword_t](#) data=0) noexcept
Construct from raw values.
- [Snd_fpage](#) ([l4_fpage_t](#) const &fp, [l4_addr_t](#) snd_base=0, [Map_type](#) map_type=[Map](#), [Cacheopt](#) cache=[None](#), [Continue](#) cont=[Last](#)) noexcept
Construct a send item for the memory space.
- [Snd_fpage](#) ([L4::Cap](#)< void > cap, unsigned rights, [Map_type](#) map_type=[Map](#)) noexcept
Construct a send item for the object space.
- unsigned [order](#) () const noexcept
(Defined only if send item or if [local_id_received\(\)](#) is true.) Get log₂ size.
- unsigned [snd_order](#) () const noexcept
(Defined only if send item or if [local_id_received\(\)](#) is true.) Get log₂ size.
- unsigned [rcv_order](#) () const noexcept
(Defined for return items only.) Get log₂ size.
- [l4_addr_t](#) [base](#) () const noexcept
(Defined only if send item or if [local_id_received\(\)](#) is true.) Get the start of the item (i.e., the start of its flexpage).
- [l4_addr_t](#) [snd_base](#) () const noexcept
Get the position in receive window for the case that this item has a different size than the corresponding receive item.
- void [snd_base](#) ([l4_addr_t](#) b) noexcept
Set the position in receive window for the case that this item has a different size than the corresponding receive item.
- bool [is_valid](#) () const noexcept
Check if the capability is valid.
- bool [cap_received](#) () const noexcept
(Defined for return items only.) Check if at least one object capability has been mapped for this item.
- bool [id_received](#) () const noexcept
(Defined for return items only.) Check if an IPC gate label has been received instead of a mapping.
- bool [local_id_received](#) () const noexcept
(Defined for return items only.) Check if a raw object flexpage has been received instead of a mapping.
- bool [is_compound](#) () const noexcept
Check if the item has the compound bit set, see [Continue](#).

Public Member Functions inherited from [L4::lpc::Gen_fpage](#)

- [Gen_fpage](#) ([l4_umword_t](#) base, [l4_umword_t](#) data) noexcept
Construct from raw values.
- [l4_umword_t](#) [data](#) () const noexcept
Return the raw flexpage descriptor.
- [l4_umword_t](#) [base_x](#) () const noexcept
Return the raw base descriptor.

Static Public Member Functions

- static [Snd_fpage obj](#) ([l4_cap_idx_t](#) base, int [order](#), unsigned char rights, [l4_addr_t](#) snd_base=0, [Map_type](#) map_type=[Map](#), [Continue](#) cont=[Last](#)) noexcept
Construct a send item for the object space.
- static [Snd_fpage mem](#) ([l4_addr_t](#) base, int [order](#), unsigned char rights, [l4_addr_t](#) snd_base=0, [Map_type](#) map_type=[Map](#), [Cacheopt](#) cache=[None](#), [Continue](#) cont=[Last](#)) noexcept
Construct a send item for the memory space.
- static [Snd_fpage io](#) (unsigned long base, int [order](#), unsigned char rights, [l4_addr_t](#) snd_base=0, [Map_type](#) map_type=[Map](#), [Continue](#) cont=[Last](#)) noexcept
Construct a send item for the I/O port space.

16.152.1 Detailed Description

Send item or return item.

This class represents a typed message item in the message registers of the UTCB. If it is provided by the sender, then it is a *send item*. If it is provided by the kernel during IPC, it is a *return item*.

Note that some members are dedicated for send items only or return items only.

Definition at line 323 of file [ipc_types](#).

16.152.2 Member Enumeration Documentation

16.152.2.1 Cacheopt

```
enum L4::Ipc::Snd_fpage::Cacheopt
```

(Defined for memory send items only.) Caching options, see [l4_fpage_cacheability_opt_t](#).

Enumerator

None	Copy options from sender.
Cached	Cacheability option to enable caches for the mapping.
Buffered	Cacheability option to enable buffered writes for the mapping.
Uncached	Cacheability option to disable caching for the mapping.

Definition at line 336 of file [ipc_types](#).

16.152.2.2 Continue

```
enum L4::Ipc::Snd_fpage::Continue
```

Specify if the following item is associated with the same receive item as this one, see [L4_ITEM_CONT](#).

Enumerator

Single	Inverse of Compound .
Last	Inverse of More .
More	Alias for Compound .
Compound	Denote that the following item shall be put into the same receive item as this one.

Definition at line [346](#) of file [ipc_types](#).

16.152.2.3 Map_type

```
enum L4::Ipc::Snd_fpage::Map_type
```

(Defined for send items only.) Kind of mapping.

Enumerator

Map	Flag as usual <i>map</i> operation.
Grant	<p>Flag as <i>grant</i> instead of <i>map</i> operation. This means, the sender delegates access to the receiver and the kernel removes the rights from the sender (basically a move operation). The mapping in the receiver gets the new parent of any child mappings of the mapping of the sender. Rights revocation via send item/flexpage is <i>not</i> guaranteed to be applied to descendant mappings in case of grant. See Spaces and Mappings for more details on map/grant.</p> <p>Note</p> <p>The grant operation is not performed if the resulting rights of the receiver mapping would not contain the L4_CAP_FPAGE_R bit (for object capabilities) or none of the L4_FPAGE_RWX bits (memory and IO ports). In that case, the mapping is not created in the receiver space and not removed from the sender space.</p> <p>If the removal of the whole mapping from the sender is not possible because the size of the mapped frame at the sender exceeds the size defined by the send or receive flexpage, the grant operation is turned into a regular map operation and the mapping is <i>not</i> removed from the sender. This would happen if, for example, a smaller part of an L4 superpage mapping shall be granted.</p>

Definition at line [328](#) of file [ipc_types](#).

16.152.3 Constructor & Destructor Documentation

16.152.3.1 Snd_fpage() [1/2]

```
L4::Ipc::Snd_fpage::Snd_fpage (
    l4_fpage_t const & fp,
    l4_addr_t snd_base = 0,
    Map_type map_type = Map,
    Cacheopt cache = None,
    Continue cont = Last) [inline], [noexcept]
```

Construct a send item for the memory space.

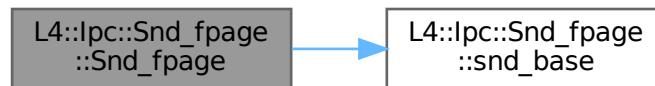
Parameters

<i>fp</i>	Memory flexpage defining which range and kind of capabilities shall be sent (see l4_fpage()).
<i>snd_base</i>	Position in receive window in case it has a different size than <i>fp</i> .
<i>map_type</i>	See Map_type .
<i>cache</i>	See Cacheopt .
<i>cont</i>	See Continue .

Definition at line 370 of file [ipc_types](#).

References [L4_ITEM_MAP](#), [Last](#), [Map](#), [None](#), and [snd_base\(\)](#).

Here is the call graph for this function:



16.152.3.2 Snd_fpage() [2/2]

```

L4::Ipc::Snd_fpage::Snd_fpage (
    L4::Cap< void > cap,
    unsigned rights,
    Map_type map_type = Map) [inline], [noexcept]
  
```

Construct a send item for the object space.

Parameters

<i>cap</i>	Capability to be sent.
<i>rights</i>	Permissions to be transferred. See L4_cap_fpage_rights and L4_obj_fpage_ctl .
<i>map_type</i>	See Map_type .

Definition at line 386 of file [ipc_types](#).

References [L4_ITEM_MAP](#), and [Map](#).

16.152.4 Member Function Documentation

16.152.4.1 cap_received()

```
bool L4::Ipc::Snd_fpage::cap_received () const [inline], [noexcept]
```

(Defined for return items only.) Check if at least one object capability has been mapped for this item.

The capabilities themselves can then be retrieved from the cap slots that have been provided in the receive operation.

Note

If this function returns `true` and the receive window covers more than one capability slot, then it is not possible to determine which slots actually got capabilities mapped from the sender.

If the received capabilities do not have type object (see [L4_FPAGE_OBJ](#)), then this function returns `false`.

Definition at line 496 of file [ipc_types](#).

16.152.4.2 id_received()

```
bool L4::Ipc::Snd_fpage::id_received () const [inline], [noexcept]
```

(Defined for return items only.) Check if an IPC gate label has been received instead of a mapping.

If the [L4_RCV_ITEM_LOCAL_ID](#) flag has been set by the receiver, the conditions for [local_id_received\(\)](#) do not apply, the sender sent an IPC gate capability, and the receiving thread is in the same task as the thread that is attached to the IPC gate, then no mapping is done for this item and only the bitwise OR (|) of the label of the IPC gate and the special and write permission ([L4_CAP_FPAGE_S](#) and [L4_CAP_FPAGE_W](#)) that would have been mapped is received.

The bitwise OR of the label and the permissions can be retrieved with [Gen_fpage::data\(\)](#).

Definition at line 512 of file [ipc_types](#).

16.152.4.3 io()

```
Snd_fpage L4::Ipc::Snd_fpage::io (
    unsigned long base,
    int order,
    unsigned char rights,
    l4_addr_t snd_base = 0,
    Map_type map_type = Map,
    Continue cont = Last) [inline], [static], [noexcept]
```

Construct a send item for the I/O port space.

Parameters

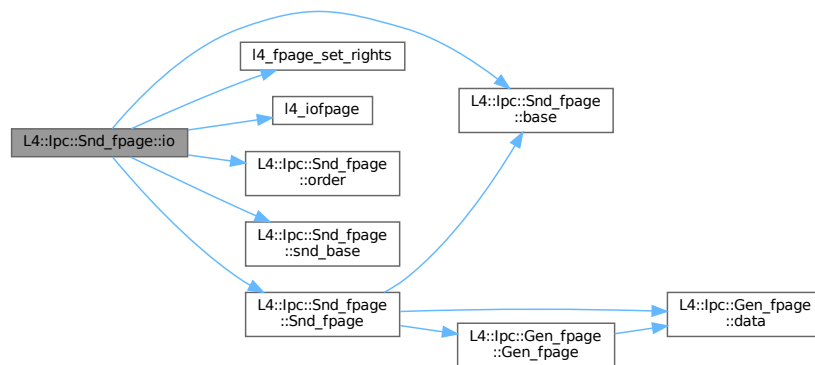
<i>base</i>	Start of flexpage (see l4_iofpage()).
-------------	--

<i>order</i>	Log ₂ size of flexpage (see l4_iofpage()).
<i>rights</i>	Permissions of flexpage (see L4_fpage_rights).
<i>snd_base</i>	Position in receive window in case it has a different size than $1 \ll order$.
<i>map_type</i>	See Map_type .
<i>cont</i>	See Continue .

Definition at line 445 of file [ipc_types](#).

References [base\(\)](#), [l4_fpage_set_rights\(\)](#), [l4_iofpage\(\)](#), [Last](#), [Map](#), [None](#), [order\(\)](#), [snd_base\(\)](#), and [Snd_fpage\(\)](#).

Here is the call graph for this function:



16.152.4.4 is_compound()

```
bool L4::Ipc::Snd_fpage::is_compound () const [inline], [noexcept]
```

Check if the item has the compound bit set, see [Continue](#).

A set compound bit means the next message item of the same type will be mapped to the same receive buffer as this message item.

Definition at line 535 of file [ipc_types](#).

16.152.4.5 local_id_received()

```
bool L4::Ipc::Snd_fpage::local_id_received () const [inline], [noexcept]
```

(Defined for return items only.) Check if a raw object flexpage has been received instead of a mapping.

If the [L4_RCV_ITEM_LOCAL_ID](#) flag has been set by the receiver, and sender and receiver are in the same task, then no mapping is done for this item and only the raw flexpage ([l4_fpage_t](#)) is received.

This function checks if this is the case and if it is an object flexpage.

The flexpage can be retrieved with [Gen_fpage::data\(\)](#).

Note

If a raw flexpage was received, but it does not have type object (see [L4_FPAGE_OBJ](#)), then this function returns `false`.

Definition at line 528 of file [ipc_types](#).

16.152.4.6 mem()

```
Snd_fpage L4::Ipc::Snd_fpage::mem (
    l4_addr_t base,
    int order,
    unsigned char rights,
    l4_addr_t snd_base = 0,
    Map_type map_type = Map,
    Cacheopt cache = None,
    Continue cont = Last) [inline], [static], [noexcept]
```

Construct a send item for the memory space.

Parameters

<i>base</i>	Start of flexpage (see l4_fpage()).
<i>order</i>	\log_2 size of flexpage (see l4_fpage()).
<i>rights</i>	Permissions of flexpage (see l4_fpage()).
<i>snd_base</i>	Position in receive window in case it has a different size than $1 \ll order$.
<i>map_type</i>	See Map_type .
<i>cache</i>	See Cacheopt .
<i>cont</i>	See Continue .

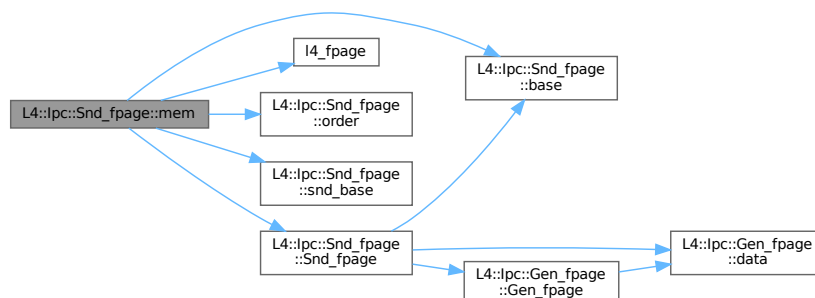
Examples

[examples/libs/l4re/streammap/server.cc](#).

Definition at line 424 of file [ipc_types](#).

References [base\(\)](#), [l4_fpage\(\)](#), [Last](#), [Map](#), [None](#), [order\(\)](#), [snd_base\(\)](#), and [Snd_fpage\(\)](#).

Here is the call graph for this function:



16.152.4.7 obj()

```
Snd_fpage L4::Ipc::Snd_fpage::obj (
    l4_cap_idx_t base,
    int order,
    unsigned char rights,
    l4_addr_t snd_base = 0,
    Map_type map_type = Map,
    Continue cont = Last) [inline], [static], [noexcept]
```

Construct a send item for the object space.

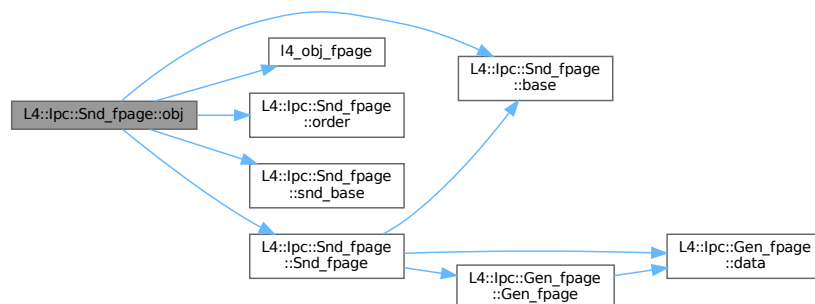
Parameters

<i>base</i>	Start of flexpage (see l4_obj_fpage()).
<i>order</i>	\log_2 size of flexpage (see l4_obj_fpage()).
<i>rights</i>	Permissions of flexpage (see l4_obj_fpage()).
<i>snd_base</i>	Position in receive window in case it has a different size than $1 \ll order$.
<i>map_type</i>	See Map_type .
<i>cont</i>	See Continue .

Definition at line 402 of file [ipc_types](#).

References [base\(\)](#), [l4_obj_fpage\(\)](#), [Last](#), [Map](#), [None](#), [order\(\)](#), [snd_base\(\)](#), and [Snd_fpage\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

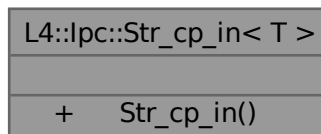
- [l4/sys/cxx/ipc_types](#)

16.153 L4::lpc::Str_cp_in< T > Class Template Reference

Abstraction for extracting a zero-terminated string from an [lpc::lstream](#).

```
#include <ipc_stream>
```

Collaboration diagram for L4::lpc::Str_cp_in< T >:



Public Member Functions

- [Str_cp_in](#) (T *v, unsigned long &size)
Create a buffer for extracting an array from an [lpc::lstream](#).

16.153.1 Detailed Description

```
template<typename T>
class L4::lpc::Str_cp_in< T >
```

Abstraction for extracting a zero-terminated string from an [lpc::lstream](#).

An instance of [Str_cp_in](#) can be used to extract a zero-terminated string an [lpc::lstream](#). The data from the received message is thereby copied to the given buffer and size is set to the number of characters found in the stream. The string is zero terminated in any circumstances. When the given buffer is smaller than the received string the last byte in the buffer will be the zero terminator. In the case the received string is shorter than the given buffer the zero termination will be placed behind the received data. This provides a zero-terminated result even in cases where the sender did not provide proper termination or in cases of too small receiver buffers.

See also

[str_cp_in\(\)](#).

Definition at line 178 of file [ipc_stream](#).

16.153.2 Constructor & Destructor Documentation

16.153.2.1 Str_cp_in()

```
template<typename T>
L4::lpc::Str_cp_in< T >::Str_cp_in (
    T * v,
    unsigned long & size) [inline]
```

Create a buffer for extracting an array from an [lpc::lstream](#).

Parameters

	<i>v</i>	The buffer for string.
<i>in, out</i>	<i>size</i>	Input: The number of bytes available in <i>v</i> Output: The number of bytes received (including the terminator).

Definition at line 189 of file [ipc_stream](#).

The documentation for this class was generated from the following file:

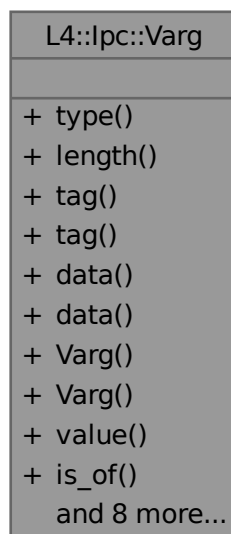
- [l4/cxx/ipc_stream](#)

16.154 L4::lpc::Varg Class Reference

Variably sized RPC argument.

```
#include <ipc_varg>
```

Collaboration diagram for L4::lpc::Varg:



Public Types

- typedef [l4_umword_t](#) **Tag**
The data type for the tag.

Public Member Functions

- L4_varg_type [type](#) () const
- unsigned [length](#) () const
Get the size of the RPC argument.
- [Tag](#) [tag](#) () const
- void [tag](#) ([Tag](#) tag)
Set [Varg](#) tag (usually from message).
- void [data](#) (char const *d)
Set [Varg](#) to indirect data value (usually in UTCB).
- char const * [data](#) () const
- [Varg](#) ()=default
Make uninitialized [Varg](#).
- [Varg](#) (L4_varg_type t, void const *v, int len)
Make an indirect varg.
- template<typename V>
Va_type< V >::Ret_value [value](#) () const
- template<typename T>
bool [is_of](#) () const
- bool [is_nil](#) () const
- bool [is_of_int](#) () const
- template<typename T>
bool [get_value](#) (typename Va_type< T >::Value *v) const
Get the value of the [Varg](#) as type T.
- template<typename T>
void [set_value](#) (void const *d)
Set to indirect value of type T.
- template<typename T>
void [set_direct_value](#) (T val, typename L4::Types::Enable_if< sizeof(T)<=sizeof(char const *)>, bool >::type=true)
Set to directly stored value of type T.
- template<typename T>
[Varg](#) (T const *[data](#))
Make [Varg](#) from indirect value (pointer).
- [Varg](#) (char const *[data](#))
Make [Varg](#) from null-terminated string.
- template<typename T>
[Varg](#) (T [data](#), typename L4::Types::Enable_if< sizeof(T)<=sizeof(char const *)>, bool >::type=true)
Make [Varg](#) from direct value.

16.154.1 Detailed Description

Variably sized RPC argument.

Definition at line 96 of file [ipc_varg](#).

16.154.2 Member Function Documentation

16.154.2.1 [data](#)()

```
char const * L4::Ipc::Varg::data () const [inline]
```

Returns

pointer to the data, also safe for direct data

Definition at line 123 of file [ipc_varg](#).

16.154.2.2 get_value()

```
template<typename T>
bool L4::Ipc::Varg::get_value (
    typename Va_type< T >::Value * v) const [inline]
```

Get the value of the [Varg](#) as type T.

Template Parameters

<i>T</i>	The expected type of the Varg .
----------	---

Parameters

<i>v</i>	Pointer to store the value
----------	----------------------------

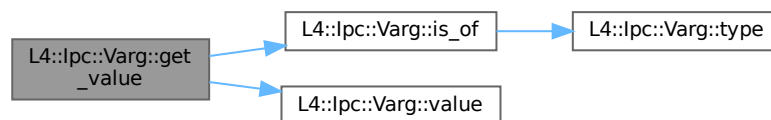
Returns

true when the [Varg](#) is of type T, false if not

Definition at line [185](#) of file [ipc_varg](#).

References [is_of\(\)](#), and [value\(\)](#).

Here is the call graph for this function:



16.154.2.3 is_nil()

```
bool L4::Ipc::Varg::is_nil () const [inline]
```

Returns

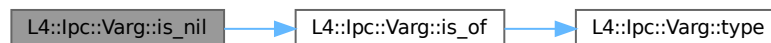
true if the [Varg](#) is of nil type.

Definition at line 172 of file [ipc_varg](#).

References [is_of\(\)](#).

Referenced by [L4::Ipc::Varg_list_ref::Iterator::equals\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**16.154.2.4 is_of()**

```
template<typename T>
bool L4::Ipc::Varg::is_of () const [inline]
```

Returns

true if the [Varg](#) is of type T

Definition at line 169 of file [ipc_varg](#).

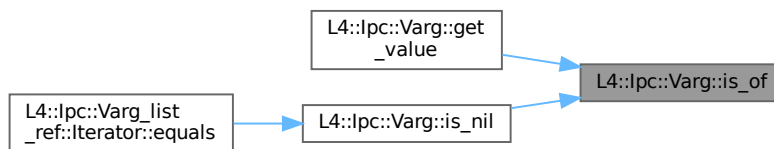
References [type\(\)](#).

Referenced by [get_value\(\)](#), and [is_nil\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.154.2.5 is_of_int()

```
bool L4::Ipc::Varg::is_of_int () const [inline]
```

Returns

true if the [Varg](#) is an integer type (signed or unsigned).

Definition at line 175 of file [ipc_varg](#).

References [type\(\)](#).

Here is the call graph for this function:



16.154.2.6 length()

```
unsigned L4::Ipc::Varg::length () const [inline]
```

Get the size of the RPC argument.

Returns

The size of the RPC argument

Definition at line 114 of file [ipc_varg](#).

16.154.2.7 tag()

```
Tag L4::Ipc::Varg::tag () const [inline]
```

Returns

the tag value (the Direct_data bit masked)

Definition at line 116 of file [ipc_varg](#).

Referenced by [tag\(\)](#).

Here is the caller graph for this function:



16.154.2.8 type()

```
L4_varg_type L4::Ipc::Varg::type () const [inline]
```

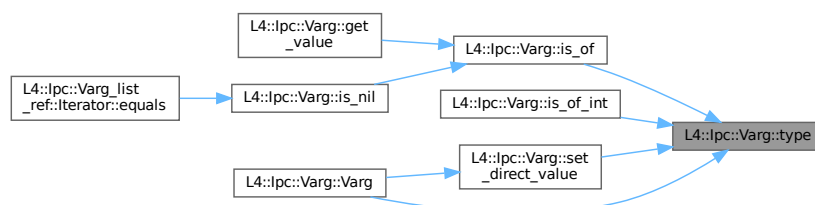
Returns

the type field of the tag

Definition at line 109 of file [ipc_varg](#).

Referenced by [is_of\(\)](#), [is_of_int\(\)](#), [set_direct_value\(\)](#), and [Varg\(\)](#).

Here is the caller graph for this function:



16.154.2.9 value()

```
template<typename V>
Va_type< V >::Ret_value L4::Ipc::Varg::value () const [inline]
```

Template Parameters

V	The data type of the value to retrieve.
---	---

Precondition

The [Varg](#) must be of type *V* (otherwise the result is unpredictable).

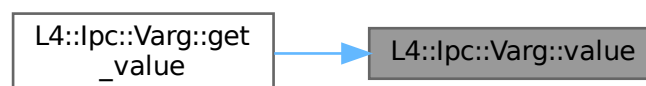
Returns

The value of the [Varg](#) as type *V*.

Definition at line [155](#) of file [ipc_varg](#).

Referenced by [get_value\(\)](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

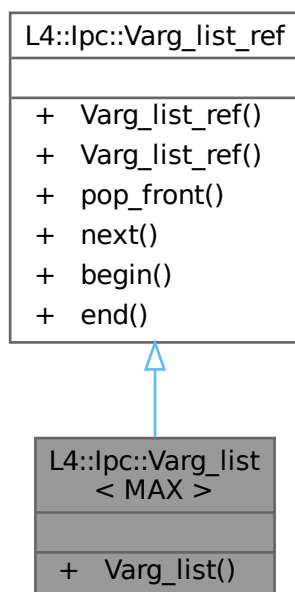
- `l4/sys/cxx/ipc_varg`

16.155 L4::lpc::Varg_list< MAX > Class Template Reference

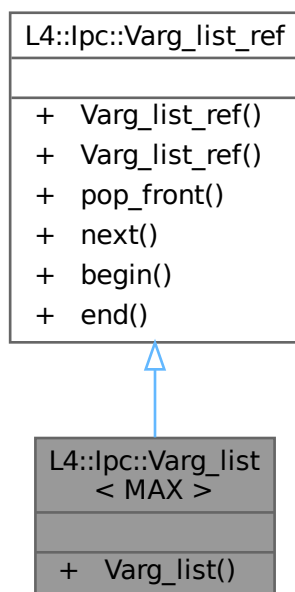
Self-contained list of variable-sized RPC parameters.

```
#include <ipc_varg>
```

Inheritance diagram for L4::lpc::Varg_list< MAX >:



Collaboration diagram for L4::lpc::Varg_list< MAX >:



Public Member Functions

- **Varg_list** ([Varg_list_ref](#) const &r)
Create a parameter list as a copy from a referencing list.

Public Member Functions inherited from [L4::lpc::Varg_list_ref](#)

- **Varg_list_ref** ()=default
Create an empty parameter list.
- [Varg_list_ref](#) (void const *start, void const *end)
Create a parameter list over a given memory region.
- [Varg](#) **pop_front** ()
Get the next parameter in the list.
- [Varg](#) **next** ()
Get the next parameter in the list.
- [Iterator](#) **begin** () const
Returns an iterator to the first [Varg](#).
- [Iterator](#) **end** () const
Returns the end of the list.

16.155.1 Detailed Description

```
template<unsigned MAX>
class L4::lpc::Varg_list< MAX >
```

Self-contained list of variable-sized RPC parameters.

Works like [Varg_list_ref](#) but contains a full copy of the data. Use this as a parameter in server functions, if the handler function needs to use the UTCB (e.g. while sending further IPC).

Definition at line 411 of file [ipc_varg](#).

The documentation for this class was generated from the following file:

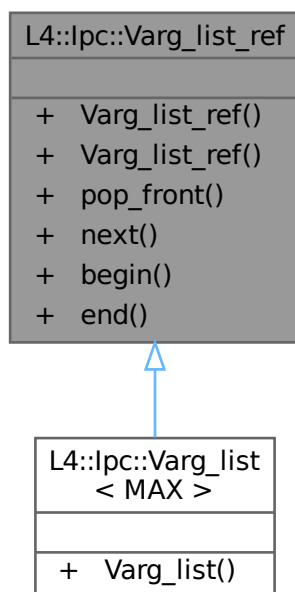
- I4/sys/cxx/ipc_varg

16.156 L4::lpc::Varg_list_ref Class Reference

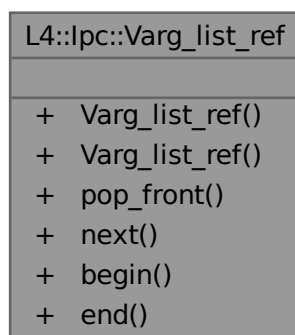
List of variable-sized RPC parameters as received by the server.

```
#include <ipc_varg>
```

Inheritance diagram for L4::lpc::Varg_list_ref:



Collaboration diagram for L4::lpc::Varg_list_ref:



Data Structures

- class [Iterator](#)
Iterator for *Valists*.

Public Member Functions

- **Varg_list_ref** ()=default
Create an empty parameter list.
- **Varg_list_ref** (void const *start, void const *end)
Create a parameter list over a given memory region.
- **Varg pop_front** ()
Get the next parameter in the list.
- **Varg next** ()
Get the next parameter in the list.
- **Iterator begin** () const
Returns an iterator to the first [Varg](#).
- **Iterator end** () const
Returns the end of the list.

16.156.1 Detailed Description

List of variable-sized RPC parameters as received by the server.

The list can be traversed exactly once using [next\(\)](#).

This is a reference list, where the returned [Varg](#) point to data in the underlying storage, conventionally the UTCB. This type should only be used in server functions when the implementation can ensure that all content is read before the UTCB is reused (e.g. for IPC), otherwise use [Varg_list](#).

Definition at line 253 of file [ipc_varg](#).

16.156.2 Constructor & Destructor Documentation

16.156.2.1 Varg_list_ref()

```
L4::Ipc::Varg_list_ref::Varg_list_ref (
    void const * start,
    void const * end) [inline]
```

Create a parameter list over a given memory region.

Parameters

<i>start</i>	Pointer to start of the parameter list.
<i>end</i>	Pointer to end of the list (inclusive).

Definition at line 332 of file [ipc_varg](#).

References [end\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

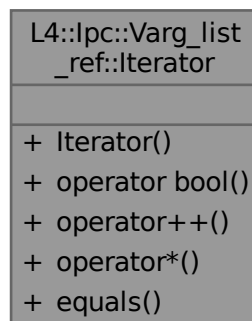
- l4/sys/cxx/ipc_varg

16.157 L4::ipc::Varg_list_ref::Iterator Class Reference

[Iterator](#) for Valists.

```
#include <ipc_varg>
```

Collaboration diagram for L4::ipc::Varg_list_ref::Iterator:



Public Member Functions

- **Iterator** (Iter_state const &s)
Create a new iterator.
- **operator bool** () const
validity check for the iterator
- **Iterator** & **operator++** ()
increment iterator to the next arg
- **Varg** **operator*** () const
dereference the iterator, get [Varg](#)
- **bool** **equals** ([Iterator](#) const &o) const
check for equality

16.157.1 Detailed Description

[Iterator](#) for Valists.

Definition at line 338 of file [ipc_varg](#).

The documentation for this class was generated from the following file:

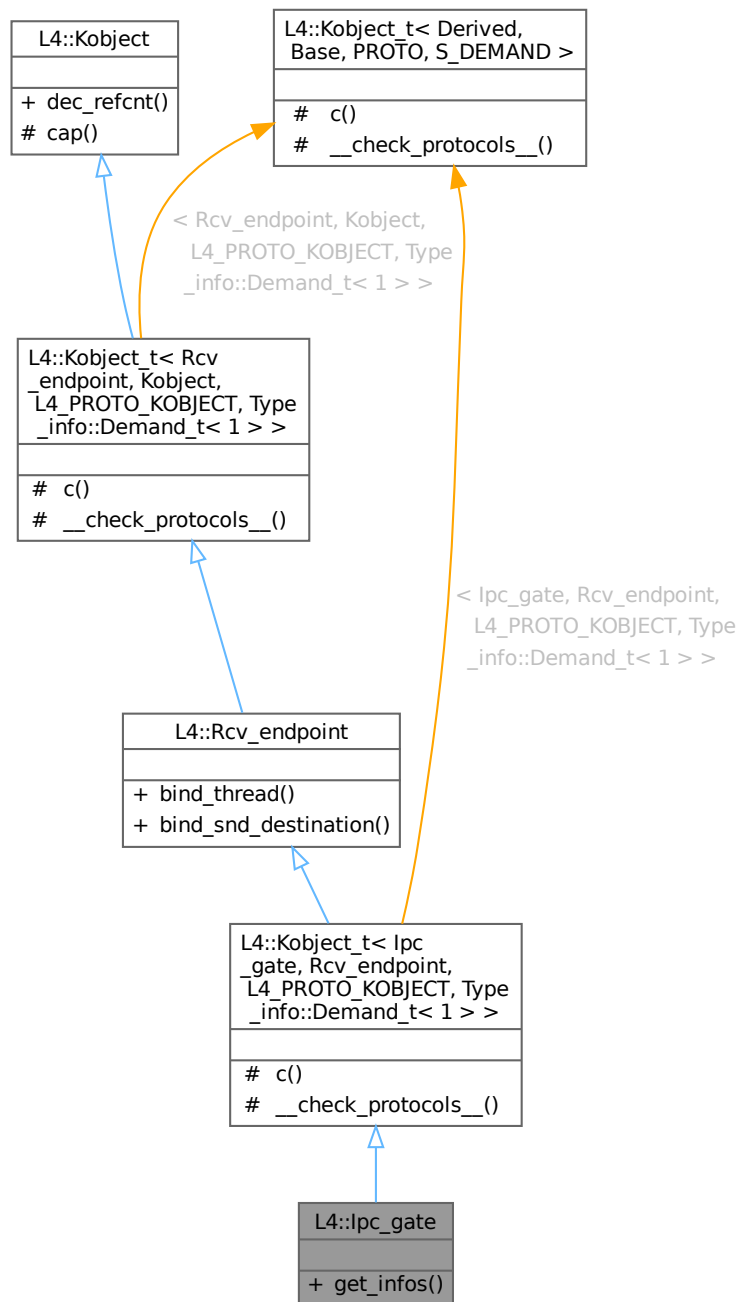
- l4/sys/cxx/ipc_varg

16.158 L4::ipc_gate Class Reference

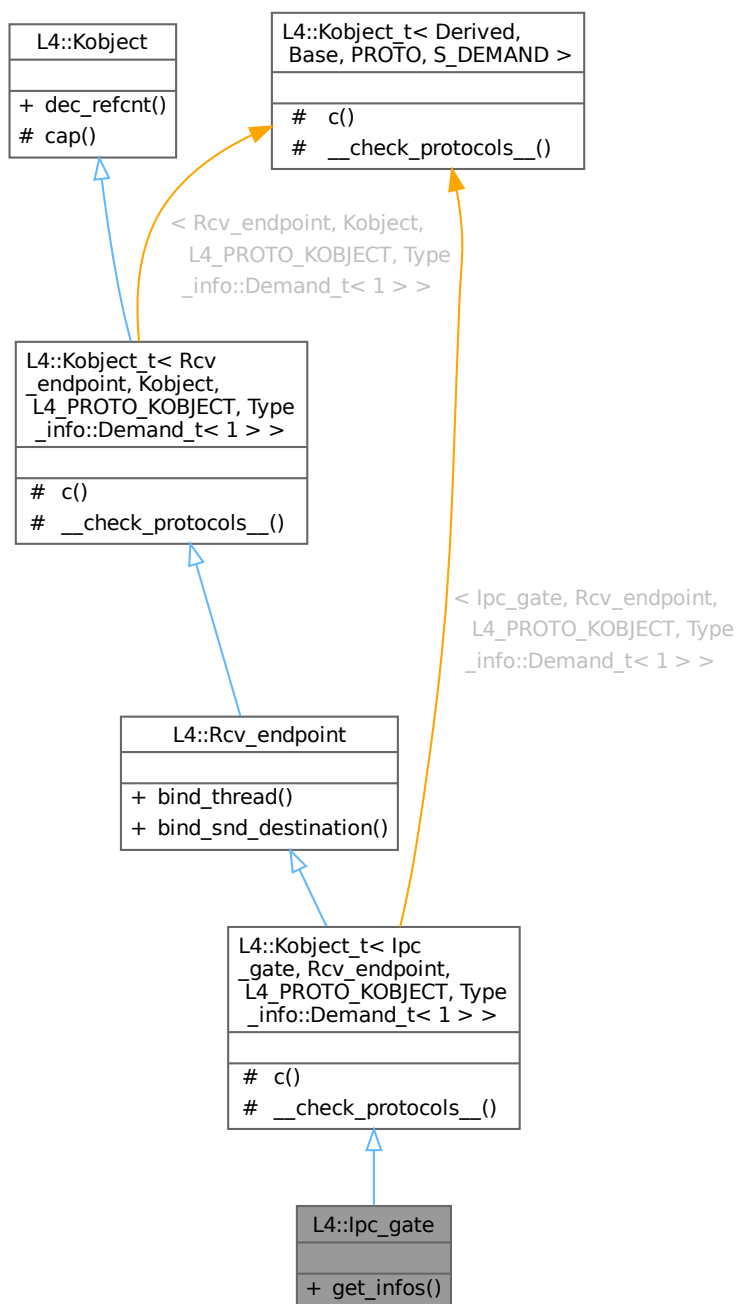
The C++ IPC gate interface, see [IPC-Gate API](#) for the C interface.

```
#include <ipc_gate>
```

Inheritance diagram for L4::lpc_gate:



Collaboration diagram for L4::lpc_gate:



Public Member Functions

- `l4_msgtag_t get_infos (l4_umword_t *label)`
Get information about the IPC-gate.

Public Member Functions inherited from L4::Rcv_endpoint

- `l4_msgtag_t bind_thread (lpc::Cap< Thread > t, l4_umword_t label)`

Bind the IPC receive endpoint to a thread.

- `l4_msgtag_t bind_snd_destination (Cap< Snd_destination > snd_dst, l4_umword_t label)`

Bind a send destination (a thread or thread group) to an IPC receive endpoint.

Public Member Functions inherited from `L4::Kobject`

- `l4_msgtag_t dec_refcnt (l4_mword_t diff, l4_utcb_t *utcb=l4_utcb())`

Decrement the in kernel reference counter for the object.

Additional Inherited Members

Protected Types inherited from

`L4::Kobject_t< lpc_gate, Rcv_endpoint, L4_PROTO_KOBJECT, Type_info::Demand_t< 1 > >`

- typedef `lpc_gate` **Class**

The target interface type (inheriting from `Kobject_t`).

- typedef `Typeid::Iface< PROTO, lpc_gate > __Iface`

The interface description for the derived class.

- typedef `Typeid::Merge_list< Typeid::Iface_list< __Iface >, typename Rcv_endpoint::__Iface_list > __Iface_list`

The list of all RPC interfaces provided directly or through inheritance.

Protected Types inherited from

`L4::Kobject_t< Rcv_endpoint, Kobject, L4_PROTO_KOBJECT, Type_info::Demand_t< 1 > >`

- typedef `Rcv_endpoint` **Class**

The target interface type (inheriting from `Kobject_t`).

- typedef `Typeid::Iface< PROTO, Rcv_endpoint > __Iface`

The interface description for the derived class.

- typedef `Typeid::Merge_list< Typeid::Iface_list< __Iface >, typename Kobject::__Iface_list > __Iface_list`

The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from

`L4::Kobject_t< lpc_gate, Rcv_endpoint, L4_PROTO_KOBJECT, Type_info::Demand_t< 1 > >`

- `L4::Cap< Class > c () const noexcept`

Get the capability to ourselves.

Protected Member Functions inherited from

`L4::Kobject_t< Rcv_endpoint, Kobject, L4_PROTO_KOBJECT, Type_info::Demand_t< 1 > >`

- `L4::Cap< Class > c () const noexcept`

Get the capability to ourselves.

Protected Member Functions inherited from L4::Kobject

- [l4_cap_idx_t cap](#) () const noexcept
Return capability selector.

Static Protected Member Functions inherited from

[L4::Kobject_t< lpc_gate, Rcv_endpoint, L4_PROTO_KOBJECT, Type_info::Demand_t< 1 > >](#)

- static void [__check_protocols__](#) () noexcept
Helper to check for protocol conflicts.

Static Protected Member Functions inherited from

[L4::Kobject_t< Rcv_endpoint, Kobject, L4_PROTO_KOBJECT, Type_info::Demand_t< 1 > >](#)

- static void [__check_protocols__](#) () noexcept
Helper to check for protocol conflicts.

16.158.1 Detailed Description

The C++ IPC gate interface, see [IPC-Gate API](#) for the C interface.

IPC gates are used to create secure communication channels between protection domains. An IPC gate can be created using the [L4::Factory](#) interface.

Depending on the permissions of the capability used, an IPC gate forwards IPC to the [L4::Thread](#) or [L4::Thread_group](#) the IPC gate is *bound* to (cf. [bind_thread\(\)](#) and [bind_snd_destination\(\)](#)). If the capability has the [L4_FPAGE_C_IPCGATE_SVR](#) permission, only IPC using a protocol different from the [L4_PROTO_KOBJECT](#) protocol is forwarded. Without the [L4_FPAGE_C_IPCGATE_SVR](#) permission, all IPC is forwarded. The latter is the usual case for a client in a client/server scenario. When not bound to a thread or thread group yet, the forwarded IPC blocks until the IPC gate is bound to a thread or thread group, or the IPC times out.

Forwarded IPC is always forwarded to the userland of the thread the IPC gate is bound to, either directly or indirectly using a thread group. That means, the [L4::Thread](#) interface of that thread is not accessible via an IPC gate. The [L4::lpc_gate](#) interface of an IPC gate is only accessible if the capability used has the [L4_FPAGE_C_IPCGATE_SVR](#) permission (cf. previous paragraph). Conversely that means, if the capability used lacks the [L4_FPAGE_C_IPCGATE_SVR](#) permission, [L4::lpc_gate](#) interface calls are forwarded to the thread or thread group the IPC gate is bound to instead of being processed by the IPC gate itself. In a client/server scenario, a client should only get IPC gate capabilities without [L4_FPAGE_C_IPCGATE_SVR](#) permission so the client cannot tamper with the IPC gate.

When binding an IPC gate to a thread or thread group, a user-defined, kernel protected, machine-word sized payload called the IPC gate's *label* is assigned to the IPC gate (note that the two least significant bits of the label must be zero; cf. [bind_thread\(\)](#) and [bind_snd_destination\(\)](#)). When a send-only IPC or call IPC is forwarded via an IPC gate, the label provided by the sender is ignored and replaced by the IPC gate's label where the two least significant bits are set to the [L4_CAP_FPAGE_S](#) and [L4_CAP_FPAGE_W](#) permissions of the capability used. The replaced label is only visible to the thread the IPC gate is bound to (or to the selected thread of the thread group the IPC gate is bound to) upon receive. However, the configured label of an IPC gate can also be queried via [get_infos\(\)](#) if the capability used has the [L4_FPAGE_C_IPCGATE_SVR](#) permission.

When deleting an IPC gate or when unbinding it from a thread or thread group, the label of IPC already in flight won't be changed. To ensure that no IPC from this IPC gate is received by a thread with an unexpected label, [L4::Thread::modify_senders\(\)](#) shall be used to change the labels of every pending IPC to that gate. This is also required if the label of an already bound IPC gate is changed. It is not necessary after binding the IPC gate to a thread or thread group for the first time.

When binding a currently bound IPC gate to a new thread or thread group, the same label should be used that was used with the old thread. Otherwise the old and the new thread need to synchronize to avoid IPC messages with unexpected labels.

Include File

```
#include <l4/sys/ipc_gate>
```

For the C interface refer to the C [IPC-Gate API](#).

See also

[Object Invocation](#)

Definition at line 85 of file [ipc_gate](#).

16.158.2 Member Function Documentation**16.158.2.1 get_infos()**

```
l4_msgtag_t L4::Ipc_gate::get_infos (
    l4_umword_t * label)
```

Get information about the IPC-gate.

Parameters

out	<i>label</i>	The label of the IPC gate is returned here.
-----	--------------	---

Returns

System call return tag.

Precondition

If the IPC gate capability used to invoke this operation does not possess the [L4_FPAGE_C_IPCGATE_SVR](#) right, the kernel will not perform the operation. Instead, the underlying IPC message will be forwarded to the thread or thread group the IPC gate is bound to, blocking the caller if the IPC gate is not bound yet.

References [get_infos\(\)](#).

Referenced by [get_infos\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

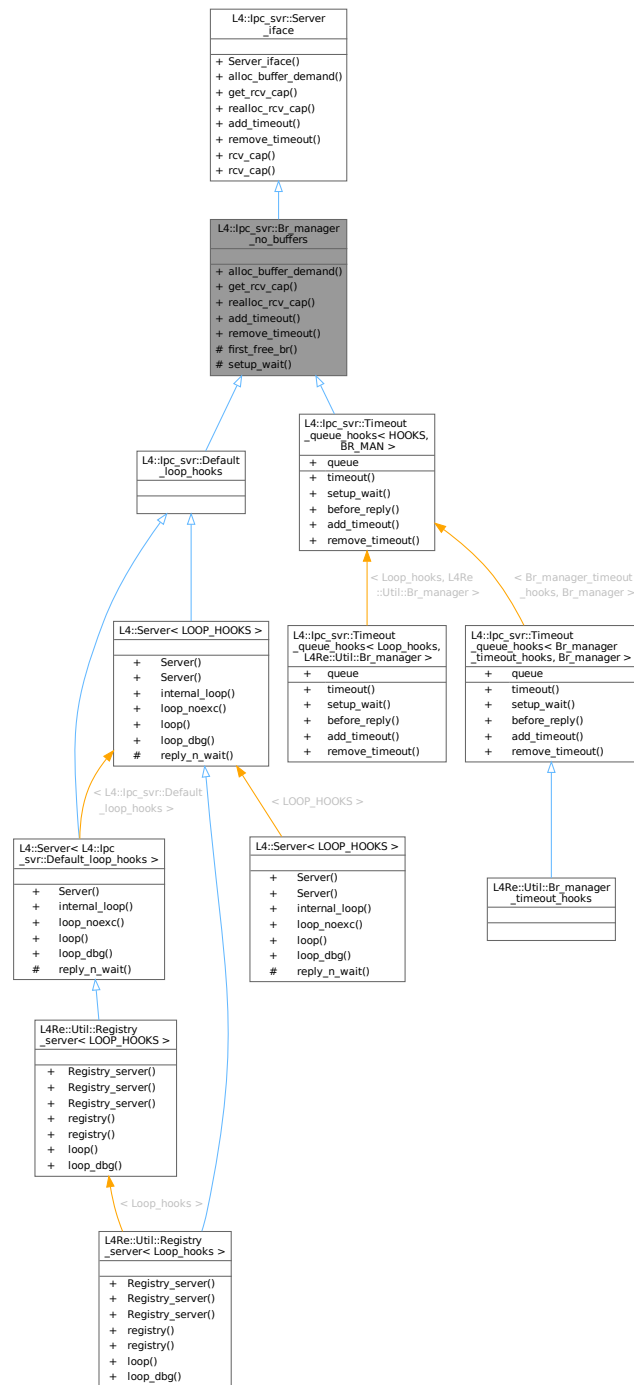
- [l4/sys/ipc_gate](#)

16.159 L4::lpc_svr::Br_manager_no_buffers Class Reference

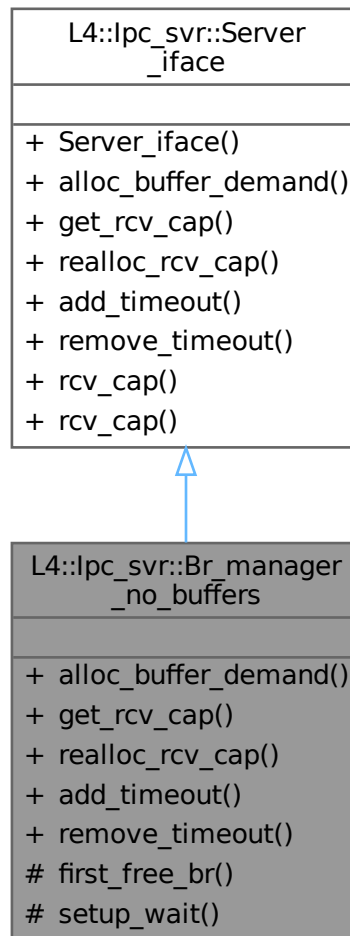
Empty implementation of [Server_iface](#).

```
#include <ipc_server_loop>
```

Inheritance diagram for L4::lpc_svr::Br_manager_no_buffers:



Collaboration diagram for L4::lpc_svr::Br_manager_no_buffers:



Public Member Functions

- int `alloc_buffer_demand` (`Demand` const &demand) override
Tells the server to allocate buffers for the given demand.
- L4::Cap< void > `get_rcv_cap` (int) const override
Returns L4::Cap<void>::Invalid, we have no buffer management.
- int `realloc_rcv_cap` (int) override
Returns -L4_ENOMEM, we have no buffer management.
- int `add_timeout` (Timeout *, l4_kernel_clock_t) override
Returns -L4_ENOSYS, we have no timeout queue.
- int `remove_timeout` (Timeout *) override
Returns -L4_ENOSYS, we have no timeout queue.

Public Member Functions inherited from [L4::lpc_svr::Server_iface](#)

- **Server_iface** ()
Make a server interface.
- `template<typename T>`
[L4::Cap](#)< T > [rcv_cap](#) (int index) const
Get given receive buffer as typed capability.
- [L4::Cap](#)< void > [rcv_cap](#) (int index) const
Get receive cap with the given index as generic (void) type.

Protected Member Functions

- unsigned **first_free_br** () const
Returns 1 as first free buffer.
- void **setup_wait** ([l4_utcb_t](#) *utcb, [L4::lpc_svr::Reply_mode](#))
Setup wait function for the server loop ([Server](#)<>).

Additional Inherited Members

Public Types inherited from [L4::lpc_svr::Server_iface](#)

- typedef [L4::Type_info::Demand](#) **Demand**
Data type expressing server-side demand for receive buffers.

16.159.1 Detailed Description

Empty implementation of [Server_iface](#).

This implementation of [Server_iface](#) provides no buffer or timeout management at all it just returns errors for all calls that express other than empty demands. However, this may be useful for very simple servers that serve simple server objects only.

Definition at line 233 of file [ipc_server_loop](#).

16.159.2 Member Function Documentation

16.159.2.1 [alloc_buffer_demand\(\)](#)

```
int L4::Ipc_svr::Br_manager_no_buffers::alloc_buffer_demand (
    Demand const & demand) [inline], [override], [virtual]
```

Tells the server to allocate buffers for the given demand.

Parameters

<i>demand</i>	The total server-side demand of receive buffers needed for a given interface, see Demand .
---------------	--

This function is not called by user applications directly. Usually the server implementation or the registry implementation calls this function whenever a new object is registered at the server.

Returns

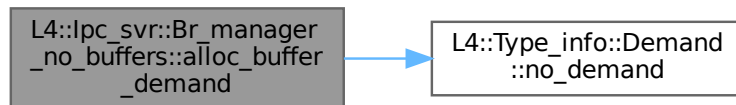
success (0) if demand is empty, -L4_ENOMEM else.

Implements [L4::lpc_svr::Server_iface](#).

Definition at line 240 of file [ipc_server_loop](#).

References [L4_ENOMEM](#), [L4_EOK](#), and [L4::Type_info::Demand::no_demand\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

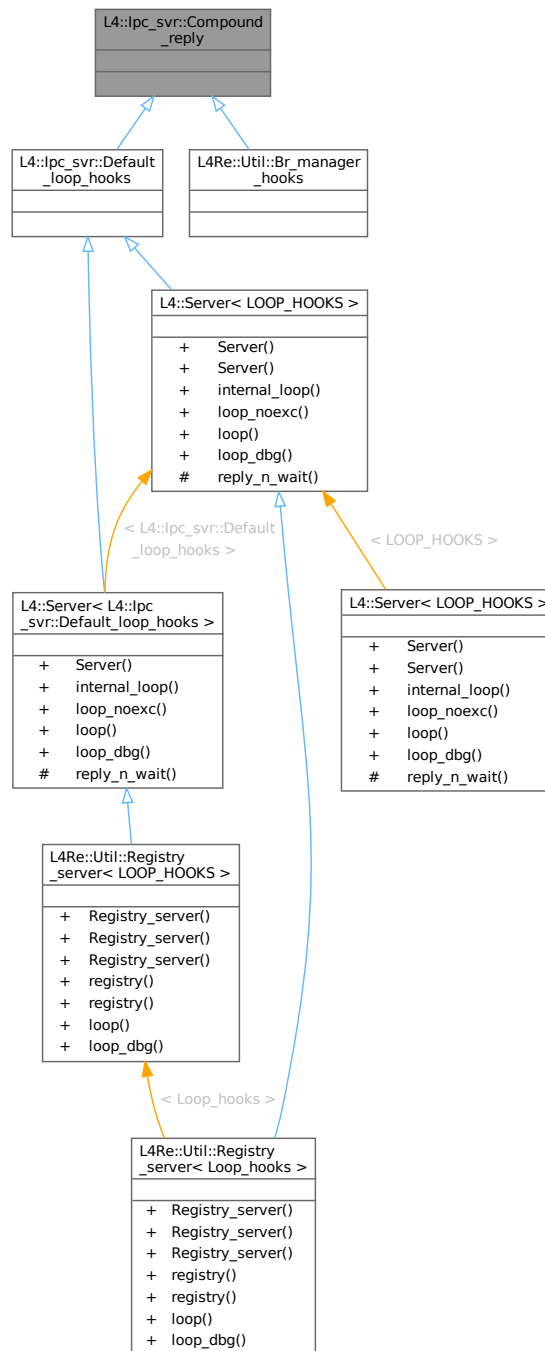
- `l4/sys/cxx/ipc_server_loop`

16.160 L4::lpc_svr::Compound_reply Struct Reference

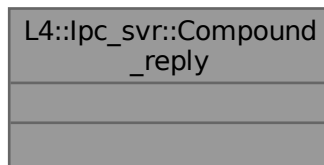
Mix in for LOOP_HOOKS to always use compound reply and wait.

```
#include <ipc_server_loop>
```

Inheritance diagram for L4::lpc_svr::Compound_reply:



Collaboration diagram for L4::lpc_svr::Compound_reply:



16.160.1 Detailed Description

Mix in for LOOP_HOOKS to always use compound reply and wait.

Definition at line 73 of file [ipc_server_loop](#).

The documentation for this struct was generated from the following file:

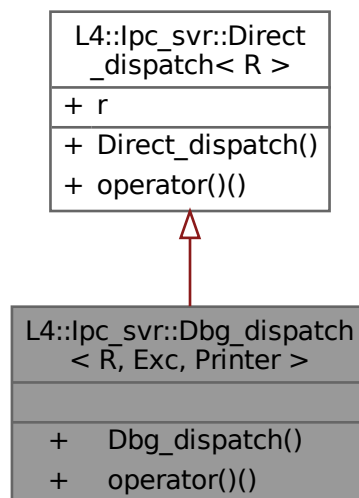
- l4/sys/cxx/ipc_server_loop

16.161 L4::lpc_svr::Dbg_dispatch< R, Exc, Printer > Struct Template Reference

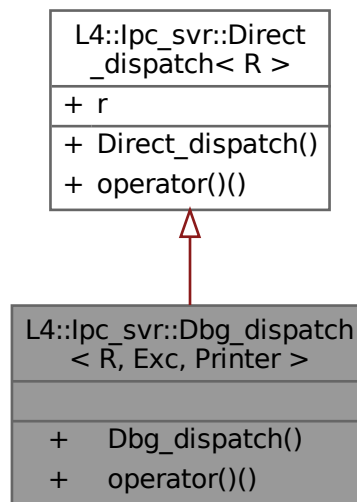
Dispatch helper that, in addition to what [Exc_dispatch](#) does, prints exception messages.

```
#include <ipc_server_loop>
```

Inheritance diagram for L4::lpc_svr::Dbg_dispatch< R, Exc, Printer >:



Collaboration diagram for `L4::lpc_svr::Dbg_dispatch< R, Exc, Printer >`:



Public Member Functions

- **Dbg_dispatch** (`R r`, `Printer p`)
Make an exception handling dispatcher.
- `l4_msgtag_t` **operator()** (`l4_msgtag_t tag`, `l4_umword_t obj`, `l4_utcb_t *utcb`)
Dispatch the call via [Direct_dispatch<R>\(\)](#), handle exceptions, and print the exception message.

16.161.1 Detailed Description

```
template<typename R, typename Exc, typename Printer>
struct L4::lpc_svr::Dbg_dispatch< R, Exc, Printer >
```

Dispatch helper that, in addition to what [Exc_dispatch](#) does, prints exception messages.

Template Parameters

<i>R</i>	Data type of the registry used for dispatching to objects.
<i>Exc</i>	Data type of the exceptions that shall be caught. This data type must provide a member <code>err_no()</code> that returns the negative integer (int) error code for the exception. In addition, methods <code>str()</code> and <code>extra_str()</code> are required that return a c-string describing the error.
<i>Printer</i>	A type that provides a <code>printf()</code> member that is used (with the usual format string syntax) to print error messages.

This dispatcher wraps `Direct_dispatch<R>` with a try-catch (`Exc`).

Definition at line 184 of file [ipc_server_loop](#).

The documentation for this struct was generated from the following file:

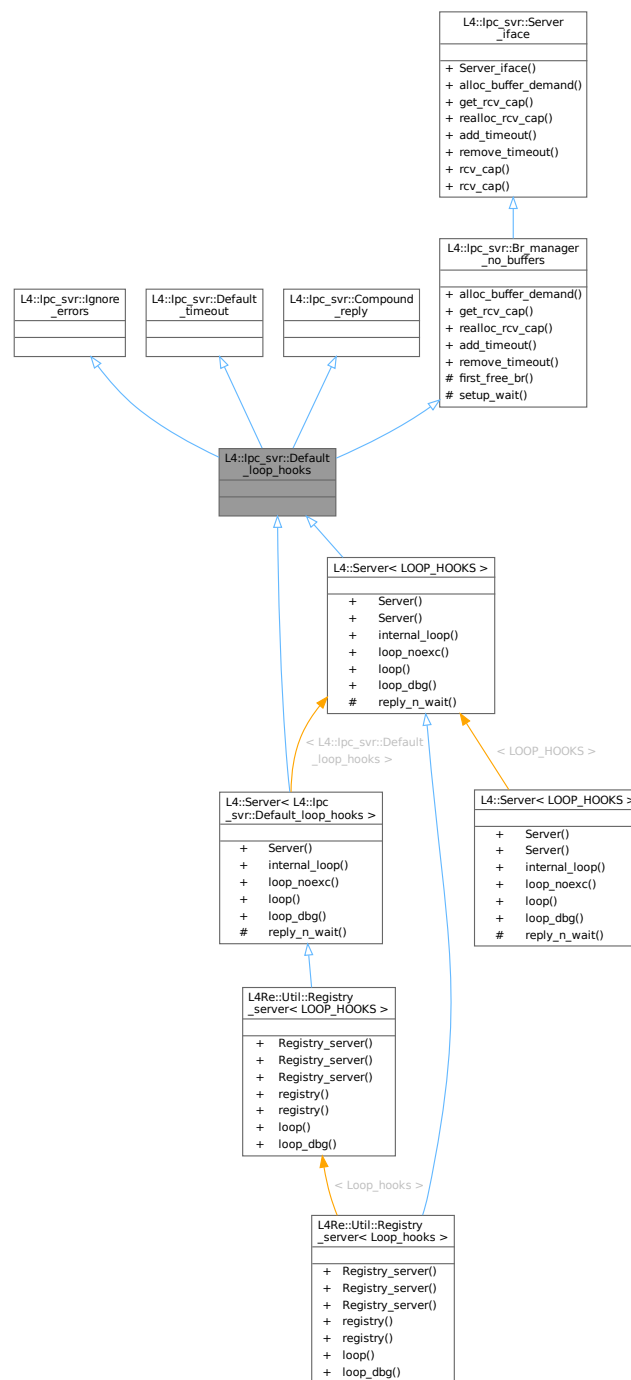
- `l4/sys/cxx/ipc_server_loop`

16.162 L4::lpc_svr::Default_loop_hooks Struct Reference

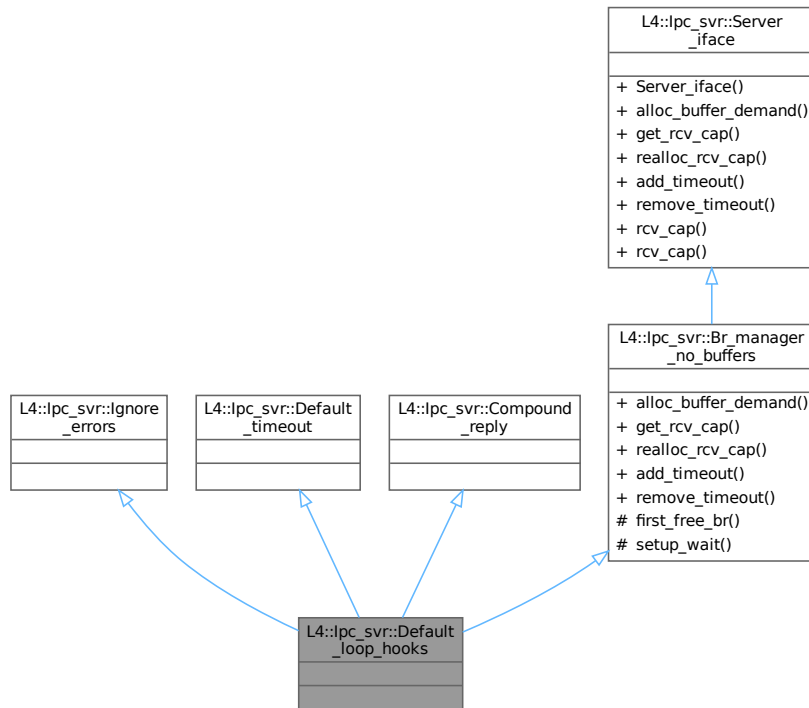
Default LOOP_HOOKS.

```
#include <ipc_server_loop>
```

Inheritance diagram for L4::lpc_svr::Default_loop_hooks:



Collaboration diagram for L4::lpc_svr::Default_loop_hooks:



Additional Inherited Members

Public Types inherited from L4::lpc_svr::Server_iface

- typedef L4::Type_info::Demand Demand
Data type expressing server-side demand for receive buffers.

Public Member Functions inherited from L4::lpc_svr::Br_manager_no_buffers

- int **alloc_buffer_demand** (Demand const &demand) override
Tells the server to allocate buffers for the given demand.
- L4::Cap< void > **get_rcv_cap** (int) const override
Returns L4::Cap< void >::Invalid, we have no buffer management.
- int **realloc_rcv_cap** (int) override
Returns -L4_ENOMEM, we have no buffer management.
- int **add_timeout** (Timeout *, l4_kernel_clock_t) override
Returns -L4_ENOSYS, we have no timeout queue.
- int **remove_timeout** (Timeout *) override
Returns -L4_ENOSYS, we have no timeout queue.

Public Member Functions inherited from L4::lpc_svr::Server_iface

- **Server_iface** ()
Make a server interface.
- template<typename T>
L4::Cap< T > **rcv_cap** (int index) const
Get given receive buffer as typed capability.
- L4::Cap< void > **rcv_cap** (int index) const
Get receive cap with the given index as generic (void) type.

Protected Member Functions inherited from L4::lpc_svr::Br_manager_no_buffers

- unsigned **first_free_br** () const
Returns 1 as first free buffer.
- void **setup_wait** (l4_utcb_t *utcb, L4::lpc_svr::Reply_mode)
Setup wait function for the server loop (Server<>).

16.162.1 Detailed Description

Default LOOP_HOOKS.

Combination of [Ignore_errors](#), [Default_timeout](#), [Compound_reply](#), and [Br_manager_no_buffers](#).

Definition at line 285 of file [ipc_server_loop](#).

The documentation for this struct was generated from the following file:

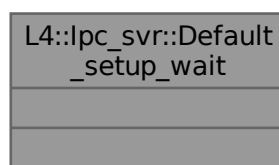
- l4/sys/cxx/ipc_server_loop

16.163 L4::lpc_svr::Default_setup_wait Struct Reference

Mix in for LOOP_HOOKS for setup_wait no op.

```
#include <ipc_server_loop>
```

Collaboration diagram for L4::lpc_svr::Default_setup_wait:



16.163.1 Detailed Description

Mix in for LOOP_HOOKS for setup_wait no op.

Definition at line 84 of file [ipc_server_loop](#).

The documentation for this struct was generated from the following file:

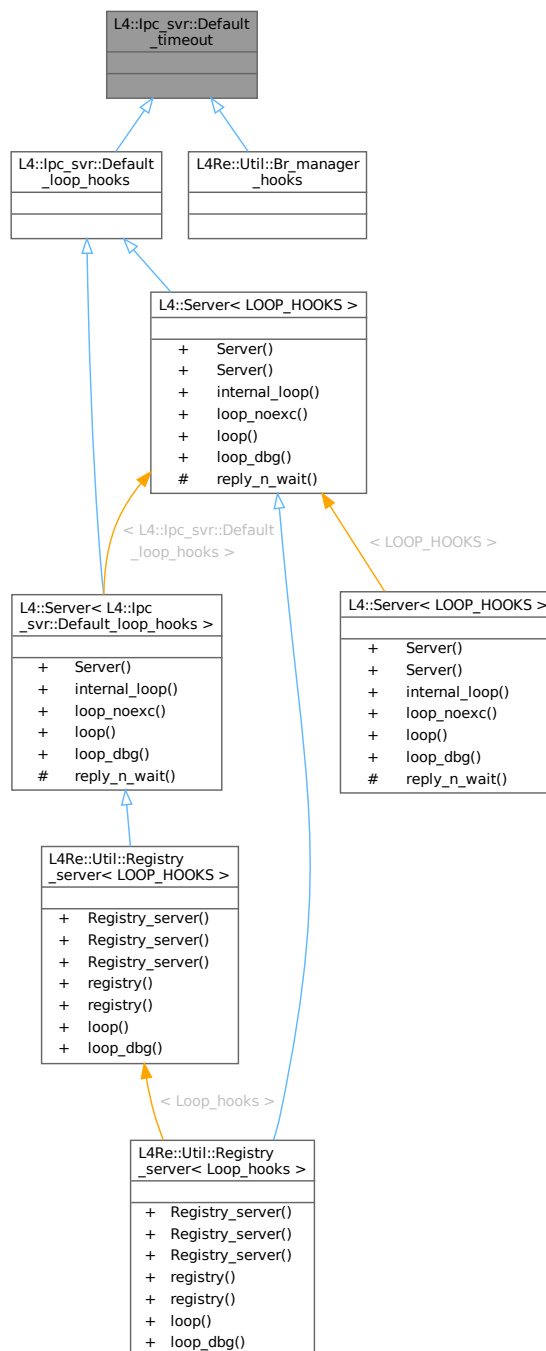
- l4/sys/cxx/ipc_server_loop

16.164 L4::lpc_svr::Default_timeout Struct Reference

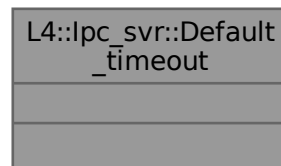
Mix in for LOOP_HOOKS to use a 0 send and a infinite receive timeout.

```
#include <ipc_server_loop>
```


Inheritance diagram for L4::lpc_svr::Default_timeout:



Collaboration diagram for L4::lpc_svr::Default_timeout:



16.164.1 Detailed Description

Mix in for LOOP_HOOKS to use a 0 send and a infinite receive timeout.

Definition at line 65 of file [ipc_server_loop](#).

The documentation for this struct was generated from the following file:

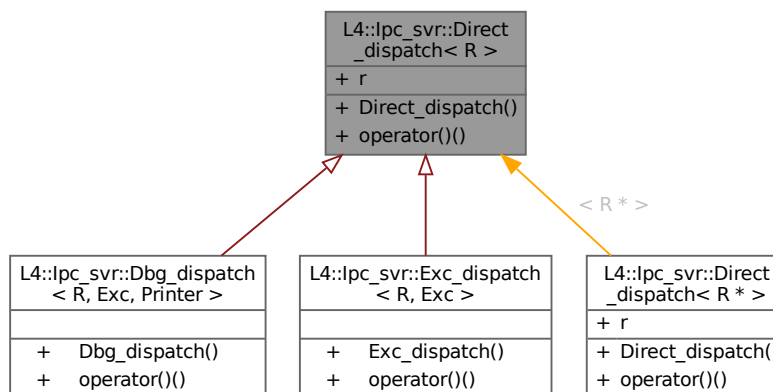
- l4/sys/cxx/ipc_server_loop

16.165 L4::lpc_svr::Direct_dispatch< R > Struct Template Reference

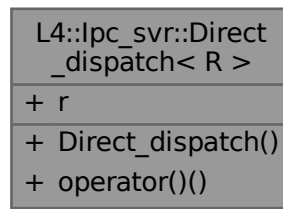
Direct dispatch helper, for forwarding dispatch calls to a registry *R*.

```
#include <ipc_server_loop>
```

Inheritance diagram for L4::lpc_svr::Direct_dispatch< R >:



Collaboration diagram for L4::lpc_svr::Direct_dispatch< R >:



Public Member Functions

- **Direct_dispatch** (R &r)
Make a direct dispatcher.
- **l4_msgtag_t operator()** (l4_msgtag_t tag, l4_umword_t obj, l4_utcb_t *utcb)
call operator forwarding to r.dispatch()

Data Fields

- **R & r**
stores a reference to the registry object

16.165.1 Detailed Description

```
template<typename R>
struct L4::lpc_svr::Direct_dispatch< R >
```

Direct dispatch helper, for forwarding dispatch calls to a registry *R*.

Template Parameters

<i>R</i>	Data type of the registry that is used for dispatching to different server objects, usually based on the protected IPC label.
----------	---

Definition at line 95 of file [ipc_server_loop](#).

The documentation for this struct was generated from the following file:

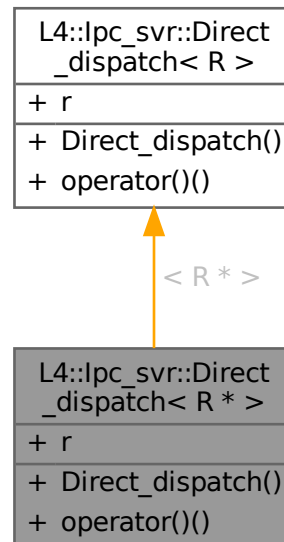
- l4/sys/cxx/ipc_server_loop

16.166 L4::lpc_svr::Direct_dispatch< R * > Struct Template Reference

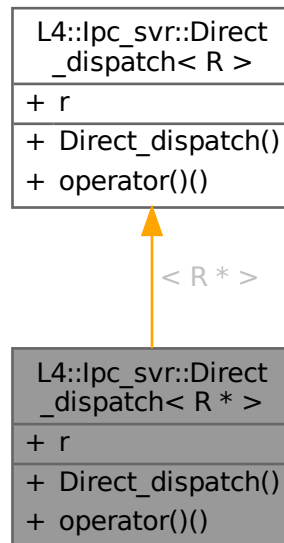
Direct dispatch helper, for forwarding dispatch calls via a pointer to a registry *R*.

```
#include <ipc_server_loop>
```

Inheritance diagram for L4::lpc_svr::Direct_dispatch< R * >:



Collaboration diagram for L4::lpc_svr::Direct_dispatch< R * >:



Public Member Functions

- **Direct_dispatch** (R *r)
Make a direct dispatcher.
- **l4_msgtag_t operator()** (l4_msgtag_t tag, l4_umword_t obj, l4_utcb_t *utcb)
call operator forwarding to r->dispatch()

Data Fields

- **R * r**
stores a pointer to the registry object

16.166.1 Detailed Description

```
template<typename R>
struct L4::lpc_svr::Direct_dispatch< R * >
```

Direct dispatch helper, for forwarding dispatch calls via a pointer to a registry *R*.

Template Parameters

<i>R</i>	Data type of the registry that is used for dispatching to different server objects, usually based on the protected IPC label.
----------	---

Definition at line 116 of file [ipc_server_loop](#).

The documentation for this struct was generated from the following file:

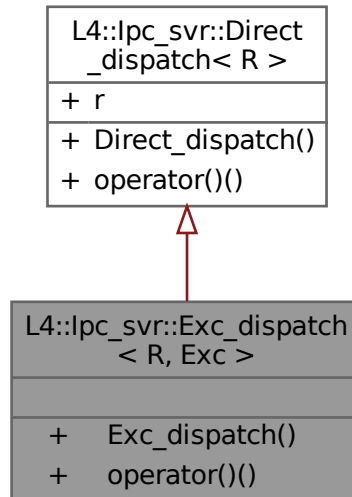
- l4/sys/cxx/ipc_server_loop

16.167 L4::lpc_svr::Exc_dispatch< R, Exc > Struct Template Reference

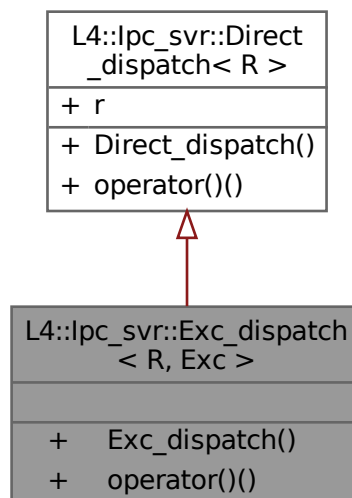
Dispatch helper wrapping try {} catch {} around the dispatch call.

```
#include <ipc_server_loop>
```

Inheritance diagram for L4::lpc_svr::Exc_dispatch< R, Exc >:



Collaboration diagram for L4::lpc_svr::Exc_dispatch< R, Exc >:



Public Member Functions

- **Exc_dispatch** (R r)
Make an exception handling dispatcher.
- **l4_msgtag_t operator()** (l4_msgtag_t tag, l4_umword_t obj, l4_utcb_t *utcb)
Dispatch the call via [Direct_dispatch<R>\(\)](#) and handle exceptions.

16.167.1 Detailed Description

```
template<typename R, typename Exc>
struct L4::lpc_svr::Exc_dispatch< R, Exc >
```

Dispatch helper wrapping try {} catch {} around the dispatch call.

Template Parameters

<i>R</i>	Data type of the registry used for dispatching to objects.
<i>Exc</i>	Data type of the exceptions that shall be caught. This data type must provide a member err_no() that returns the negative integer (int) error code for the exception.

This dispatcher wraps [Direct_dispatch<R>](#) with a try-catch (Exc).

Definition at line 140 of file [ipc_server_loop](#).

The documentation for this struct was generated from the following file:

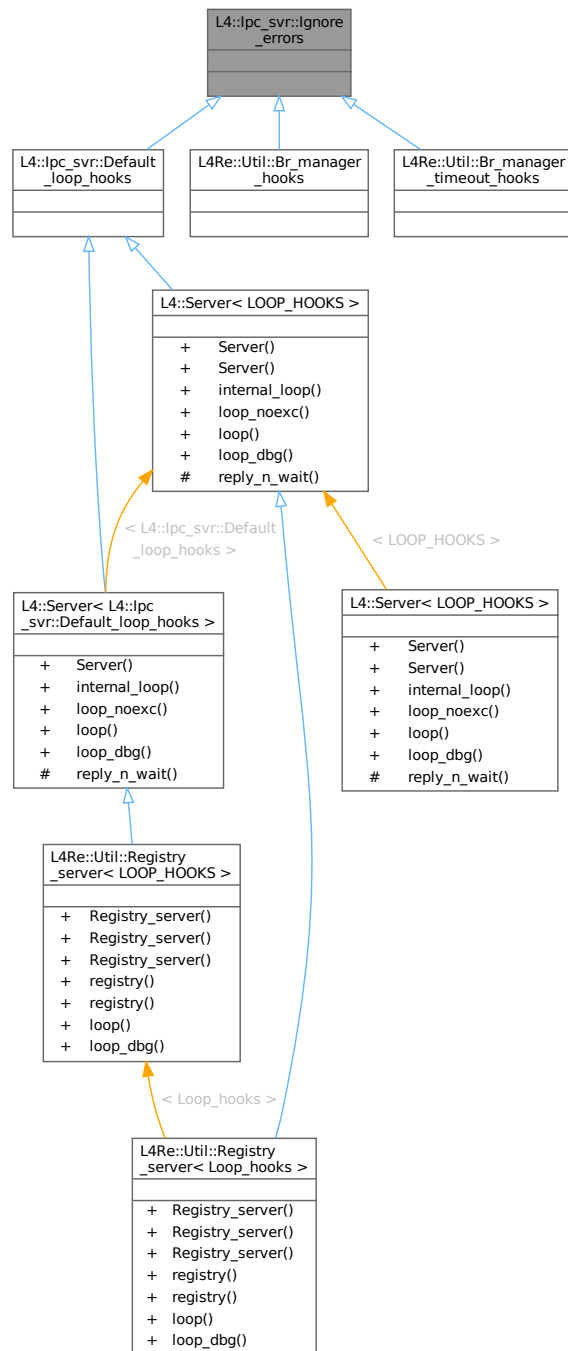
- l4/sys/cxx/ipc_server_loop

16.168 L4::lpc_svr::Ignore_errors Struct Reference

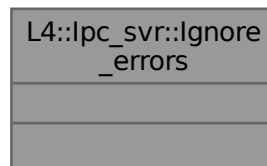
Mix in for LOOP_HOOKS to ignore IPC errors.

```
#include <ipc_server_loop>
```

Inheritance diagram for L4::ipc_svr::Ignore_errors:



Collaboration diagram for L4::lpc_svr::Ignore_errors:



16.168.1 Detailed Description

Mix in for LOOP_HOOKS to ignore IPC errors.

Definition at line 57 of file [ipc_server_loop](#).

The documentation for this struct was generated from the following file:

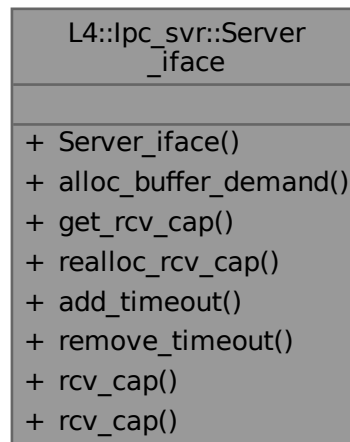
- `I4/sys/cxx/ipc_server_loop`

16.169 L4::lpc_svr::Server_iface Class Reference

Interface for server-loop related functions.

```
#include <ipc_epiface>
```


Collaboration diagram for L4::lpc_svr::Server_iface:



Public Types

- typedef [L4::Type_info::Demand](#) **Demand**
Data type expressing server-side demand for receive buffers.

Public Member Functions

- **Server_iface** ()
Make a server interface.
- virtual int [alloc_buffer_demand](#) ([Demand](#) const &demand)=0
Tells the server to allocate buffers for the given demand.
- virtual [L4::Cap](#)< void > [get_rcv_cap](#) (int index) const =0
Get capability slot allocated to the given receive buffer.
- virtual int [realloc_rcv_cap](#) (int index)=0
Allocate a new capability for the given receive buffer.
- virtual int [add_timeout](#) ([Timeout](#) *timeout, [l4_kernel_clock_t](#) time)=0
Add a timeout to the server internal timeout queue.
- virtual int [remove_timeout](#) ([Timeout](#) *timeout)=0
Remove the given timeout from the timer queue.
- template<typename T>
[L4::Cap](#)< T > [rcv_cap](#) (int index) const
Get given receive buffer as typed capability.
- [L4::Cap](#)< void > [rcv_cap](#) (int index) const
Get receive cap with the given index as generic (void) type.

16.169.1 Detailed Description

Interface for server-loop related functions.

This interface provides access to high-level server-loop related functions, such as management of receive buffers and timeouts.

Definition at line 36 of file [ipc_epiface](#).

16.169.2 Member Function Documentation

16.169.2.1 add_timeout()

```
virtual int L4::Ipc_svr::Server_iface::add_timeout (
    Timeout * timeout,
    l4_kernel_clock_t time) [pure virtual]
```

Add a timeout to the server internal timeout queue.

Parameters

<i>timeout</i>	The timeout object to register.
<i>time</i>	The time (absolute) at which the timeout shall expire.

Precondition

timeout must not be in any queue.

Returns

0 on success, 1 if timeout is already expired, < 0 on error.

Implemented in [L4::Ipc_svr::Br_manager_no_buffers](#), [L4::Ipc_svr::Timeout_queue_hooks< HOOKS, BR_MAN >](#), [L4::Ipc_svr::Timeout_queue_hooks< Br_manager_timeout_hooks, Br_manager >](#), [L4::Ipc_svr::Timeout_queue_hooks< Loop_hook_timeout_hooks, Loop_hook_manager >](#) and [L4Re::Util::Br_manager](#).

16.169.2.2 alloc_buffer_demand()

```
virtual int L4::Ipc_svr::Server_iface::alloc_buffer_demand (
    Demand const & demand) [pure virtual]
```

Tells the server to allocate buffers for the given demand.

Parameters

<i>demand</i>	The total server-side demand of receive buffers needed for a given interface, see Demand .
---------------	--

This function is not called by user applications directly. Usually the server implementation or the registry implementation calls this function whenever a new object is registered at the server.

Implemented in [L4::Ipc_svr::Br_manager_no_buffers](#), and [L4Re::Util::Br_manager](#).

16.169.2.3 get_rcv_cap()

```
virtual L4::Cap< void > L4::Ipc_svr::Server_iface::get_rcv_cap (  
    int index) const [pure virtual]
```

Get capability slot allocated to the given receive buffer.

Parameters

<i>index</i>	The receive buffer index of the expected capability argument ($0 \leq \text{index} < \text{caps}$ registered with alloc_buffer_demand()).
--------------	---

Precondition

$0 \leq \text{index} < \text{caps}$ registered with [alloc_buffer_demand\(\)](#)

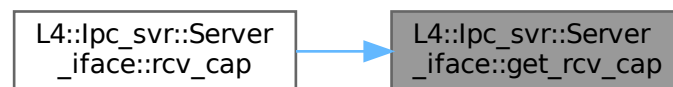
Returns

Capability slot currently allocated to the given receive buffer.

Implemented in [L4::lpc_svr::Br_manager_no_buffers](#), and [L4Re::Util::Br_manager](#).

Referenced by [rcv_cap\(\)](#).

Here is the caller graph for this function:

**16.169.2.4 rcv_cap() [1/2]**

```

L4::Cap< void > L4::lpc_svr::Server_iface::rcv_cap (
    int index) const [inline]
  
```

Get receive cap with the given index as generic (void) type.

Parameters

<i>index</i>	The index of the cap receive buffer of the expected capability. ($0 \leq \text{index} < \text{caps}$ registered with alloc_buffer_demand() .)
--------------	--

Returns

Capability slot currently allocated to the given capability buffer.

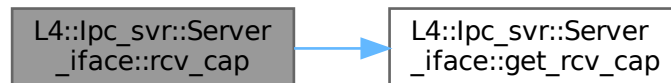
Note

This is a convenience wrapper for [get_rcv_cap\(\)](#).

Definition at line 126 of file [ipc_epiface](#).

References [get_rcv_cap\(\)](#).

Here is the call graph for this function:

**16.169.2.5 rcv_cap()** [2/2]

```
template<typename T>
L4::Cap< T > L4::lpc_svr::Server_iface::rcv_cap (
    int index) const [inline]
```

Get given receive buffer as typed capability.

See also

[get_rcv_cap\(\)](#)

Parameters

<i>index</i>	The receive buffer index of the expected capability argument. ($0 \leq \text{index} < \text{caps}$ registered with alloc_buffer_demand() .)
--------------	--

Precondition

$0 \leq \text{index} < \text{caps}$ registered with [alloc_buffer_demand\(\)](#)

Returns

Capability slot currently allocated to the given receive buffer.

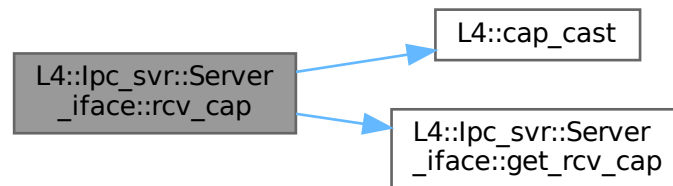
Note

This is a convenience wrapper for [get_rcv_cap\(\)](#) to avoid [L4::cap_cast<>\(\)](#).

Definition at line 114 of file [ipc_epiface](#).

References [L4::cap_cast\(\)](#), and [get_rcv_cap\(\)](#).

Here is the call graph for this function:

**16.169.2.6 realloc_rcv_cap()**

```
virtual int L4::Ipc_svr::Server_iface::realloc_rcv_cap (
    int index) [pure virtual]
```

Allocate a new capability for the given receive buffer.

Parameters

<i>index</i>	The receive buffer index of the expected capability argument ($0 \leq \text{index} < \text{caps}$ registered with alloc_buffer_demand()).
--------------	---

Precondition

$0 \leq \text{index} < \text{caps}$ registered with [alloc_buffer_demand\(\)](#)

Returns

0 on success, < 0 on error.

Implemented in [L4::lpc_svr::Br_manager_no_buffers](#), and [L4Re::Util::Br_manager](#).

16.169.2.7 remove_timeout()

```
virtual int L4::Ipc_svr::Server_iface::remove_timeout (
    Timeout * timeout) [pure virtual]
```

Remove the given timeout from the timer queue.

Parameters

<i>timeout</i>	The timeout object to remove.
----------------	-------------------------------

Returns

0 on success, < 0 on error.

Implemented in [L4::lpc_svr::Br_manager_no_buffers](#), [L4::lpc_svr::Timeout_queue_hooks< HOOKS, BR_MAN >](#), [L4::lpc_svr::Timeout_queue_hooks< Br_manager_timeout_hooks, Br_manager >](#), [L4::lpc_svr::Timeout_queue_hooks< Loop_hooks, Loop_manager >](#) and [L4Re::Util::Br_manager](#).

The documentation for this class was generated from the following file:

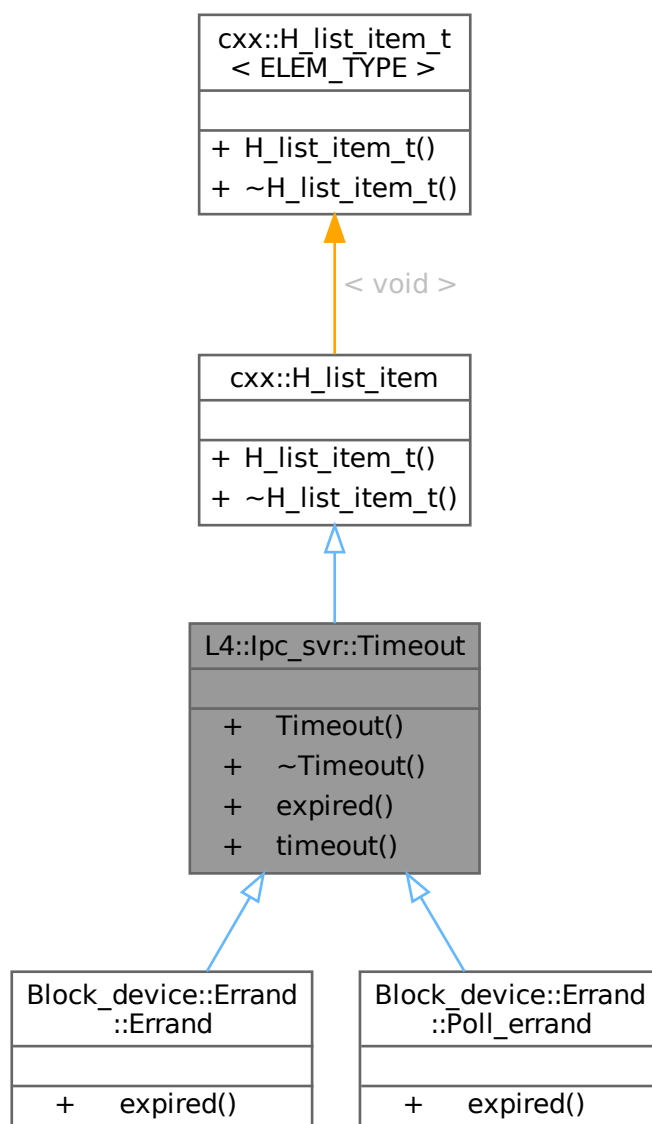
- [l4/sys/cxx/ipc_epiface](#)

16.170 L4::lpc_svr::Timeout Class Reference

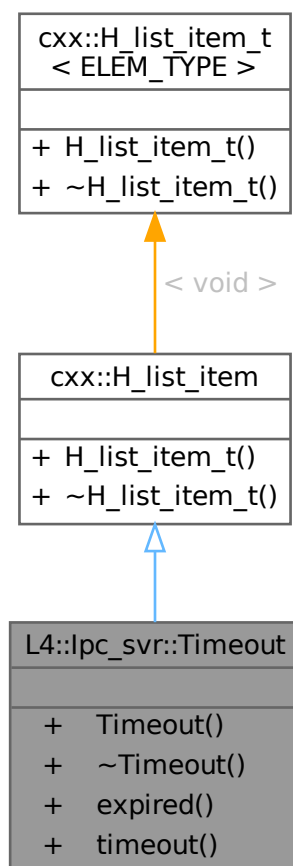
Callback interface for [Timeout_queue](#).

```
#include <ipc_timeout_queue>
```

Inheritance diagram for L4::lpc_svr::Timeout:



Collaboration diagram for L4::lpc_svr::Timeout:



Public Member Functions

- **Timeout ()**
Make a timeout.
- virtual **~Timeout ()=0**
Destroy a timeout.
- virtual void **expired ()=0**
callback function to be called when timeout happened
- **l4_kernel_clock_t timeout () const**
return absolute timeout of this callback.

Public Member Functions inherited from `cxx::H_list_item_t < void >`

- **H_list_item_t ()**
Constructor.
- **~H_list_item_t () noexcept**
Destructor.

16.170.1 Detailed Description

Callback interface for [Timeout_queue](#).

Definition at line 18 of file [ipc_timeout_queue](#).

16.170.2 Member Function Documentation

16.170.2.1 expired()

```
virtual void L4::Ipc_svr::Timeout::expired () [pure virtual]
```

callback function to be called when timeout happened

Note

The timeout object is already dequeued when this function is called, this means the timeout may be safely queued again within the [expired\(\)](#) function.

Implemented in [Block_device::Errand::Errand](#), and [Block_device::Errand::Poll_errand](#).

Referenced by [L4::ipc_svr::Timeout_queue::handle_expired_timeouts\(\)](#).

Here is the caller graph for this function:



16.170.2.2 timeout()

```
l4_kernel_clock_t L4::Ipc_svr::Timeout::timeout () const [inline]
```

return absolute timeout of this callback.

Returns

absolute timeout for this instance of the timeout.

Precondition

The timeout object must have been in a queue before, otherwise the timeout is not set.

Definition at line 42 of file [ipc_timeout_queue](#).

The documentation for this class was generated from the following file:

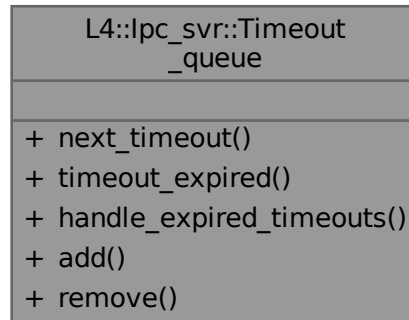
- [l4/cxx/ipc_timeout_queue](#)

16.171 L4::lpc_svr::Timeout_queue Class Reference

[Timeout](#) queue to be used in l4re server loop.

```
#include <ipc_timeout_queue>
```

Collaboration diagram for L4::lpc_svr::Timeout_queue:



Public Types

- typedef [L4::lpc_svr::Timeout](#) **Timeout**
Provide a local definition of [Timeout](#) for backward compatibility.

Public Member Functions

- [l4_kernel_clock_t](#) **next_timeout** () const
Get the time for the next timeout.
- bool **timeout_expired** ([l4_kernel_clock_t](#) now) const
Determine if a timeout has happened.
- void **handle_expired_timeouts** ([l4_kernel_clock_t](#) now)
run the callbacks of expired timeouts
- void **add** ([Timeout](#) *timeout, [l4_kernel_clock_t](#) time)
Add a timeout to the queue.
- void **remove** ([Timeout](#) *timeout)
Remove timeout from the queue.

16.171.1 Detailed Description

[Timeout](#) queue to be used in l4re server loop.

Definition at line 55 of file [ipc_timeout_queue](#).

16.171.2 Member Function Documentation

16.171.2.1 add()

```
void L4::Ipc_svr::Timeout_queue::add (
    Timeout * timeout,
    l4_kernel_clock_t time) [inline]
```

Add a timeout to the queue.

Parameters

<i>timeout</i>	timeout object to add
<i>time</i>	the time when the timeout expires

Precondition

timeout must not be in any queue already

Definition at line 111 of file [ipc_timeout_queue](#).

16.171.2.2 handle_expired_timeouts()

```
void L4::Ipc_svr::Timeout_queue::handle_expired_timeouts (
    l4_kernel_clock_t now) [inline]
```

run the callbacks of expired timeouts

Parameters

<i>now</i>	the current time.
------------	-------------------

Definition at line 91 of file [ipc_timeout_queue](#).

References [L4::Ipc_svr::Timeout::expired\(\)](#).

Here is the call graph for this function:



16.171.2.3 next_timeout()

```
l4_kernel_clock_t L4::Ipc_svr::Timeout_queue::next_timeout () const [inline]
```

Get the time for the next timeout.

Returns

the time for the next timeout or 0 if there is none

Definition at line 65 of file [ipc_timeout_queue](#).

Referenced by [timeout_expired\(\)](#).

Here is the caller graph for this function:

**16.171.2.4 remove()**

```
void L4::Ipc_svr::Timeout_queue::remove (
    Timeout * timeout) [inline]
```

Remove *timeout* from the queue.

Parameters

<i>timeout</i>	timeout to remove from timeout queue
----------------	--------------------------------------

Precondition

timeout must be in this queue

Definition at line 126 of file [ipc_timeout_queue](#).

16.171.2.5 timeout_expired()

```
bool L4::Ipc_svr::Timeout_queue::timeout_expired (
    l4_kernel_clock_t now) const [inline]
```

Determine if a timeout has happened.

Parameters

<i>now</i>	The current time.
------------	-------------------

Return values

<i>true</i>	There is at least one expired timeout in the queue. <i>false</i> No expired timeout in the queue.
-------------	---

Definition at line 81 of file [ipc_timeout_queue](#).

References [next_timeout\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

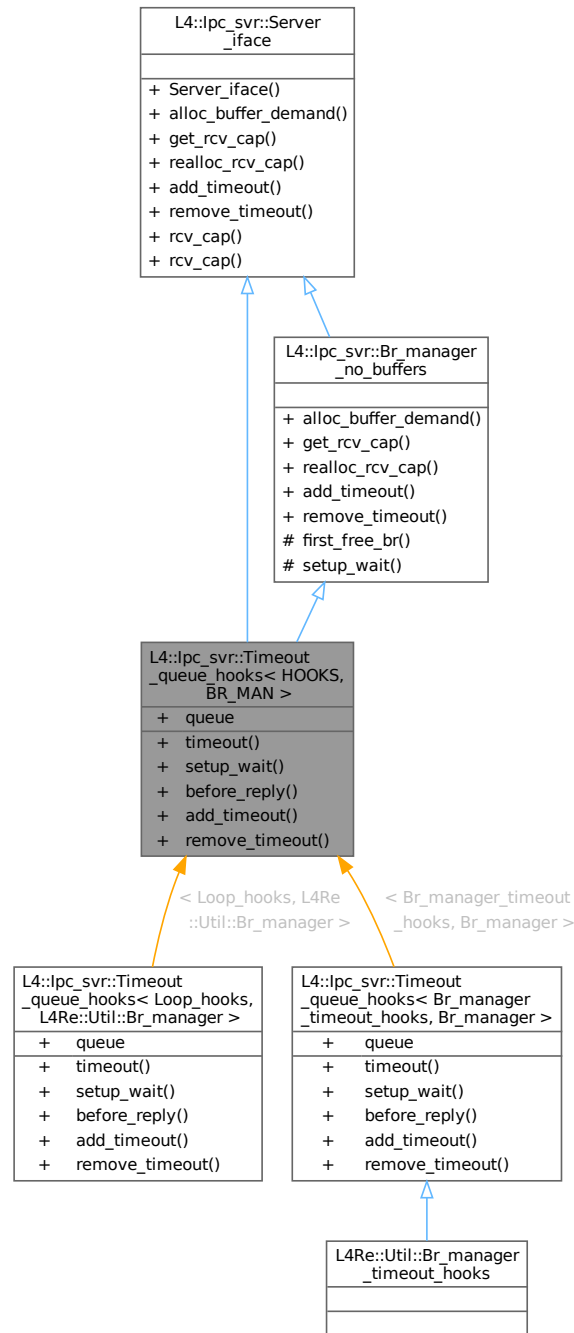
- [l4/cxx/ipc_timeout_queue](#)

16.172 L4::lpc_svr::Timeout_queue_hooks< HOOKS, BR_MAN > Class Template Reference

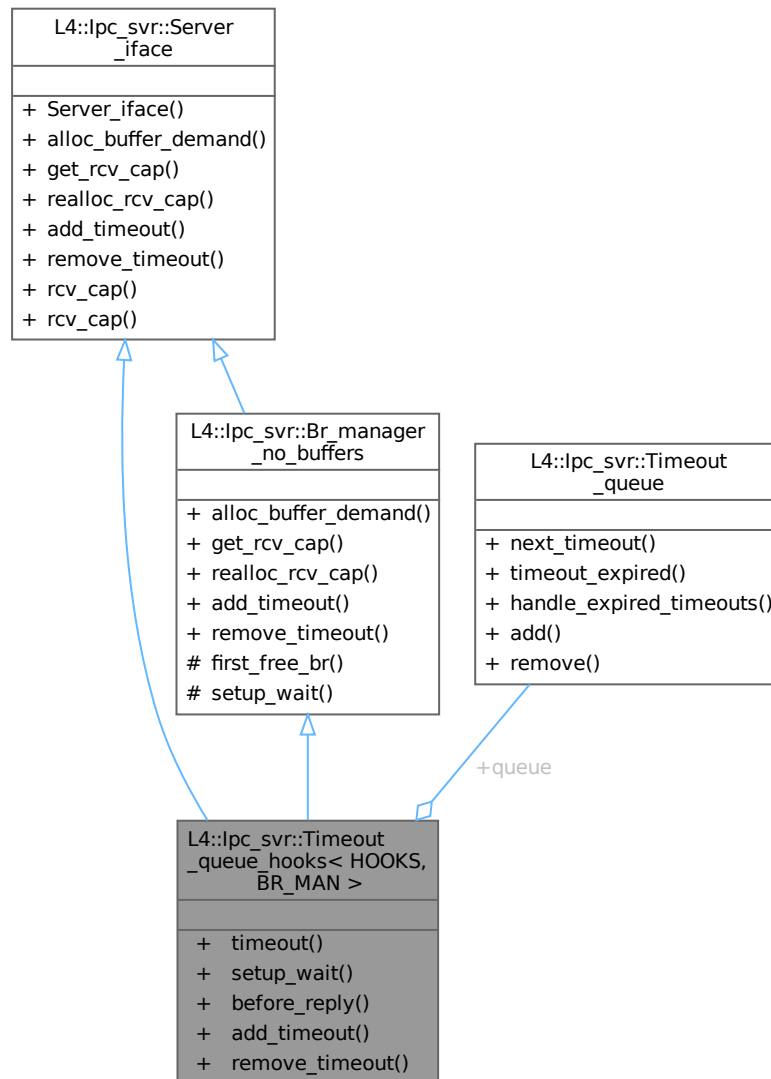
Loop hooks mixin for integrating a timeout queue into the server loop.

```
#include <ipc_timeout_queue>
```


Inheritance diagram for L4::lpc_svr::Timeout_queue_hooks< HOOKS, BR_MAN >:



Collaboration diagram for L4::lpc_svr::Timeout_queue_hooks< HOOKS, BR_MAN >:



Public Member Functions

- [l4_timeout_t timeout \(\)](#)
get the time for the next timeout
- void **setup_wait** ([l4_utcb_t](#) *utcb, [L4::lpc_svr::Reply_mode](#) mode)
setup_wait() for the server loop
- [L4::lpc_svr::Reply_mode](#) **before_reply** ([l4_msgtag_t](#), [l4_utcb_t](#) *)
server loop hook
- int **add_timeout** ([Timeout](#) *timeout, [l4_kernel_clock_t](#) time) override
Add a timeout to the queue for time time.
- int **remove_timeout** ([Timeout](#) *timeout) override
Remove timeout from the queue.

Public Member Functions inherited from L4::lpc_svr::Server_iface

- **Server_iface** ()
Make a server interface.
- template<typename T>
L4::Cap< T > **rcv_cap** (int index) const
Get given receive buffer as typed capability.
- L4::Cap< void > **rcv_cap** (int index) const
Get receive cap with the given index as generic (void) type.

Public Member Functions inherited from L4::lpc_svr::Br_manager_no_buffers

- int **alloc_buffer_demand** (Demand const &demand) override
Tells the server to allocate buffers for the given demand.
- L4::Cap< void > **get_rcv_cap** (int) const override
Returns L4::Cap< void> ::Invalid, we have no buffer management.
- int **realloc_rcv_cap** (int) override
Returns -L4_ENOMEM, we have no buffer management.

Data Fields

- **Timeout_queue** queue
Use this timeout queue.

Additional Inherited Members

Public Types inherited from L4::lpc_svr::Server_iface

- typedef L4::Type_info::Demand **Demand**
Data type expressing server-side demand for receive buffers.

Protected Member Functions inherited from L4::lpc_svr::Br_manager_no_buffers

- unsigned **first_free_br** () const
Returns 1 as first free buffer.
- void **setup_wait** (l4_utcb_t *utcb, L4::lpc_svr::Reply_mode)
Setup wait function for the server loop (Server<>).

16.172.1 Detailed Description

```
template<typename HOOKS, typename BR_MAN = Br_manager_no_buffers>
class L4::lpc_svr::Timeout_queue_hooks< HOOKS, BR_MAN >
```

Loop hooks mixin for integrating a timeout queue into the server loop.

Template Parameters

<i>HOOKS</i>	has to inherit from Timeout_queue_hooks<> and provide the functions <code>now()</code> that has to return the current time.
<i>BR_MAN</i>	This used as a base class for and provides the API for selecting the buffer register (BR) that is used to store the timeout value. This is usually L4Re::Util::Br_manager or L4::lpc_svr::Br_manager_no_buffers .

Definition at line 150 of file [ipc_timeout_queue](#).

16.172.2 Member Function Documentation

16.172.2.1 `add_timeout()`

```
template<typename HOOKS, typename BR_MAN = Br_manager_no_buffers>
int L4::lpc_svr::Timeout_queue_hooks< HOOKS, BR_MAN >::add_timeout (
    Timeout * timeout,
    l4_kernel_clock_t time) [inline], [override], [virtual]
```

Add a timeout to the queue for time *time*.

Parameters

<i>timeout</i>	The timeout object to add into the queue (must not be in any queue currently).
<i>time</i>	The time when the timeout shall expire.

Precondition

timeout must not be in any queue.

Note

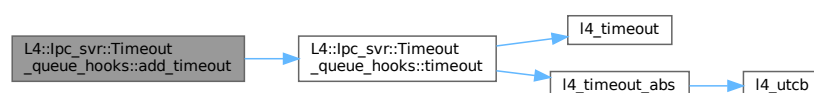
The timeout is automatically dequeued before the [Timeout::expired\(\)](#) function is called

Implements [L4::lpc_svr::Server_iface](#).

Definition at line 202 of file [ipc_timeout_queue](#).

References [queue](#), and [timeout\(\)](#).

Here is the call graph for this function:



16.172.2.2 remove_timeout()

```
template<typename HOOKS, typename BR_MAN = Br_manager_no_buffers>
int L4::lpc_svr::Timeout_queue_hooks< HOOKS, BR_MAN >::remove_timeout (
    Timeout * timeout) [inline], [override], [virtual]
```

Remove timeout from the queue.

Parameters

<i>timeout</i>	The timeout object to be removed from the queue.
----------------	--

Note

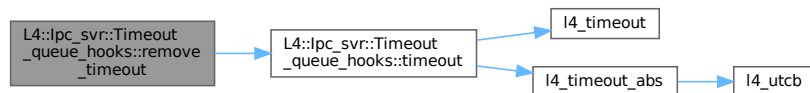
This function may be safely called even if the timeout is not currently enqueued.
 in [Timeout::expired\(\)](#) the timeout is already dequeued!

Implements [L4::ipc_svr::Server_iface](#).

Definition at line 215 of file [ipc_timeout_queue](#).

References [queue](#), and [timeout\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

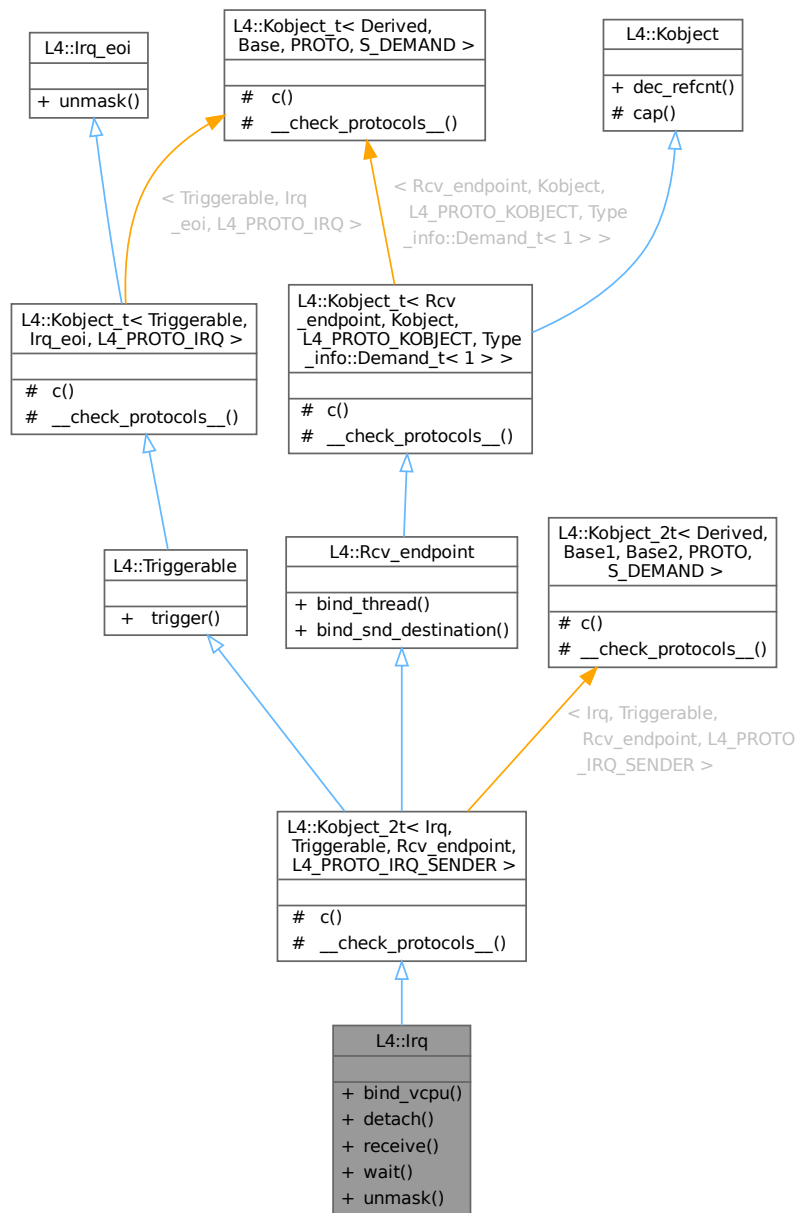
- `I4/cxx/ipc_timeout_queue`

16.173 L4::Irq Class Reference

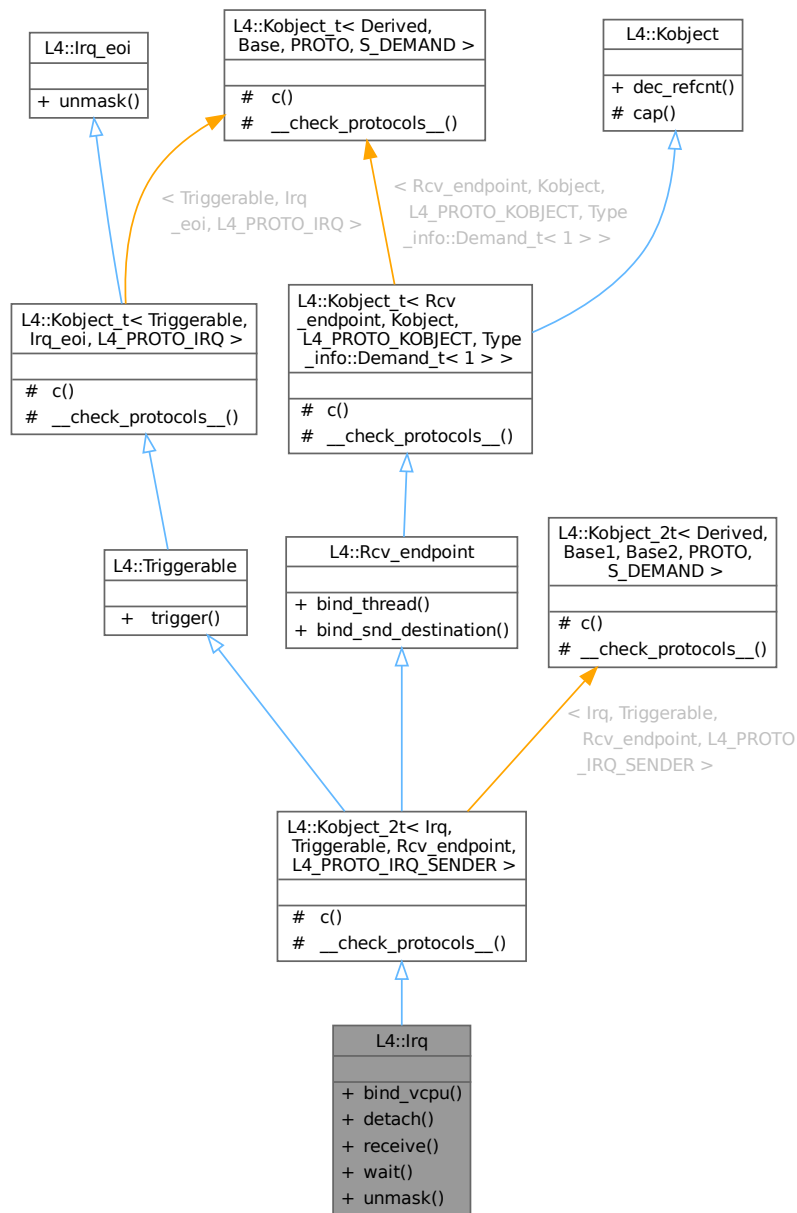
C++ [Irq](#) interface, see [IRQs](#) for the C interface.

```
#include <irq>
```

Inheritance diagram for L4::Irq:



Collaboration diagram for L4::Irq:



Public Member Functions

- **[l4_msgtag_t bind_vcpu](#)** (**[L4::Cap< Thread >](#)** const &thread, **[l4_umword_t](#)** cfg, **[l4_utcb_t](#)** *utcb=**[l4_utcb\(\)](#)**) noexcept
Bind a thread to this *Irq* for vCPU interrupt forwarding.
- **[l4_msgtag_t detach](#)** (**[l4_utcb_t](#)** *utcb=**[l4_utcb\(\)](#)**) noexcept
Detach from this interrupt.
- **[l4_msgtag_t receive](#)** (**[l4_timeout_t](#)** timeout=**[L4_IPC_NEVER](#)**, **[l4_utcb_t](#)** *utcb=**[l4_utcb\(\)](#)**) noexcept
Unmask and wait for this IRQ.

- `l4_msgtag_t wait (l4_umword_t *label, l4_timeout_t timeout=L4_IPC_NEVER, l4_utcb_t *utcb=l4_utcb()) noexcept`
Unmask IRQ and (open) wait for any message.
- `l4_msgtag_t unmask (l4_utcb_t *utcb=l4_utcb()) noexcept`
Unmask this IRQ.

Public Member Functions inherited from [L4::Triggerable](#)

- `l4_msgtag_t trigger (l4_utcb_t *utcb=l4_utcb()) noexcept`
Trigger the object.

Public Member Functions inherited from [L4::Irq_eoi](#)

- `l4_msgtag_t unmask (unsigned irqnum, l4_umword_t *label=0, l4_timeout_t to=L4_IPC_NEVER, l4_utcb_t *utcb=l4_utcb()) noexcept`
Unmask the given interrupt line.

Public Member Functions inherited from [L4::Rcv_endpoint](#)

- `l4_msgtag_t bind_thread (lpc::Cap< Thread > t, l4_umword_t label)`
Bind the IPC receive endpoint to a thread.
- `l4_msgtag_t bind_snd_destination (Cap< Snd_destination > snd_dst, l4_umword_t label)`
Bind a send destination (a thread or thread group) to an IPC receive endpoint.

Public Member Functions inherited from [L4::Kobject](#)

- `l4_msgtag_t dec_refcnt (l4_mword_t diff, l4_utcb_t *utcb=l4_utcb())`
Decrement the in kernel reference counter for the object.

Additional Inherited Members

Protected Types inherited from

[L4::Kobject_2t< Irq, Triggerable, Rcv_endpoint, L4_PROTO_IRQ_SENDER >](#)

- `typedef Irq Class`
The target interface type (inheriting from [Kobject_t](#)).
- `typedef Typeid::Iface< PROTO, Irq > __lface`
The interface description for the derived class.
- `typedef Typeid::Merge_list< Typeid::Iface_list< __lface >, Typeid::Merge_list< typename Triggerable::__lface_list, typename Rcv_endpoint::__lface_list > > __lface_list`
The list of all RPC interfaces provided directly or through inheritance.

Protected Types inherited from [L4::Kobject_t< Triggerable, Irq_eoi, L4_PROTO_IRQ >](#)

- `typedef Triggerable Class`
The target interface type (inheriting from [Kobject_t](#)).
- `typedef Typeid::Iface< PROTO, Triggerable > __lface`
The interface description for the derived class.
- `typedef Typeid::Merge_list< Typeid::Iface_list< __lface >, typename Irq_eoi::__lface_list > __lface_list`
The list of all RPC interfaces provided directly or through inheritance.

Protected Types inherited from**L4::Kobject_t**< Rcv_endpoint, Kobject, L4_PROTO_KOBJECT, Type_info::Demand_t< 1 > >

- typedef **Rcv_endpoint** **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< PROTO, **Rcv_endpoint** > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__Iface** >, typename Kobject::__Iface_list > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from**L4::Kobject_2t**< Irq, Triggerable, Rcv_endpoint, L4_PROTO_IRQ_SENDER >

- **L4::Cap**< **Class** > **c** () const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from**L4::Kobject_t**< Triggerable, Irq_eoi, L4_PROTO_IRQ >

- **L4::Cap**< **Class** > **c** () const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from**L4::Kobject_t**< Rcv_endpoint, Kobject, L4_PROTO_KOBJECT, Type_info::Demand_t< 1 > >

- **L4::Cap**< **Class** > **c** () const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from [L4::Kobject](#)

- **l4_cap_idx_t** **cap** () const noexcept
Return capability selector.

Static Protected Member Functions inherited from**L4::Kobject_2t**< Irq, Triggerable, Rcv_endpoint, L4_PROTO_IRQ_SENDER >

- static void **__check_protocols__** () noexcept
Helper to check for protocol conflicts.

Static Protected Member Functions inherited from**L4::Kobject_t**< Triggerable, Irq_eoi, L4_PROTO_IRQ >

- static void **__check_protocols__** () noexcept
Helper to check for protocol conflicts.

Static Protected Member Functions inherited from**L4::Kobject_t< Rcv_endpoint, Kobject, L4_PROTO_KOBJECT, Type_info::Demand_t< 1 > >**

- static void `__check_protocols__()` noexcept
Helper to check for protocol conflicts.

16.173.1 Detailed Description

C++ [Irq](#) interface, see [IRQs](#) for the C interface.

Note

"IRQ" is short for "interrupt request". This is often used interchangeably for "interrupt"

The [Irq](#) class provides access to abstract interrupts provided by the microkernel. Interrupts may be

- hardware interrupts provided by the platform interrupt controller,
- virtual device interrupts provided by the microkernel's virtual devices (virtual serial or trace buffer) or
- virtual interrupts that can be triggered by user programs (IRQs) via the inherited method [L4::Triggerable::trigger\(\)](#).

For hardware and virtual device interrupts the [Irq](#) object must be bound to an interrupt source, see [L4::lcu](#). To receive interrupts, the [Irq](#) object must be bound to a thread, see [L4::Rcv_endpoint](#).

[Irq](#) objects can be created using a factory, see the [L4::Factory](#) API ([L4::Factory::create\(\)](#)).

Include File

```
#include <l4/sys/irq>
```

For the C interface refer to the [IRQs](#) API for an overview.

Examples

[examples/libs/l4re/c++/shared_ds/ds_clnt.cc](#).

Definition at line 120 of file [irq](#).

16.173.2 Member Function Documentation**16.173.2.1 bind_vcpu()**

```
l4_msgtag_t L4::Irq::bind_vcpu (
    L4::Cap< Thread > const & thread,
    l4_umword_t cfg,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
```

Bind a thread to this [Irq](#) for vCPU interrupt forwarding.

If the interrupt is triggered, the kernel will directly inject the interrupt into the guest. This requires that the thread is currently in extended vCPU user mode. Otherwise the interrupt will stay pending and gets injected on the next vCPU user mode transition. Optionally a doorbell [Irq](#) can be registered on the thread (see [Thread::register_doorbell_irq\(\)](#)) that is triggered in this case.

If a guest has acknowledged the interrupt but has not yet issued an EOI (i.e. the interrupt is in "active" state), it is not possible to bind the [Irq](#) to a new thread object. Either wait for the guest to issue the EOI or [detach\(\)](#) from the current thread. In this case the interrupt will stay active in the guest and it is the responsibility of the VMM to handle the eventual EOI of the guest.

Parameters

<i>thread</i>	Thread object this Irq shall be bound to.
<i>cfg</i>	Architecture specific interrupt configuration.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Returns

Syscall return tag

Return values

<code>-L4_EPERM</code>	Insufficient permissions; see precondition.
<code>-L4_EBUSY</code>	Cannot bind to the new thread because interrupt is active on previous thread and guest has to issue end-of-interrupt first.
<code>-L4_ENOSYS</code>	The kernel does not support direct interrupt forwarding.

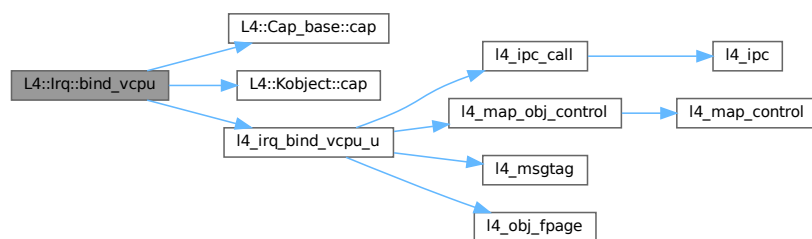
Precondition

The invoked [Irq](#) capability and the capability `thread` both must have the permission [L4_CAP_FPAGE_S](#).

Definition at line 158 of file [irq](#).

References [L4::Cap_base::cap\(\)](#), [L4::Kobject::cap\(\)](#), and [l4_irq_bind_vcpu_u\(\)](#).

Here is the call graph for this function:



16.173.2.2 detach()

```

l4_msgtag_t L4::Irq::detach (
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]

```

Detach from this interrupt.

Parameters

<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .
-------------	--

Returns

Syscall return tag

Return values

0	Successfully detached, there was no interrupt pending.
1	Successfully detached, there was an interrupt pending.
2	Successfully detached, an active vIRQ was abandoned.
-L4_EPERM	Insufficient permissions; see precondition.

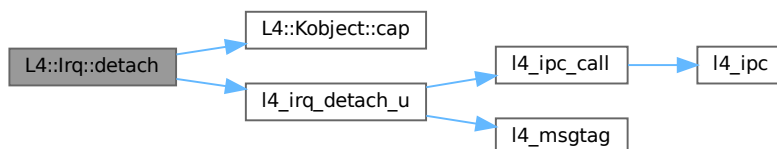
Precondition

The invoked [Irq](#) capability must have the permission [L4_CAP_FPAGE_S](#).

Definition at line 176 of file [irq](#).

References [L4::Kobject::cap\(\)](#), and [l4_irq_detach_u\(\)](#).

Here is the call graph for this function:

**16.173.2.3 receive()**

```

l4_msgtag_t L4::Irq::receive (
    l4_timeout_t timeout = L4_IPC_NEVER,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Unmask and wait for this IRQ.

Parameters

<i>timeout</i>	Timeout.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Returns

Syscall return tag

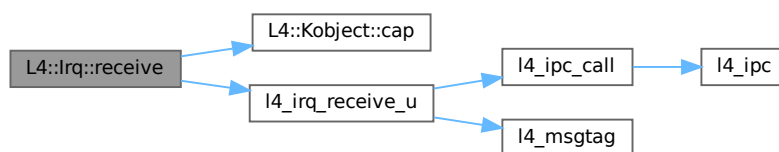
Note

If this is the function normally used for your IRQs consider using [L4::Semaphore](#) instead of [L4::Irq](#).

Definition at line 191 of file [irq](#).

References [L4::Kobject::cap\(\)](#), [L4_IPC_NEVER](#), and [l4_irq_receive_u\(\)](#).

Here is the call graph for this function:

**16.173.2.4 unmask()**

```

l4_msgtag_t L4::Irq::unmask (
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Unmask this IRQ.

Parameters

<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .
-------------	--

Returns

Syscall return tag for a send-only operation, this means there is no return value except [L4_MSGTAG_ERROR](#) indicating success or failure of the send operation. Use [l4_ipc_error\(\)](#) to check for errors and **do not** use [l4_error\(\)](#).

[Irq::wait\(\)](#) and [Irq::receive\(\)](#) operations already include an [unmask\(\)](#), do not use an extra [unmask\(\)](#) in these cases.

Definition at line 221 of file [irq](#).

References [L4_IPC_NEVER](#), and [unmask\(\)](#).

Referenced by [unmask\(\)](#), and [wait\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.173.2.5 wait()

```

l4_msgtag_t L4::Irq::wait (
    l4_umword_t * label,
    l4_timeout_t timeout = L4_IPC_NEVER,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Unmask IRQ and (open) wait for any message.

Parameters

<i>label</i>	The <i>protected label</i> shall be received here.
<i>timeout</i>	Timeout.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Returns

Syscall return tag

Definition at line 204 of file [irq](#).

References [L4_IPC_NEVER](#), and [unmask\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

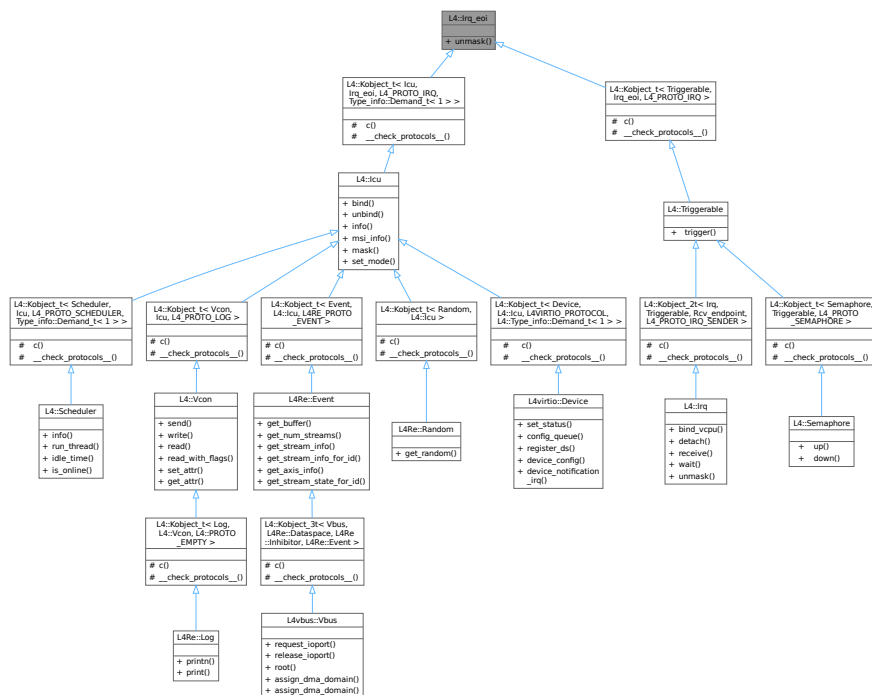
- `l4/sys/irq`

16.174 L4::Irq_eoi Class Reference

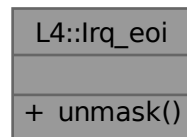
Interface for sending an unmask message to an object.

```
#include <irq>
```

Inheritance diagram for L4::Irq_eoi:



Collaboration diagram for L4::Irq_eoi:



Public Member Functions

- [l4_msgtag_t](#) **unmask** (unsigned irqnum, [l4_umword_t](#) *label=0, [l4_timeout_t](#) to=[L4_IPC_NEVER](#), [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept

Unmask the given interrupt line.

16.174.1 Detailed Description

Interface for sending an unmask message to an object.

The object is usually an ICU or an IRQ.

When the kernel receives an IRQ, it masks the interrupt line at the interrupt controller and immediately acknowledges the interrupt. This interface is used to let the kernel know that userspace has dealt with the IRQ. The kernel will unmask the interrupt line and further IRQs can then be delivered.

See also

[L4::lcu](#), [L4::Irq](#)

Definition at line 37 of file [irq](#).

16.174.2 Member Function Documentation

16.174.2.1 unmask()

```

l4_msgtag_t L4::Irq_eoi::unmask (
    unsigned irqnum,
    l4_umword_t * label = 0,
    l4_timeout_t to = L4_IPC_NEVER,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Unmask the given interrupt line.

When the object is an IRQ, the given interrupt line is ignored and instead the line which the IRQ is bound to (if any) is unmasked.

Its counterpart for explicitly masking an interrupt line is [L4::lcu::mask\(\)](#).

Parameters

	<i>irqnum</i>	The interrupt line that shall be unmasked. Ignored if the object is an IRQ.
out	<i>label</i>	If NULL, this is a send-only unmask. If not NULL, this operation enters an open wait and the <i>protected label</i> shall be received here.
	<i>to</i>	The timeout-pair (send and receive) that shall be used for this operation. The receive timeout is used with a non-NULL <i>label</i> only.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Returns

Syscall return tag. If *label* is NULL, this function performs an IPC send-only operation and there is no return value except [L4_MSGTAG_ERROR](#) indicating success or failure of the send operation. In this case use [l4_ipc_error\(\)](#) to check for errors and **do not** use [l4_error\(\)](#).

Definition at line 64 of file [irq](#).

References [L4::Kobject::cap\(\)](#), and [L4_IPC_NEVER](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

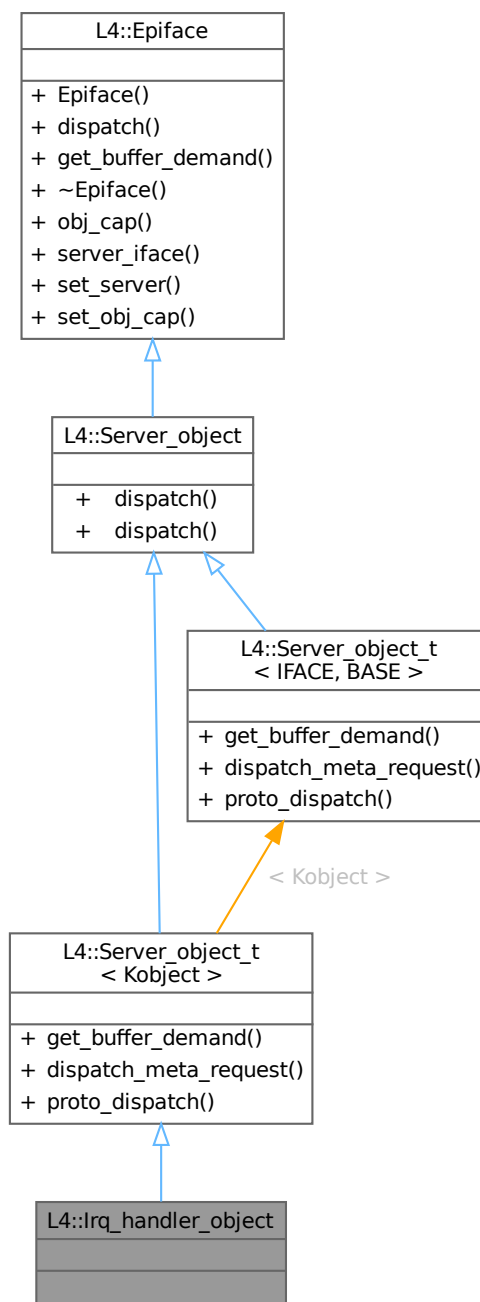
- [l4/sys/irq](#)

16.175 L4::Irq_handler_object Struct Reference

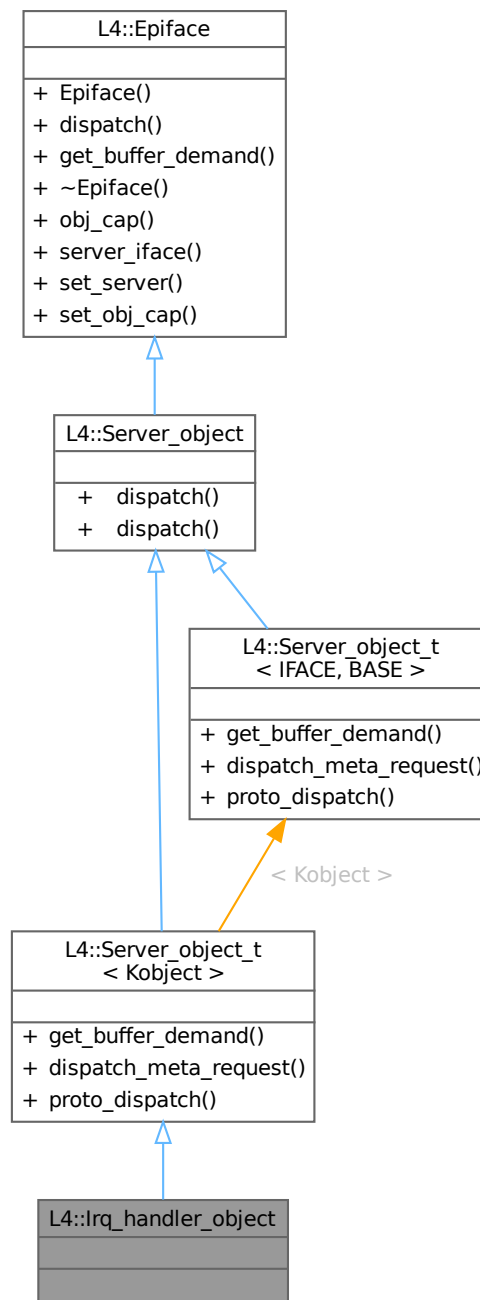
[Server](#) object base class for handling IRQ messages.

```
#include <ipc_server>
```

Inheritance diagram for L4::lrq_handler_object:



Collaboration diagram for L4::Irq_handler_object:



Additional Inherited Members

Public Types inherited from `L4::Server_object_t< Kobject >`

- typedef `Kobject` Interface

Data type of the IPC interface definition.

Public Types inherited from L4::Epiface

- typedef [lpc_svr::Server_iface](#) **Server_iface**
Type for abstract server interface.
- typedef [lpc_svr::Server_iface::Demand](#) **Demand**
Type for server-side receive buffer demand.

Public Member Functions inherited from L4::Server_object_t< Kobject >

- [L4::Server_object::Demand](#) **get_buffer_demand** () const override
- int [dispatch_meta_request](#) ([L4::lpc::lostream](#) &ios)
Implementation of the meta protocol based on IFACE.

Public Member Functions inherited from L4::Server_object

- virtual int [dispatch](#) (unsigned long rights, [lpc::lostream](#) &ios)=0
The abstract handler for client requests to the object.
- [l4_msgtag_t](#) [dispatch](#) ([l4_msgtag_t](#) tag, unsigned rights, [l4_utcb_t](#) *utcb) override
The abstract handler for client requests to the object.

Public Member Functions inherited from L4::Epiface

- **Epiface** ()
Make a server object.
- virtual ~**Epiface** ()=0
Destroy the object.
- Stored_cap [obj_cap](#) () const
Get the capability to the kernel object belonging to this object.
- [Server_iface](#) * [server_iface](#) () const
Get pointer to server interface at which the object is currently registered.
- int [set_server](#) ([Server_iface](#) *srv, [Cap](#)< void > cap, bool managed=false)
Set server registration info for the object.
- void [set_obj_cap](#) ([Cap](#)< void > const &cap)
Deprecated server registration function.

Static Public Member Functions inherited from L4::Server_object_t< Kobject >

- static int [proto_dispatch](#) (THIS *self, [l4_umword_t](#) rights, [L4::lpc::lostream](#) &ios)
Implementation of protocol-based dispatch for this server object.

16.175.1 Detailed Description

[Server](#) object base class for handling IRQ messages.

This server object base class implements the empty interface ([L4::Kobject](#)). The implementation of [Server_object::dispatch\(\)](#) must return [-L4_ENOREPLY](#), because IRQ messages do not handle replies.

Examples

[examples/libs/l4re/c++/shared_ds/ds_srv.cc](#).

Definition at line 161 of file [ipc_server](#).

The documentation for this struct was generated from the following file:

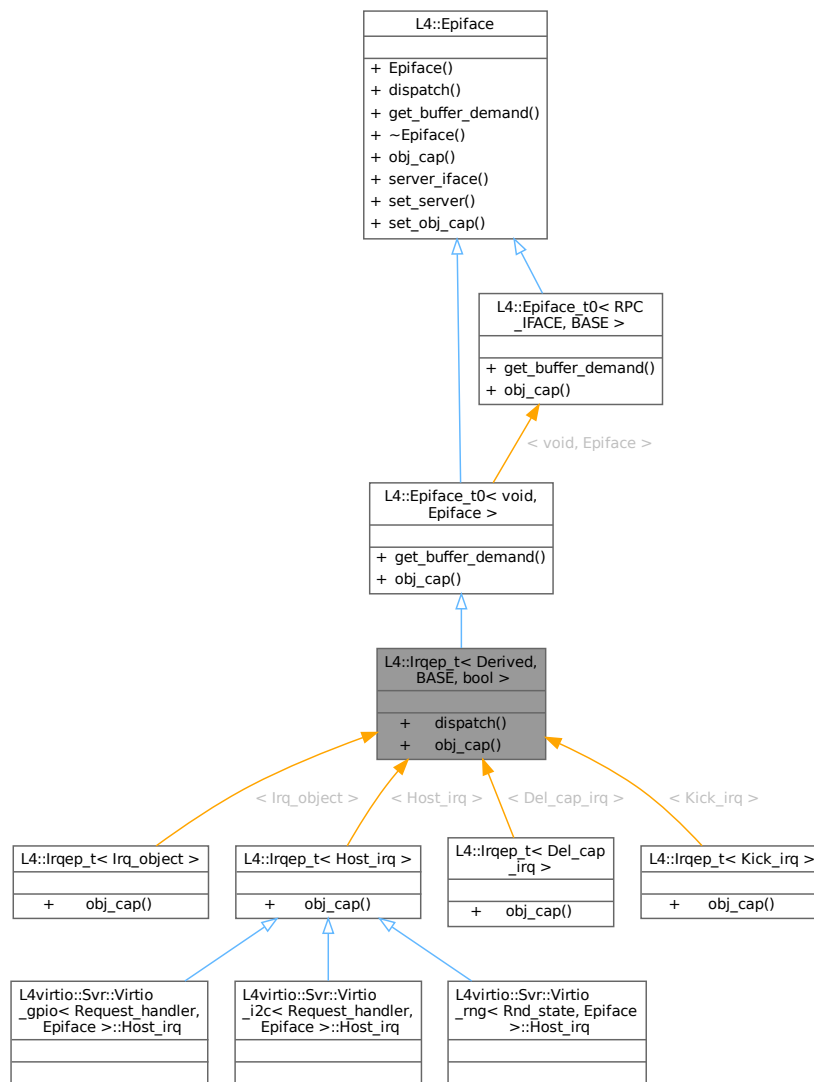
- [l4/cxx/ipc_server](#)

16.176 L4::lrqep_t< Derived, BASE, bool > Struct Template Reference

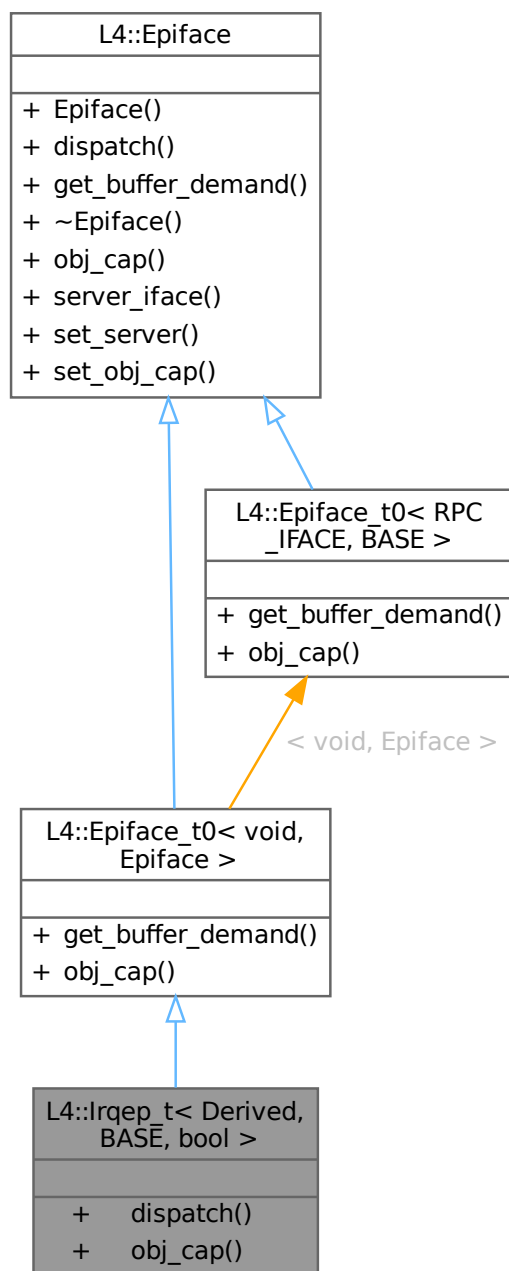
[Epiface](#) implementation for interrupt handlers.

```
#include <ipc_epiface>
```

Inheritance diagram for L4::lrqep_t< Derived, BASE, bool >:



Collaboration diagram for L4::lrqep_t< Derived, BASE, bool >:



Public Member Functions

- `l4_msgtag_t dispatch (l4_msgtag_t, unsigned, l4_utcb_t *)` final
The abstract handler for client requests to the object.
- `Cap< L4::lrq > obj_cap ()` const
Get the (typed) capability to this object.

Public Member Functions inherited from [L4::Epiface_t0< void, Epiface >](#)

- [Type_info::Demand](#) **get_buffer_demand** () const
Get the server-side buffer demand based in IFACE.
- [Cap< void >](#) **obj_cap** () const
Get the (typed) capability to this object.

Public Member Functions inherited from [L4::Epiface](#)

- **Epiface** ()
Make a server object.
- virtual **~Epiface** ()=0
Destroy the object.
- Stored_cap [obj_cap](#) () const
Get the capability to the kernel object belonging to this object.
- [Server_iface](#) * **server_iface** () const
Get pointer to server interface at which the object is currently registered.
- int **set_server** ([Server_iface](#) *srv, [Cap< void >](#) cap, bool managed=false)
Set server registration info for the object.
- void **set_obj_cap** ([Cap< void >](#) const &cap)
Deprecated server registration function.

Additional Inherited Members

Public Types inherited from [L4::Epiface_t0< void, Epiface >](#)

- typedef void **Interface**
Data type of the IPC interface definition.

Public Types inherited from [L4::Epiface](#)

- typedef [lpc_svr::Server_iface](#) **Server_iface**
Type for abstract server interface.
- typedef [lpc_svr::Server_iface::Demand](#) **Demand**
Type for server-side receive buffer demand.

16.176.1 Detailed Description

```
template<typename Derived, typename BASE = Epiface, bool = cxx::is_polymorphic<BASE>::value>
struct L4::Irqep_t< Derived, BASE, bool >
```

[Epiface](#) implementation for interrupt handlers.

Template Parameters

<i>Derived</i>	Irq handler implementation class. The class must provide a single function <code>handle_irq()</code> .
----------------	--

<i>BASE</i>	Base Epiface class.
-------------	-------------------------------------

Definition at line [282](#) of file [ipc_epiface](#).

16.176.2 Member Function Documentation

16.176.2.1 dispatch()

```
template<typename Derived, typename BASE = Epiface, bool = cxx::is_polymorphic<BASE>::value>
l4_msgtag_t L4::lrqep_t< Derived, BASE, bool >::dispatch (
    l4_msgtag_t tag,
    unsigned rights,
    l4_utcb_t * utcb) [inline], [final], [virtual]
```

The abstract handler for client requests to the object.

Parameters

<i>tag</i>	The message tag for this invocation.
<i>rights</i>	The rights bits in the invoked capability.
<i>utcb</i>	The UTCB used for the invocation.

Return values

<i>-L4_ENOREPLY</i>	No reply message is send.
<i><0</i>	Error, reply with error code.
<i>>=0</i>	Success, reply with return value.

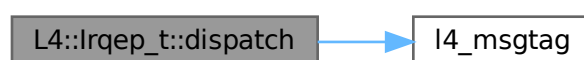
This function must be implemented by application specific server objects.

Implements [L4::Epiface](#).

Definition at line [284](#) of file [ipc_epiface](#).

References [L4_ENOREPLY](#), and [l4_msgtag\(\)](#).

Here is the call graph for this function:



16.176.2.2 obj_cap()

```
template<typename Derived, typename BASE = Epiface, bool = cxx::is_polymorphic<BASE>::value>  
Cap< L4::Irq > L4::Irqep_t< Derived, BASE, bool >::obj_cap () const [inline]
```

Get the (typed) capability to this object.

Returns

[Irq](#) capability for the kernel object behind the server.

Definition at line 294 of file [ipc_epiface](#).

References [L4::cap_cast\(\)](#).

Here is the call graph for this function:



The documentation for this struct was generated from the following file:

- `l4/sys/cxx/ipc_epiface`

16.177 L4::Kip::Mem_desc Class Reference

Memory descriptors stored in the kernel interface page.

```
#include <kip>
```

Collaboration diagram for L4::Kip::Mem_desc:

L4::Kip::Mem_desc
<ul style="list-style-type: none"> + Mem_desc() + start() + end() + size() + type() + sub_type() + is_virtual() + eager_map() + set() + first() + count() + count() + all() + all()

Public Types

- enum [Mem_type](#) {
[Undefined](#) = 0x0 , [Conventional](#) = 0x1 , [Reserved](#) = 0x2 , [Dedicated](#) = 0x3 ,
[Shared](#) = 0x4 , [Info](#) = 0xd , [Bootloader](#) = 0xe , [Arch](#) = 0xf }
Memory types.
- enum [Info_sub_type](#) { [Info_acpi_rsdp](#) = 0 }
Memory sub types for the [Mem_type::Info](#) type.
- enum [Arch_sub_type_common](#) { [Arch_acpi_tables](#) = 3 , [Arch_acpi_nvs](#) = 4 }
Common sub types across all architectures for the [Mem_type::Arch](#) type.

Public Member Functions

- [Mem_desc](#) (unsigned long [start](#), unsigned long [end](#), [Mem_type](#) t, unsigned char st=0, bool virt=false, bool eager=false) noexcept
Initialize memory descriptor.
- unsigned long [start](#) () const noexcept
Return start address of memory descriptor.
- unsigned long [end](#) () const noexcept
Return end address of memory descriptor.
- unsigned long [size](#) () const noexcept
Return size of region described by the memory descriptor.
- [Mem_type type](#) () const noexcept

Return type of the memory descriptor.

- unsigned char [sub_type](#) () const noexcept

Return sub-type of the memory descriptor.

- unsigned [is_virtual](#) () const noexcept

Return whether the memory descriptor describes a virtual or physical region.

- unsigned [eager_map](#) () const noexcept

Return whether the region shall be eligible for eager mapping in sigma0 or the root task.

- void [set](#) (unsigned long [start](#), unsigned long [end](#), [Mem_type](#) t, unsigned char st=0, bool virt=false, bool eager=false) noexcept

Set values of a memory descriptor.

Static Public Member Functions

- static [Mem_desc](#) * [first](#) ([l4_kernel_info_t](#) *kip) noexcept

Get first memory descriptor.

- static unsigned long [count](#) ([l4_kernel_info_t](#) const *kip) noexcept

Return number of memory descriptors stored in the kernel info page.

- static void [count](#) ([l4_kernel_info_t](#) *kip, unsigned count) noexcept

Set number of memory descriptors.

- static [cxx::static_vector](#)< [Mem_desc](#) const > [all](#) ([l4_kernel_info_t](#) const *kip)

Return enumerable list of memory descriptors.

- static [cxx::static_vector](#)< [Mem_desc](#) > [all](#) ([l4_kernel_info_t](#) *kip)

Return enumerable list of memory descriptors.

16.177.1 Detailed Description

Memory descriptors stored in the kernel interface page.

Include File

```
#include <l4/sys/kip>
```

Definition at line 42 of file [kip](#).

16.177.2 Member Enumeration Documentation

16.177.2.1 Arch_sub_type_common

```
enum L4::Kip::Mem\_desc::Arch\_sub\_type\_common
```

Common sub types across all architectures for the [Mem_type::Arch](#) type.

Enumerator

Arch_acpi_tables	Firmware ACPI tables.
Arch_acpi_nvs	Firmware reserved address space.

Definition at line 72 of file [kip](#).

16.177.2.2 Info_sub_type

enum [L4::Kip::Mem_desc::Info_sub_type](#)

Memory sub types for the [Mem_type::Info](#) type.

Enumerator

Info_acpi_rsdp	Physical address of the ACPI root pointer.
----------------	--

Definition at line 64 of file [kip](#).

16.177.2.3 Mem_type

```
enum L4::Kip::Mem_desc::Mem_type
```

Memory types.

Enumerator

Undefined	Undefined memory.
Conventional	Conventional memory.
Reserved	Reserved region, do not use this memory.
Dedicated	Dedicated.
Shared	Shared.
Info	Info by boot loader.
Bootloader	Memory belongs to the boot loader.
Arch	Architecture specific memory.

Definition at line 48 of file [kip](#).

16.177.3 Constructor & Destructor Documentation

16.177.3.1 Mem_desc()

```
L4::Kip::Mem_desc::Mem_desc (  
    unsigned long start,  
    unsigned long end,  
    Mem_type t,  
    unsigned char st = 0,  
    bool virt = false,  
    bool eager = false) [inline], [noexcept]
```

Initialize memory descriptor.

Parameters

<i>start</i>	Start address
<i>end</i>	End address
<i>t</i>	Memory type
<i>st</i>	Memory subtype, defaults to 0
<i>virt</i>	True for virtual memory, false for physical memory, defaults to physical

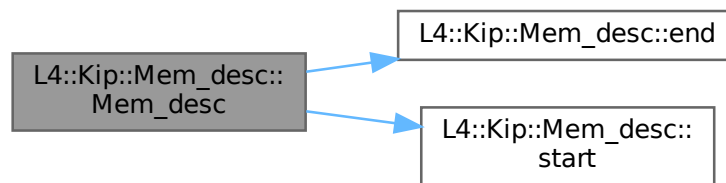
<i>eager</i>	The region shall be eligible for eager mapping in sigma0 or the root task. This is just an optimization to prevent on-demand paging.
--------------	--

Definition at line 158 of file [kip](#).

References [end\(\)](#), and [start\(\)](#).

Referenced by [first\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.177.4 Member Function Documentation

16.177.4.1 `all()` [1/2]

```

cxx::static_vector< Mem_desc > L4::Kip::Mem_desc::all (
    l4_kernel_info_t * kip) [inline], [static]
  
```

Return enumerable list of memory descriptors.

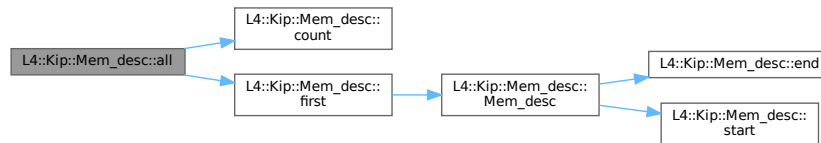
Parameters

<i>kip</i>	Pointer to the kernel info page.
------------	----------------------------------

Definition at line 139 of file [kip](#).

References [count\(\)](#), and [first\(\)](#).

Here is the call graph for this function:



16.177.4.2 [all\(\)](#) [2/2]

```

cxx::static_vector< Mem_desc const > L4::Kip::Mem_desc::all (
    l4_kernel_info_t const * kip) [inline], [static]
  
```

Return enumerable list of memory descriptors.

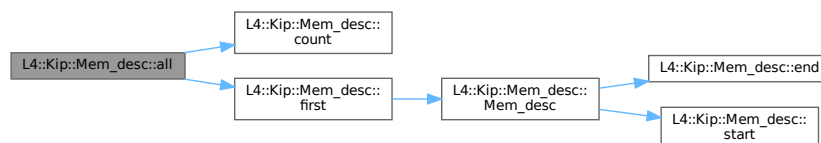
Parameters

<i>kip</i>	Pointer to the kernel info page.
------------	----------------------------------

Definition at line 128 of file [kip](#).

References [count\(\)](#), and [first\(\)](#).

Here is the call graph for this function:



16.177.4.3 [count\(\)](#) [1/2]

```

void L4::Kip::Mem_desc::count (
    l4_kernel_info_t * kip,
    unsigned count) [inline], [static], [noexcept]
  
```

Set number of memory descriptors.

Parameters

<i>kip</i>	Pointer to the kernel info page
<i>count</i>	Number of memory descriptors

Definition at line 118 of file [kip](#).

References [count\(\)](#).

Here is the call graph for this function:



16.177.4.4 count() [2/2]

```
unsigned long L4::Kip::Mem_desc::count (  
    l4_kernel_info_t const * kip) [inline], [static], [noexcept]
```

Return number of memory descriptors stored in the kernel info page.

Parameters

<i>kip</i>	Pointer to the kernel info page
------------	---------------------------------

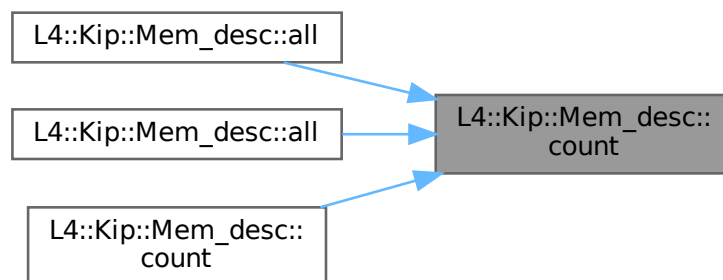
Returns

Number of memory descriptors in the kernel info page.

Definition at line 107 of file [kip](#).

Referenced by [all\(\)](#), [all\(\)](#), and [count\(\)](#).

Here is the caller graph for this function:



16.177.4.5 end()

```
unsigned long L4::Kip::Mem_desc::end () const [inline], [noexcept]
```

Return end address of memory descriptor.

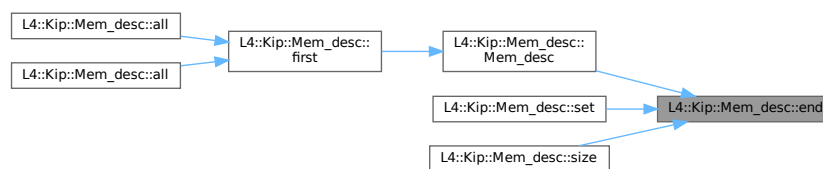
Returns

End address of memory descriptor

Definition at line 177 of file [kip](#).

Referenced by [Mem_desc\(\)](#), [set\(\)](#), and [size\(\)](#).

Here is the caller graph for this function:



16.177.4.6 first()

```
Mem_desc * L4::Kip::Mem_desc::first (
    l4_kernel_info_t * kip) [inline], [static], [noexcept]
```

Get first memory descriptor.

Parameters

<i>kip</i>	Pointer to the kernel info page
------------	---------------------------------

Returns

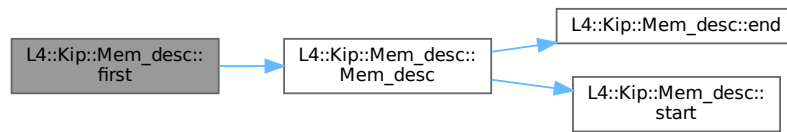
First memory descriptor stored in the kernel info page

Definition at line 89 of file [kip](#).

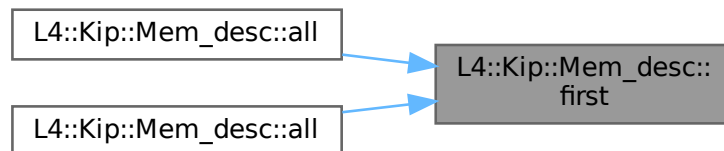
References [Mem_desc\(\)](#).

Referenced by [all\(\)](#), and [all\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.177.4.7 is_virtual()

```
unsigned L4::Kip::Mem_desc::is_virtual () const [inline], [noexcept]
```

Return whether the memory descriptor describes a virtual or physical region.

Returns

True for virtual region, false for physical region.

Definition at line 209 of file [kip](#).

16.177.4.8 set()

```
void L4::Kip::Mem_desc::set (
    unsigned long start,
    unsigned long end,
    Mem_type t,
    unsigned char st = 0,
    bool virt = false,
    bool eager = false) [inline], [noexcept]
```

Set values of a memory descriptor.

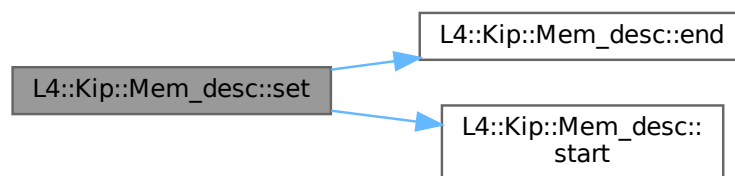
Parameters

<i>start</i>	Start address
<i>end</i>	End address
<i>t</i>	Memory type
<i>st</i>	Sub-type, defaults to 0
<i>virt</i>	Virtual or physical memory region, defaults to physical
<i>eager</i>	The region shall be eligible for eager mapping in sigma0 or the root task. This is just an optimization to prevent on-demand paging.

Definition at line 229 of file [kip](#).

References [end\(\)](#), and [start\(\)](#).

Here is the call graph for this function:



16.177.4.9 `size()`

```
unsigned long L4::Kip::Mem_desc::size () const [inline], [noexcept]
```

Return size of region described by the memory descriptor.

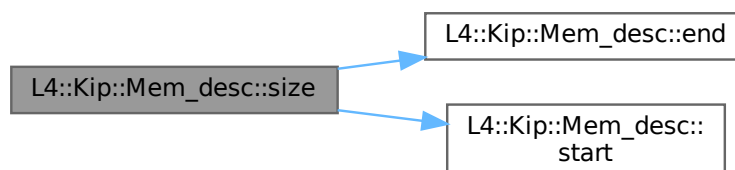
Returns

Size of the region described by the memory descriptor

Definition at line 184 of file [kip](#).

References [end\(\)](#), and [start\(\)](#).

Here is the call graph for this function:



16.177.4.10 start()

```
unsigned long L4::Kip::Mem_desc::start () const [inline], [noexcept]
```

Return start address of memory descriptor.

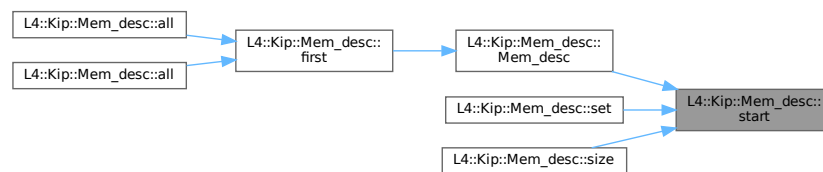
Returns

Start address of memory descriptor

Definition at line 170 of file [kip](#).

Referenced by [Mem_desc\(\)](#), [set\(\)](#), and [size\(\)](#).

Here is the caller graph for this function:

**16.177.4.11 sub_type()**

```
unsigned char L4::Kip::Mem_desc::sub_type () const [inline], [noexcept]
```

Return sub-type of the memory descriptor.

Returns

Sub-type of the memory descriptor

Definition at line 201 of file [kip](#).

16.177.4.12 type()

```
Mem\_type L4::Kip::Mem_desc::type () const [inline], [noexcept]
```

Return type of the memory descriptor.

Returns

Type of the memory descriptor

Definition at line 191 of file [kip](#).

The documentation for this class was generated from the following file:

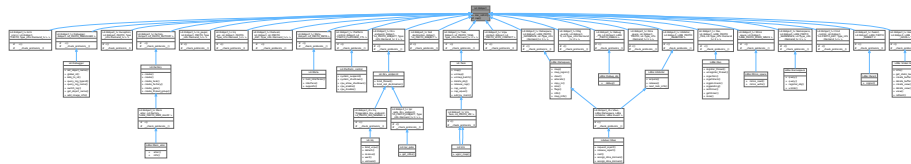
- [l4/sys/kip](#)

16.178 L4::Kobject Class Reference

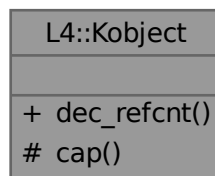
Base class for all kinds of kernel objects and remote objects, referenced by capabilities.

```
#include <kobject>
```

Inheritance diagram for L4::Kobject:



Collaboration diagram for L4::Kobject:



Public Member Functions

- [l4_msgtag_t dec_refcnt](#) ([l4_mword_t diff](#), [l4_utcb_t *utcb=l4_utcb\(\)](#))
Decrement the in kernel reference counter for the object.

Protected Member Functions

- [l4_cap_idx_t cap](#) () const noexcept
Return capability selector.

16.178.1 Detailed Description

Base class for all kinds of kernel objects and remote objects, referenced by capabilities.

Include File

```
#include <l4/sys/capability>
```

This is the base class for all remote objects accessible using RPC. However, subclasses do not directly inherit from [L4::Kobject](#) but *must* use [L4::Kobject_t](#) ([L4::Kobject_0t](#), [L4::Kobject_2t](#), [L4::Kobject_3t](#), or [L4::Kobject_x](#)) for inheritance, otherwise these classes cannot be used as RPC interfaces.

Attention

Objects derived from [Kobject](#) *must* never add any data to those objects. Kobjects can act only as proxy object for encapsulating object invocations.

Definition at line [36](#) of file [kobject](#).

16.178.2 Member Function Documentation

16.178.2.1 cap()

```
l4_cap_idx_t L4::Kobject::cap () const [inline], [protected], [noexcept]
```

Return capability selector.

Returns

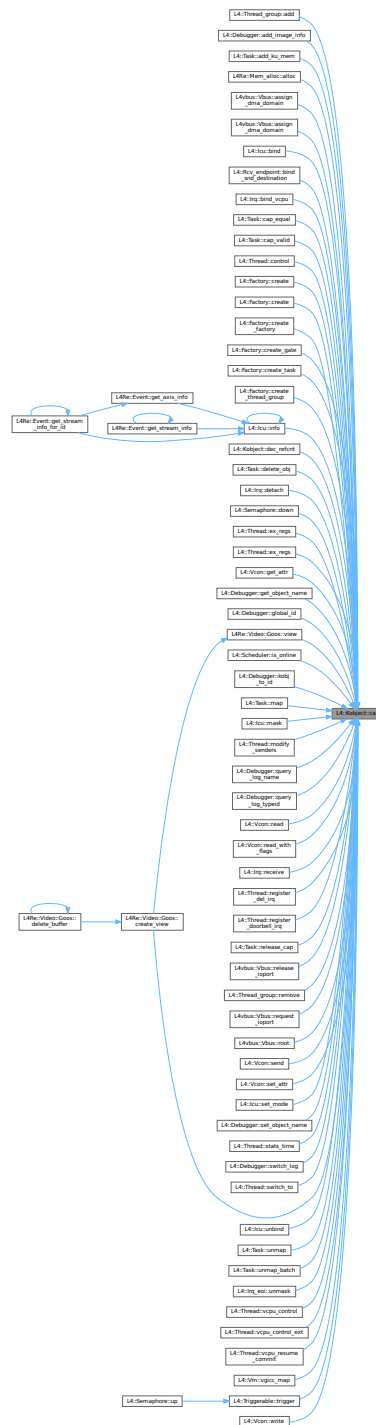
Capability selector.

This method is for derived classes to gain access to the actual capability selector.

Definition at line 69 of file [kobject](#).

Referenced by [L4::Thread_group::add\(\)](#), [L4::Debugger::add_image_info\(\)](#), [L4::Task::add_ku_mem\(\)](#), [L4Re::Mem_alloc::alloc\(\)](#), [L4vbus::Vbus::assign_dma_domain\(\)](#), [L4vbus::Vbus::assign_dma_domain\(\)](#), [L4::lcu::bind\(\)](#), [L4::Rcv_endpoint::bind_snd_destination\(\)](#), [L4::Irq::bind_vcpu\(\)](#), [L4::Task::cap_equal\(\)](#), [L4::Task::cap_valid\(\)](#), [L4::Thread::control\(\)](#), [L4::Factory::create\(\)](#), [L4::Factory::create\(\)](#), [L4::Factory::create_factory\(\)](#), [L4::Factory::create_gate\(\)](#), [L4::Factory::create_task\(\)](#), [L4::Factory::create_thread_group\(\)](#), [L4Re::Video::Goos::create_view\(\)](#), [dec_refcnt\(\)](#), [L4::Task::delete_obj\(\)](#), [L4::Irq::detach\(\)](#), [L4::Semaphore::down\(\)](#), [L4::Thread::ex_regs\(\)](#), [L4::Thread::ex_regs\(\)](#), [L4::Vcon::get_attr\(\)](#), [L4::Debugger::get_object_name\(\)](#), [L4::Debugger::global_id\(\)](#), [L4::lcu::info\(\)](#), [L4::Scheduler::is_online\(\)](#), [L4::Debugger::kobj_to_id\(\)](#), [L4::Task::map\(\)](#), [L4::lcu::mask\(\)](#), [L4::Thread::modify_senders\(\)](#), [L4::Debugger::query_log_name\(\)](#), [L4::Debugger::query_log_typeid\(\)](#), [L4::Vcon::read\(\)](#), [L4::Vcon::read_with_flags\(\)](#), [L4::Irq::receive\(\)](#), [L4::Thread::register_del_irq\(\)](#), [L4::Thread::register_doorbell_irq\(\)](#), [L4::Task::release_cap\(\)](#), [L4vbus::Vbus::release_ioport\(\)](#), [L4::Thread_group::remove\(\)](#), [L4vbus::Vbus::request_ioport\(\)](#), [L4vbus::Vbus::root\(\)](#), [L4::Vcon::send\(\)](#), [L4::Vcon::set_attr\(\)](#), [L4::lcu::set_mode\(\)](#), [L4::Debugger::set_object_name\(\)](#), [L4::Thread::stats_time\(\)](#), [L4::Debugger::switch_log\(\)](#), [L4::Thread::switch_to\(\)](#), [L4::Triggerable::trigger\(\)](#), [L4::lcu::unbind\(\)](#), [L4::Task::unmap\(\)](#), [L4::Task::unmap_batch\(\)](#), [L4::Irq_eoi::unmask\(\)](#), [L4::Thread::vcpu_control\(\)](#), [L4::Thread::vcpu_control_ext\(\)](#), [L4::Thread::vcpu_resume_commit\(\)](#), [L4::Vm::vgicc_map\(\)](#), [L4Re::Video::Goos::view\(\)](#), and [L4::Vcon::write\(\)](#).

Here is the caller graph for this function:



16.178.2.2 dec_refcnt()

```
14_msgtag_t L4::Kobject::dec_refcnt (
    14_mword_t diff,
    14_utcb_t * utcb = 14_utcb()) [inline]
```


Decrement the in kernel reference counter for the object.

Parameters

<i>diff</i>	The delta that shall be subtracted from the reference count.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Returns

Syscall return tag

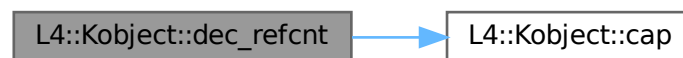
This function is intended for servers to be able to remove the servers own capability from the counted references. This leads to the semantics that the kernel will delete the object even if the capability of the server is valid. The server can detect the deletion by polling its capabilities or by using the IPC-gate deletion IRQs. And to cleanup if the clients dropped the last reference (capability) to the object.

This function only succeeds on a kernel object of type [L4::lpc_gate](#) which has the server right ([L4_FPAGE_C_IPCGATE_SVR](#)). For other kernel objects, -L4_ENOSYS is returned.

Definition at line 100 of file [kobject](#).

References [cap\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

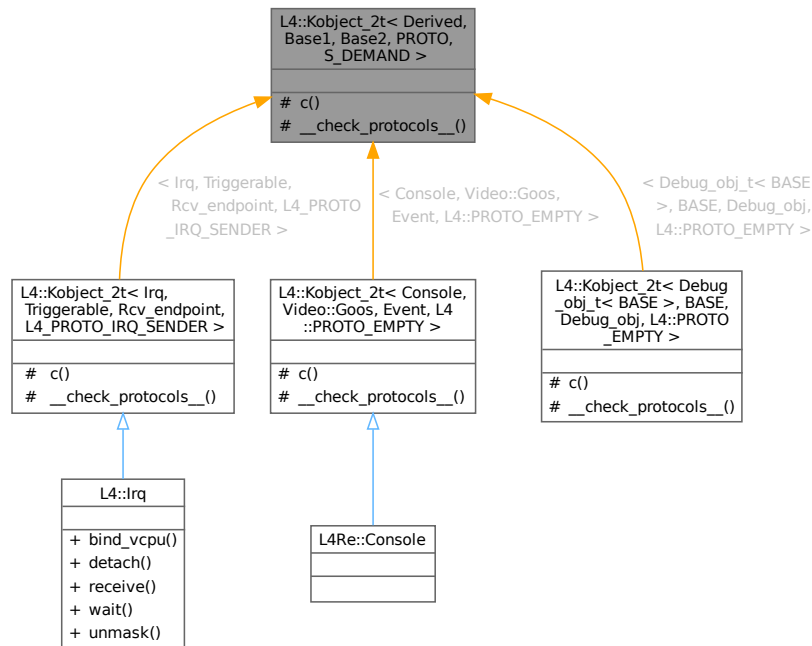
- [l4/sys/kobject](#)

16.179 L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND > Class Template Reference

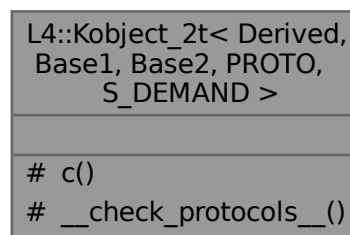
Helper class to create an [L4Re](#) interface class that is derived from two base classes (see [L4::Kobject_t](#)).

```
#include <l4/sys/capability>
```

Inheritance diagram for L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND >:



Collaboration diagram for L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND >:



Protected Types

- typedef `Derived` [Class](#)
The target interface type (inheriting from [Kobject_t](#)).
- typedef `Typeid::Iface< PROTO, Derived > __lf ace`
The interface description for the derived class.
- typedef `Typeid::Merge_list< Typeid::Iface_list< __lf ace >, Typeid::Merge_list< typename Base1::__lf ace←_list, typename Base2::__lf ace_list > > __lf ace_list`
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions

- [L4::Cap< Class > c \(\)](#) const noexcept
Get the capability to ourselves.

Static Protected Member Functions

- static void [__check_protocols__ \(\)](#) noexcept
Helper to check for protocol conflicts.

16.179.1 Detailed Description

```
template<typename Derived, typename Base1, typename Base2, long PROTO = PROTO_ANY, typename
S_DEMAND = Type_info::Demand_t<>>
class L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND >
```

Helper class to create an [L4Re](#) interface class that is derived from two base classes (see [L4::Kobject_t](#)).

Template Parameters

<i>Derived</i>	is the name of the new interface.
<i>Base1</i>	is the name of the interface's first base class.
<i>Base2</i>	is the name of the interface's second base class.
<i>PROTO</i>	may be set to the statically assigned protocol number used to communicate with this interface.
<i>S_DEMAND</i>	type defining the demand of server-side resources for this interface, usually a L4::Type_info::Demand_t . This value must describe the server-side resources needed by the interface itself, the resource demand of the base interfaces (Base1 and Base2) are automatically included.

The typical usage pattern is shown in the following code snippet. The semantics of this example is an interface `My_iface` that is derived from [L4::Icu](#) and [L4Re::Dataspace](#).

```
class My_iface : public L4::Kobject_2t<My_iface, L4::Icu, L4Re::Dataspace>
{
    ...
};
```

Definition at line [827](#) of file [__typeinfo.h](#).

16.179.2 Member Typedef Documentation

16.179.2.1 __Iface

```
template<typename Derived, typename Base1, typename Base2, long PROTO = PROTO_ANY, typename
S_DEMAND = Type_info::Demand_t<>>
typedef Typeid::Iface<PROTO, Derived> L4::Kobject\_2t< Derived, Base1, Base2, PROTO, S\_DEMAND
>::\_\_Iface [protected]
```

The interface description for the derived class.

Definition at line [833](#) of file [__typeinfo.h](#).

16.179.2.2 __Iface_list

```
template<typename Derived, typename Base1, typename Base2, long PROTO = PROTO_ANY, typename
S_DEMAND = Type_info::Demand_t<>>
typedef Typeid::Merge_list< Typeid::Iface_list<__Iface>, Typeid::Merge_list< typename Base1<
::__Iface_list, typename Base2::__Iface_list > > L4::Kobject_2t< Derived, Base1, Base2, PROTO,
S_DEMAND >::__Iface_list [protected]
```

The list of all RPC interfaces provided directly or through inheritance.

Definition at line 841 of file [__typeinfo.h](#).

16.179.2.3 Class

```
template<typename Derived, typename Base1, typename Base2, long PROTO = PROTO_ANY, typename
S_DEMAND = Type_info::Demand_t<>>
typedef Derived L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND >::Class [protected]
```

The target interface type (inheriting from [Kobject_t](#)).

Definition at line 831 of file [__typeinfo.h](#).

16.179.3 Member Function Documentation

16.179.3.1 __check_protocols__()

```
template<typename Derived, typename Base1, typename Base2, long PROTO = PROTO_ANY, typename
S_DEMAND = Type_info::Demand_t<>>
void L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND >::__check_protocols__ () [inline],
[static], [protected], [noexcept]
```

Helper to check for protocol conflicts.

Definition at line 844 of file [__typeinfo.h](#).

16.179.3.2 c()

```
template<typename Derived, typename Base1, typename Base2, long PROTO = PROTO_ANY, typename
S_DEMAND = Type_info::Demand_t<>>
L4::Cap< Class > L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND >::c () const [inline],
[protected], [noexcept]
```

Get the capability to ourselves.

Definition at line 863 of file [__typeinfo.h](#).

The documentation for this class was generated from the following file:

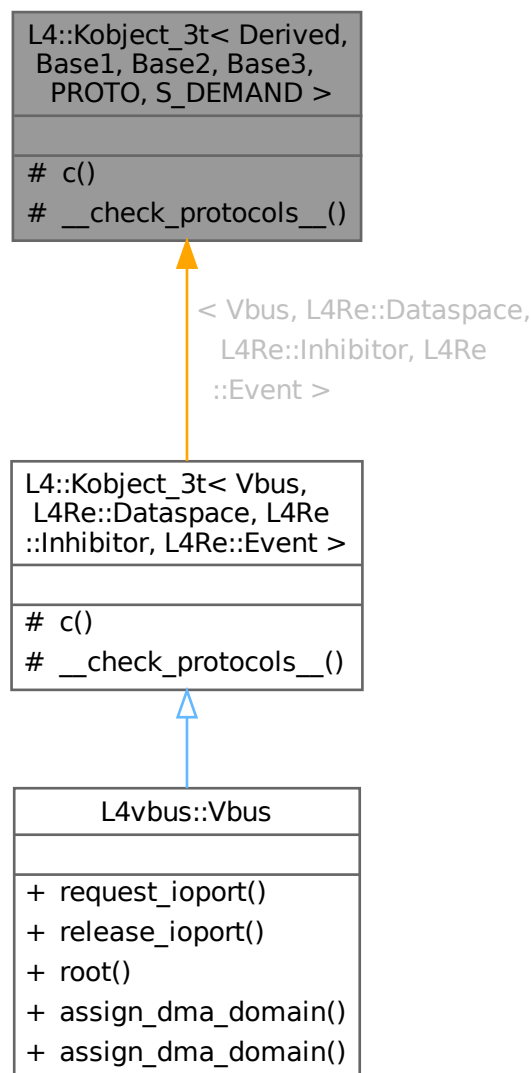
- [l4/sys/__typeinfo.h](#)

16.180 L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND > Struct Template Reference

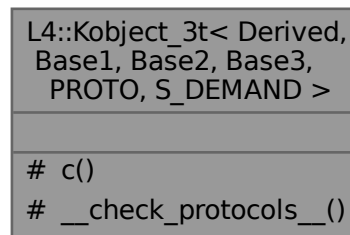
Helper class to create an [L4Re](#) interface class that is derived from three base classes (see [L4::Kobject_t](#)).

```
#include <l4/sys/capability>
```

Inheritance diagram for L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND >:



Collaboration diagram for L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND >:



Protected Types

- typedef Derived [Class](#)
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< PROTO, Derived > [__iface](#)
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< [__iface](#) >, Typeid::Merge_list< typename Base1::__iface_list, Typeid::Merge_list< typename Base2::__iface_list, typename Base3::__iface_list > > > [__iface_list](#)
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions

- [L4::Cap](#)< [Class](#) > [c](#) () const noexcept
Get the capability to ourselves.

Static Protected Member Functions

- static void [__check_protocols__](#) () noexcept
Helper to check for protocol conflicts.

16.180.1 Detailed Description

```
template<typename Derived, typename Base1, typename Base2, typename Base3, long PROTO = PROTO_
_ANY, typename S_DEMAND = Type_info::Demand_t<>>
struct L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND >
```

Helper class to create an [L4Re](#) interface class that is derived from three base classes (see [L4 : Kobject_t](#)).

Template Parameters

<i>Derived</i>	is the name of the new interface.
----------------	-----------------------------------

<i>Base1</i>	is the name of the interface's first base class.
<i>Base2</i>	is the name of the interface's second base class.
<i>Base3</i>	is the name of the interfaces third base class.
<i>PROTO</i>	may be set to the statically assigned protocol number used to communicate with this interface.
<i>S_DEMAND</i>	type defining the demand on server-side resources for this interface, usually a L4::Type_info::Demand_t . This value must describe the server-side resources needed by the interface itself, the resource demand of the base interfaces (Base1 and Base2) are automatically included.

See also

[L4::Kobject_t](#), [L4::Kobject_2t](#), [L4::Kobject_0t](#), [L4::Kobject_x](#)

Definition at line 930 of file [__typeinfo.h](#).

16.180.2 Member Typedef Documentation

16.180.2.1 __Iface

```
template<typename Derived, typename Base1, typename Base2, typename Base3, long PROTO = PROTO↵
_ANY, typename S_DEMAND = Type_info::Demand_t<>>
typedef Typeid::Iface<PROTO, Derived> L4::Kobject\_3t< Derived, Base1, Base2, Base3, PROTO,
S_DEMAND >::__Iface [protected]
```

The interface description for the derived class.

Definition at line 936 of file [__typeinfo.h](#).

16.180.2.2 __Iface_list

```
template<typename Derived, typename Base1, typename Base2, typename Base3, long PROTO = PROTO↵
_ANY, typename S_DEMAND = Type_info::Demand_t<>>
typedef Typeid::Merge_list< Typeid::Iface_list<\_\_Iface>, Typeid::Merge_list< typename Base1↵
::__Iface_list, Typeid::Merge_list< typename Base2::__Iface_list, typename Base3::__Iface↵
_list > > > L4::Kobject\_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND >::__Iface_list
[protected]
```

The list of all RPC interfaces provided directly or through inheritance.

Definition at line 947 of file [__typeinfo.h](#).

16.180.2.3 Class

```
template<typename Derived, typename Base1, typename Base2, typename Base3, long PROTO = PROTO↵
_ANY, typename S_DEMAND = Type_info::Demand_t<>>
typedef Derived L4::Kobject\_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND >::Class [protected]
```

The target interface type (inheriting from [Kobject_t](#)).

Definition at line 934 of file [__typeinfo.h](#).

16.180.3 Member Function Documentation

16.180.3.1 __check_protocols__()

```
template<typename Derived, typename Base1, typename Base2, typename Base3, long PROTO = PROTO←
_ANY, typename S_DEMAND = Type_info::Demand_t<>>
void L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND >::__check_protocols__ ()
[inline], [static], [protected], [noexcept]
```

Helper to check for protocol conflicts.

Definition at line 950 of file [__typeinfo.h](#).

16.180.3.2 c()

```
template<typename Derived, typename Base1, typename Base2, typename Base3, long PROTO = PROTO←
_ANY, typename S_DEMAND = Type_info::Demand_t<>>
L4::Cap< Class > L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND >::c () const
[inline], [protected], [noexcept]
```

Get the capability to ourselves.

Definition at line 978 of file [__typeinfo.h](#).

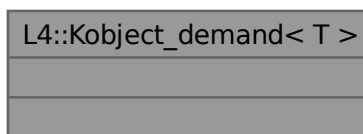
The documentation for this struct was generated from the following file:

- [l4/sys/__typeinfo.h](#)

16.181 L4::Kobject_demand< T > Struct Template Reference

Get the combined server-side resource requirements for all type T...

Collaboration diagram for L4::Kobject_demand< T >:



16.181.1 Detailed Description

```
template<typename ... T>
struct L4::Kobject_demand< T >
```

Get the combined server-side resource requirements for all type T...

Template Parameters

T	List of IPC interface types for which the combined server-side resource requirements shall be calculated.
---	---

Definition at line 1031 of file [__typeinfo.h](#).

The documentation for this struct was generated from the following file:

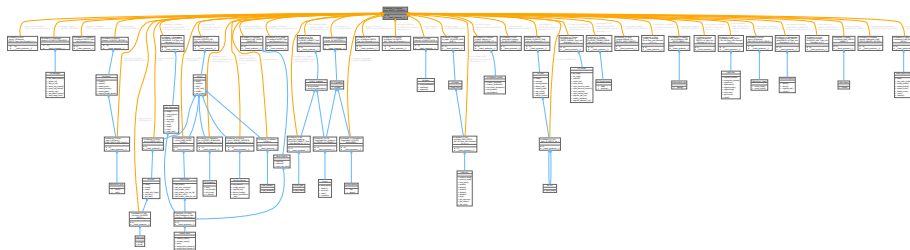
- l4/sys/[__typeinfo.h](#)

16.182 L4::Kobject_t< Derived, Base, PROTO, S_DEMAND > Class Template Reference

Helper class to create an [L4Re](#) interface class that is derived from a single base class.

```
#include <l4/sys/capability>
```

Inheritance diagram for L4::Kobject_t< Derived, Base, PROTO, S_DEMAND >:



Collaboration diagram for L4::Kobject_t< Derived, Base, PROTO, S_DEMAND >:

L4::Kobject_t< Derived, Base, PROTO, S_DEMAND >
<pre># c() # __check_protocols__()</pre>

Protected Types

- typedef Derived **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< PROTO, Derived > **__iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< [__iface](#) >, typename Base::__iface_list > **__iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions

- [L4::Cap](#)< [Class](#) > [c](#) () const noexcept
Get the capability to ourselves.

Static Protected Member Functions

- static void [__check_protocols__](#) () noexcept
Helper to check for protocol conflicts.

16.182.1 Detailed Description

```
template<typename Derived, typename Base, long PROTO = PROTO_ANY, typename S_DEMAND = Type_↵
_info::Demand_t<>>
class L4::Kobject_t< Derived, Base, PROTO, S_DEMAND >
```

Helper class to create an [L4Re](#) interface class that is derived from a single base class.

Template Parameters

<i>Derived</i>	is the name of the new interface.
<i>Base</i>	is the name of the interfaces single base class.
<i>PROTO</i>	may be set to the statically assigned protocol number used to communicate with this interface.
<i>S_DEMAND</i>	type defining the demand on server-side resources for this interface, usually a L4::Type_info::Demand_t . This value must describe the server-side resources needed by the interface itself, the resource demand of the base interface <i>Base</i> is automatically included.

The typical usage pattern is shown in the following code snippet. The semantics of this example is an interface [My_iface](#) that is derived from [L4::Kobject](#).

```
class My_iface : public L4::Kobject_t<My_iface, L4::Kobject>
{
    ...
};
```

Examples

[examples/clntsrv/src/shared.h](#).

Definition at line [749](#) of file [__typeinfo.h](#).

The documentation for this class was generated from the following file:

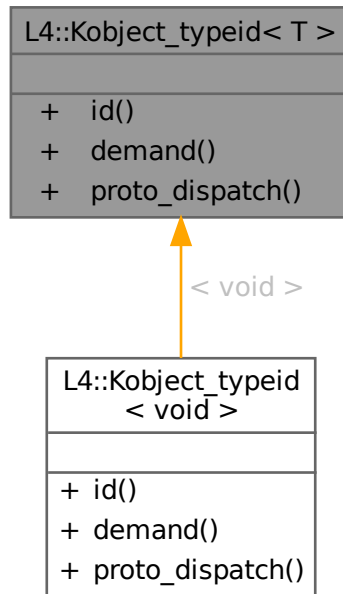
- [l4/sys/__typeinfo.h](#)

16.183 L4::Kobject_typeid< T > Struct Template Reference

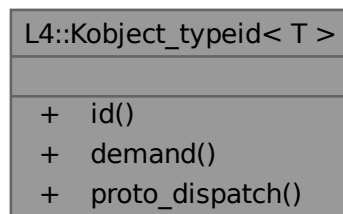
[Meta](#) object for handling access to type information of Kobjects.

```
#include <__typeinfo.h>
```

Inheritance diagram for L4::Kobject_typeid< T >:



Collaboration diagram for L4::Kobject_typeid< T >:



Public Types

- `typedef T::__Kobject_typeid::Demand` [Demand](#)

Data type expressing the static demand of receive buffers in a server.

Static Public Member Functions

- static [Type_info](#) const * [id](#) () noexcept
Get a pointer to the [Kobject](#) type information of T.
- static [Type_info::Demand](#) [demand](#) () noexcept
Get the receive-buffer demand for the server providing the interface T.
- template<typename THIS, typename A1, typename A2>
static int [proto_dispatch](#) (THIS *self, long proto, A1 a1, A2 &a2)
Protocol based server-side dispatch function.

16.183.1 Detailed Description

template<typename T>
struct L4::Kobject_typeid< T >

[Meta](#) object for handling access to type information of Kobjects.

Template Parameters

T	The data type derived from Kobject , usually using Kobject_t .
-------------------	--

Definition at line 610 of file [__typeinfo.h](#).

16.183.2 Member Typedef Documentation

16.183.2.1 Demand

```
template<typename T>
typedef T::__Kobject_typeid::Demand L4::Kobject\_typeid< T >::Demand
```

Data type expressing the static demand of receive buffers in a server.

This information is the combined demand of all base interfaces for T and the buffer demand of T itself. The buffer demand of T is usually specified as the S_DEMAND argument of the [Kobject_t](#) or [Kobject_2t](#) inheritance helpers. S_DEMAND is usually of type [L4::Type_info::Demand_t](#), or [L4::Type_info::Demand_union_t](#).

Definition at line 622 of file [__typeinfo.h](#).

16.183.3 Member Function Documentation

16.183.3.1 demand()

```
template<typename T>
Type\_info::Demand L4::Kobject\_typeid< T >::demand () [inline], [static], [noexcept]
```

Get the receive-buffer demand for the server providing the interface T.

Returns

A demand value describing the minimum receive buffers needed for handling server side requests for interface T.

Definition at line 639 of file [__typeinfo.h](#).

16.183.3.2 id()

```
template<typename T>
Type_info const * L4::Kobject_typeid< T >::id () [inline], [static], [noexcept]
```

Get a pointer to the [Kobject](#) type information of T.

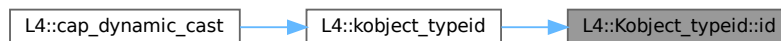
Returns

a pointer to the [Kobject](#) typeinfo of T.

Definition at line 630 of file [__typeinfo.h](#).

Referenced by [L4::kobject_typeid\(\)](#).

Here is the caller graph for this function:



16.183.3.3 proto_dispatch()

```
template<typename T>
template<typename THIS, typename A1, typename A2>
int L4::Kobject_typeid< T >::proto_dispatch (
    THIS * self,
    long proto,
    A1 a1,
    A2 & a2) [inline], [static]
```

Protocol based server-side dispatch function.

Template Parameters

<i>THIS</i>	Data type of the server-side object implementing the interface T.
<i>A1</i>	Data type of second argument for p_dispatch()
<i>A2</i>	Data type of third argument for p_dispatch()

Parameters

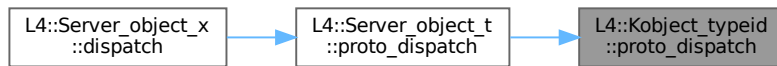
<i>self</i>	The pointer to the server object
<i>proto</i>	The protocol number used by the caller
<i>a1</i>	The second argument passed to self->p_dispatch()
<i>a2</i>	The third argument passed to self->p_dispatch()

This function forwards the call to the overloaded `p_dispatch()` function of `self`. The data type of the first argument for `p_dispatch` is determined by the given protocol number.

Definition at line 660 of file [__typeinfo.h](#).

Referenced by [L4::Server_object_t< IFACE, BASE >::proto_dispatch\(\)](#).

Here is the caller graph for this function:



The documentation for this struct was generated from the following file:

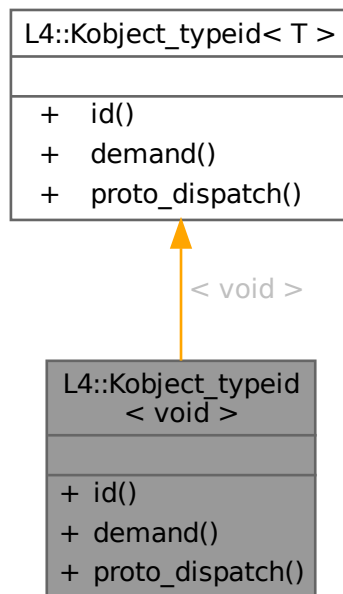
- [l4/sys/__typeinfo.h](#)

16.184 L4::Kobject_typeid< void > Struct Reference

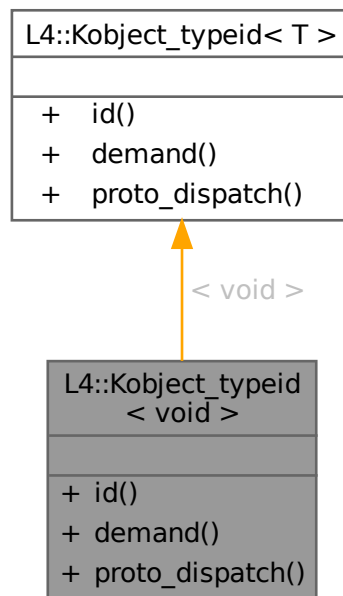
Minimalistic ID for `void` interface.

```
#include <__typeinfo.h>
```

Inheritance diagram for `L4::Kobject_typeid< void >`:



Collaboration diagram for L4::Kobject_typeid< void >:



Static Public Member Functions

- static `Type_info` const * `id` () noexcept
Get a pointer to teh *Kobject* type information of *T*.
- static `Type_info::Demand` `demand` () noexcept
Get the receive-buffer demand for the server providing the interface *T*.
- static int `proto_dispatch` (THIS *self, long proto, A1 a1, A2 &a2)
Protocol based server-side dispatch function.

16.184.1 Detailed Description

Minimalistic ID for `void` interface.

Definition at line 667 of file `__typeinfo.h`.

16.184.2 Member Function Documentation

16.184.2.1 demand()

`Type_info::Demand L4::Kobject_typeid< void >::demand () [inline], [static], [noexcept]`

Get the receive-buffer demand for the server providing the interface *T*.

Returns

A demand value describing the minimum receive buffers needed for handling server side requests for interface *T*.

Definition at line 639 of file `__typeinfo.h`.

16.184.2.2 id()

```
Type_info const * L4::Kobject_typeid< void >::id () [inline], [static], [noexcept]
```

Get a pointer to the [Kobject](#) type information of T.

Returns

a pointer to the [Kobject](#) typeinfo of T.

Definition at line 630 of file [__typeinfo.h](#).

16.184.2.3 proto_dispatch()

```
int L4::Kobject_typeid< void >::proto_dispatch (
    THIS * self,
    long proto,
    A1 a1,
    A2 & a2) [inline], [static]
```

Protocol based server-side dispatch function.

Template Parameters

<i>THIS</i>	Data type of the server-side object implementing the interface T.
<i>A1</i>	Data type of second argument for p_dispatch()
<i>A2</i>	Data type of third argument for p_dispatch()

Parameters

<i>self</i>	The pointer to the server object
<i>proto</i>	The protocol number used by the caller
<i>a1</i>	The second argument passed to self->p_dispatch()
<i>a2</i>	The third argument passed to self->p_dispatch()

This function forwards the call to the overloaded p_dispatch() function of self. The data type of the first argument for p_dispatch is determined by the given protocol number.

Definition at line 660 of file [__typeinfo.h](#).

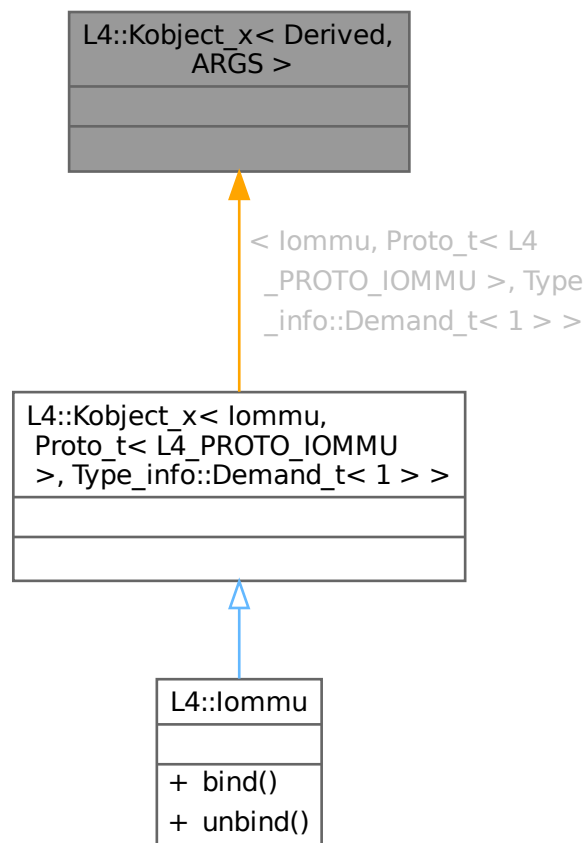
The documentation for this struct was generated from the following file:

- [l4/sys/__typeinfo.h](#)

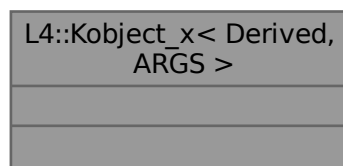
16.185 L4::Kobject_x< Derived, ARGS > Struct Template Reference

Generic [Kobject](#) inheritance template.

Inheritance diagram for L4::Kobject_x< Derived, ARGS >:



Collaboration diagram for L4::Kobject_x< Derived, ARGS >:



16.185.1 Detailed Description

```
template<typename Derived, typename ... ARGS>
struct L4::Kobject_x< Derived, ARGS >
```

Generic [Kobject](#) inheritance template.

Template Parameters

<i>Derived</i>	The class name that derives from Kobject_x .
<i>ARGS</i>	An optional protocol number via L4::Proto_t , followed by an optional server-side requirement passed as L4::Type_info::Demand_t , followed by the list of base classes.

Definition at line 1197 of file [__typeinfo.h](#).

The documentation for this struct was generated from the following file:

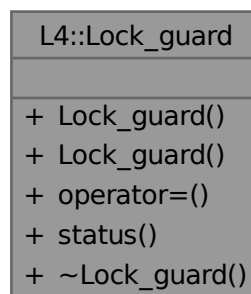
- [l4/sys/__typeinfo.h](#)

16.186 L4::Lock_guard Class Reference

Basic lock guard implementation that prevents forgotten unlocks on exit paths from a method or a block of code.

```
#include <lock_guard.h>
```

Collaboration diagram for L4::Lock_guard:



Public Member Functions

- [Lock_guard](#) (pthread_mutex_t &lock)
Construct the lock guard and lock the associated mutex.
- [Lock_guard](#) ([Lock_guard](#) &&guard)
Move constructor from other lock guard.
- [Lock_guard](#) & [operator=](#) ([Lock_guard](#) &&guard)
Move assignment from other lock guard.
- int [status](#) () const
Return last lock/unlock operation error status.
- [~Lock_guard](#) ()
Lock guard destructor.

16.186.1 Detailed Description

Basic lock guard implementation that prevents forgotten unlocks on exit paths from a method or a block of code.

Targeting `pthread_mutex_t`.

An instance of lock guard cannot be copied, but it can be moved.

The typical usage pattern of the lock guard is:

```
pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;

{
    auto guard = L4Re::Lock_guard(mtx);

    // Correctness check.
    assert(guard.status() == 0);

    // Critical section protected by mtx.

    // The mtx is automatically unlocked when guard goes out of scope.
}
```

Definition at line 44 of file [lock_guard.h](#).

16.186.2 Constructor & Destructor Documentation

16.186.2.1 Lock_guard() [1/2]

```
L4::Lock_guard::Lock_guard (
    pthread_mutex_t & lock) [inline], [explicit]
```

Construct the lock guard and lock the associated mutex.

The error condition of the locking operation can be checked by the [status\(\)](#) method.

Parameters

<i>lock</i>	Associated mutex to be locked.
-------------	--------------------------------

Definition at line 59 of file [lock_guard.h](#).

16.186.2.2 Lock_guard() [2/2]

```
L4::Lock_guard::Lock_guard (
    Lock_guard && guard) [inline]
```

Move constructor from other lock guard.

The mutex associated with the other lock guard is kept locked.

Parameters

<i>guard</i>	Lock guard to be moved.
--------------	-------------------------

Definition at line 71 of file [lock_guard.h](#).

16.186.2.3 ~Lock_guard()

```
L4::Lock_guard::~Lock_guard () [inline]
```

Lock guard destructor.

The associated mutex (if any) is unlocked.

There is no mechanism for indicating any error conditions. However, if the mutex has been previously locked successfully by this class and if the implementation of the mutex behaves according to the POSIX specification, the construction of this class guarantees that the unlock operation does not fail.

Definition at line 126 of file [lock_guard.h](#).

16.186.3 Member Function Documentation

16.186.3.1 operator=()

```
Lock_guard & L4::Lock_guard::operator= (
    Lock_guard && guard) [inline]
```

Move assignment from other lock guard.

The mutex currently associated with this lock guard is unlocked. The mutex associated with the other lock guard is kept locked.

There is no mechanism for indicating any error conditions of the unlocking operation. However, if the mutex has been previously locked successfully by this class and if the implementation of the mutex behaves according to the POSIX specification, the construction of this class guarantees that the unlock operation does not fail.

Parameters

<i>guard</i>	Lock guard to be moved.
--------------	-------------------------

Definition at line 90 of file [lock_guard.h](#).

16.186.3.2 status()

```
int L4::Lock_guard::status () const [inline]
```

Return last lock/unlock operation error status.

Returns

Zero indicating no errors, any other value indicating an error.

Definition at line 110 of file [lock_guard.h](#).

The documentation for this class was generated from the following file:

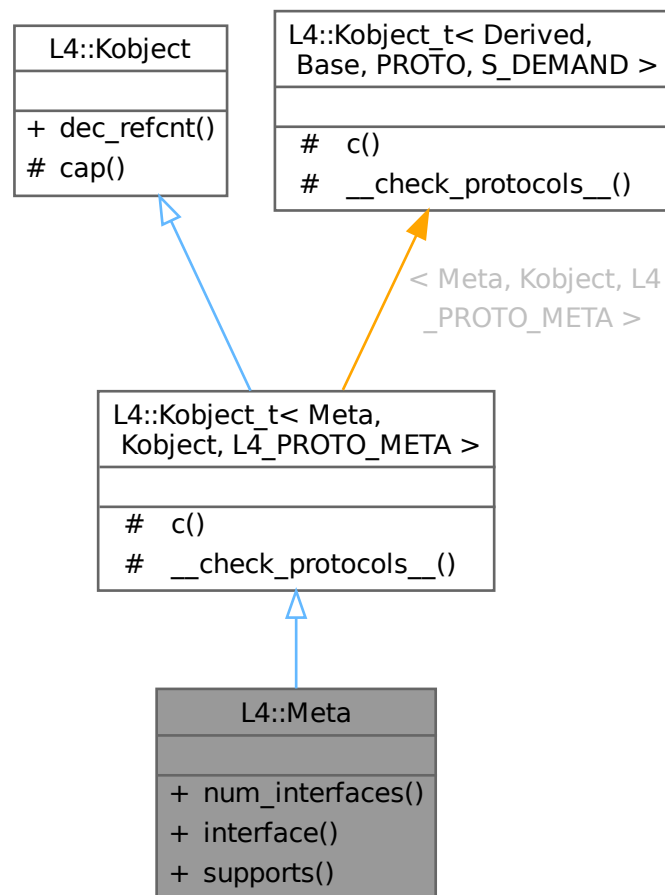
- [l4/cxx/lock_guard.h](#)

16.187 L4::Meta Class Reference

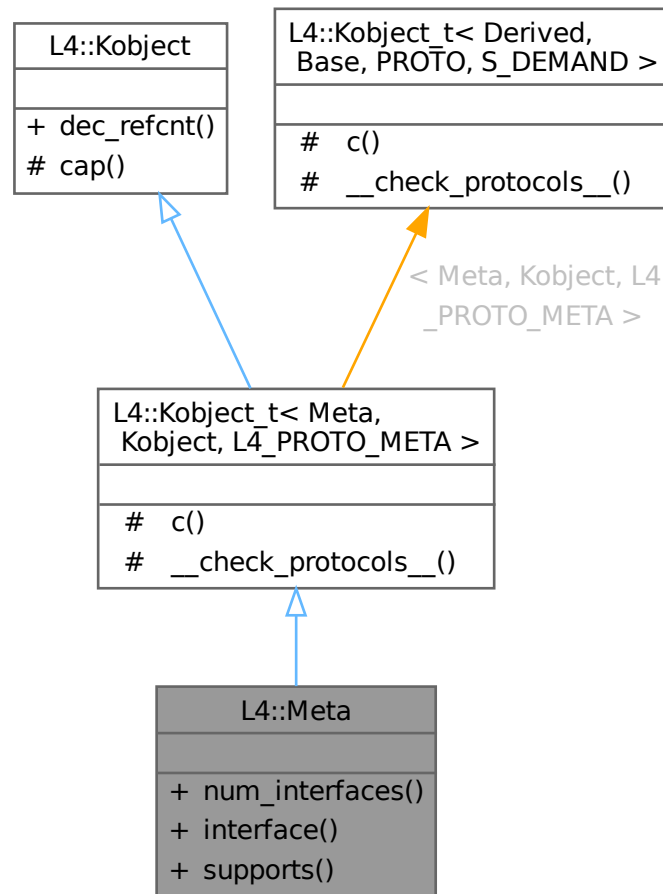
[Meta](#) interface that shall be implemented by each [L4Re](#) object and gives access to the dynamic type information for [L4Re](#) objects.

```
#include <meta>
```

Inheritance diagram for L4::Meta:



Collaboration diagram for L4::Meta:



Public Member Functions

- [l4_msgtag_t num_interfaces \(\)](#)
Get the number of interfaces implemented by this object.
- [l4_msgtag_t interface \(l4_umword_t idx, long *proto, L4::lpc::String< char > *name\)](#)
Get the protocol number that must be used for the interface with the number `idx`.
- [l4_msgtag_t supports \(l4_mword_t protocol\)](#)
Figure out if the object supports the given protocol (number).

Public Member Functions inherited from [L4::Kobject](#)

- [l4_msgtag_t dec_refcnt \(l4_mword_t diff, l4_utcb_t *utcb=l4_utcb\(\)\)](#)
Decrement the in kernel reference counter for the object.

Additional Inherited Members

Protected Types inherited from [L4::Kobject_t< Meta, Kobject, L4_PROTO_META >](#)

- typedef [Meta](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< PROTO, [Meta](#) > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__Iface** >, typename Kobject::__Iface_list > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from [L4::Kobject_t< Meta, Kobject, L4_PROTO_META >](#)

- [L4::Cap< Class > c](#) () const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from [L4::Kobject](#)

- [l4_cap_idx_t](#) [cap](#) () const noexcept
Return capability selector.

Static Protected Member Functions inherited from [L4::Kobject_t< Meta, Kobject, L4_PROTO_META >](#)

- static void **__check_protocols__** () noexcept
Helper to check for protocol conflicts.

16.187.1 Detailed Description

[Meta](#) interface that shall be implemented by each [L4Re](#) object and gives access to the dynamic type information for [L4Re](#) objects.

Definition at line 26 of file [meta](#).

16.187.2 Member Function Documentation

16.187.2.1 interface()

```
l4_msgtag_t L4::Meta::interface (
    l4_umword_t idx,
    long * proto,
    L4::Ipc::String< char > * name)
```

Get the protocol number that must be used for the interface with the number `idx`.

Parameters

	<i>idx</i>	The index of the interface to get the protocol number for. <i>idx</i> must be ≥ 0 and $<$ the return value of num_interfaces() .
out	<i>proto</i>	The protocol number for interface <i>idx</i> .
out	<i>name</i>	The protocol name for interface <i>idx</i> .

Return values

<i>l4_msgtag_t::label()</i> == 0	Successful; see `proto` and `name`.
<i>l4_msgtag_t::label()</i> < 0	Error code.

Referenced by [num_interfaces\(\)](#).

Here is the caller graph for this function:

**16.187.2.2 num_interfaces()**

```
l4_msgtag_t L4::Meta::num_interfaces ()
```

Get the number of interfaces implemented by this object.

Return values

<i>l4_msgtag_t::label()</i> ≥ 0	The number of supported interfaces.
<i>l4_msgtag_t::label()</i> < 0	Error code of the occurred error.

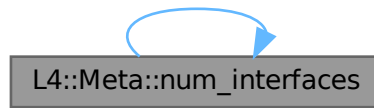
References [interface\(\)](#), and [num_interfaces\(\)](#).

Referenced by [num_interfaces\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.187.2.3 supports()

```
l4_msgtag_t L4::Meta::supports (
    l4_mword_t protocol)
```

Figure out if the object supports the given protocol (number).

Parameters

<i>protocol</i>	The protocol number to check for.
-----------------	-----------------------------------

Return values

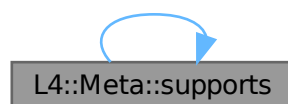
<i>l4_msgtag_t::label()</i> == 1	protocol is supported.
<i>l4_msgtag_t::label()</i> == 0	protocol is not supported.

This method is intended to be used for statically assigned protocol numbers.

References [supports\(\)](#).

Referenced by [supports\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

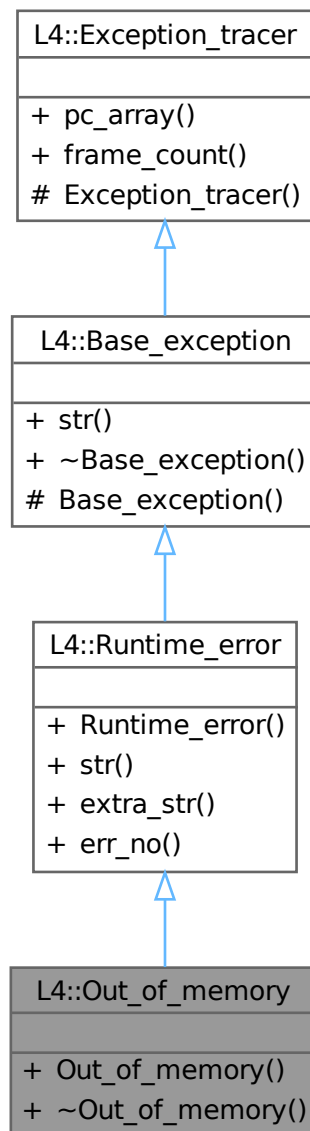
- [l4/sys/meta](#)

16.188 L4::Out_of_memory Class Reference

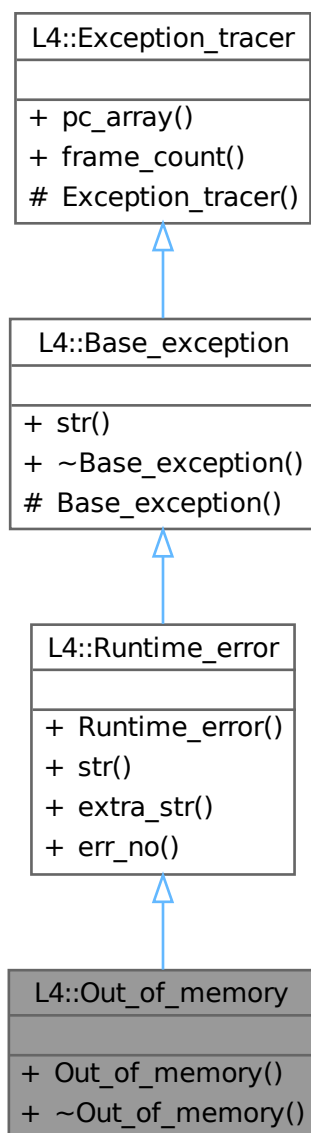
[Exception](#) signalling insufficient memory.

```
#include <l4/cxx/exceptions>
```

Inheritance diagram for L4::Out_of_memory:



Collaboration diagram for L4::Out_of_memory:



Public Member Functions

- **Out_of_memory** (char const *extra="") noexcept
Create an out-of-memory exception.
- **~Out_of_memory** () noexcept
Destruction.

Public Member Functions inherited from [L4::Runtime_error](#)

- [Runtime_error](#) (long err_no, char const *extra=0) noexcept

Create a new [Runtime_error](#).

- char const * **str** () const noexcept override
Return a human readable string for the exception.
- char const * **extra_str** () const
Get the description text for this runtime error.
- long **err_no** () const noexcept
Get the error value for this runtime error.

Public Member Functions inherited from [L4::Base_exception](#)

- virtual ~**Base_exception** () noexcept
Destruction.

Public Member Functions inherited from [L4::Exception_tracer](#)

- void const *const * **pc_array** () const noexcept
Get the array containing the call trace.
- int **frame_count** () const noexcept
Get the number of entries that are valid in the call trace.

Additional Inherited Members

Protected Member Functions inherited from [L4::Base_exception](#)

- **Base_exception** () noexcept
Create a base exception.

Protected Member Functions inherited from [L4::Exception_tracer](#)

- **Exception_tracer** () noexcept
Create a back trace.

16.188.1 Detailed Description

[Exception](#) signalling insufficient memory.

Definition at line 177 of file [exceptions](#).

The documentation for this class was generated from the following file:

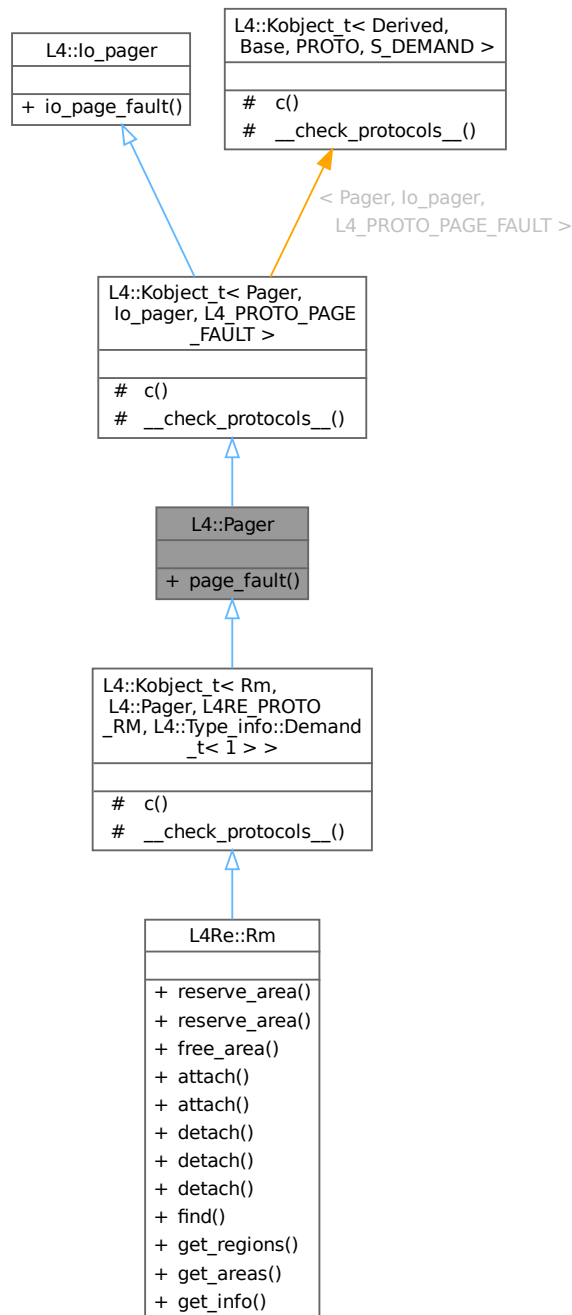
- [l4/cxx/exceptions](#)

16.189 L4::Pager Class Reference

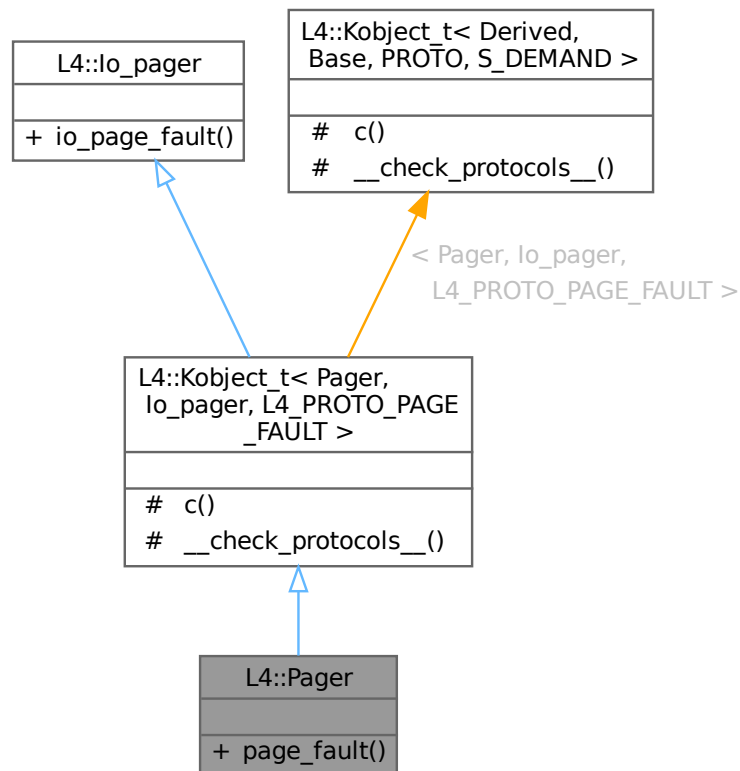
[Pager](#) interface including the [io_pager](#) interface.

```
#include <pager>
```

Inheritance diagram for L4::Pager:



Collaboration diagram for L4::Pager:



Public Member Functions

- [l4_msgtag_t page_fault](#) ([l4_umword_t](#) pfa, [l4_umword_t](#) pc, [L4::lpc::Rcv_fpage](#) rwin, [L4::lpc::Opt<L4::lpc::Snd_fpage & >](#) fp)
Page-fault protocol message.

Public Member Functions inherited from [L4::lo_pager](#)

- [l4_msgtag_t io_page_fault](#) ([l4_fpage_t](#) io_pfa, [l4_umword_t](#) pc, [L4::lpc::Rcv_fpage](#) rwin, [L4::lpc::Opt<L4::lpc::Snd_fpage & >](#) fp)
IO page fault protocol message.

Additional Inherited Members

Protected Types inherited from

[L4::Kobject_t< Pager, lo_pager, L4_PROTO_PAGE_FAULT >](#)

- typedef [Pager](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef [Typeid::Iface< PROTO, Pager >](#) **__Iface**
The interface description for the derived class.
- typedef [Typeid::Merge_list< Typeid::Iface_list< \[__Iface\]\(#\) >, typename lo_pager::__Iface_list >](#) **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from [L4::Kobject_t< Pager, lo_pager, L4_PROTO_PAGE_FAULT >](#)

- [L4::Cap< Class > c \(\)](#) const noexcept
Get the capability to ourselves.

Static Protected Member Functions inherited from [L4::Kobject_t< Pager, lo_pager, L4_PROTO_PAGE_FAULT >](#)

- static void [__check_protocols__ \(\)](#) noexcept
Helper to check for protocol conflicts.

16.189.1 Detailed Description

[Pager](#) interface including the [lo_pager](#) interface.

This class defines the interface for handling page fault IPC. If a thread causes a page fault, the microkernel synthesises a page fault IPC message and sends it to the thread's page fault handler (pager). The pager can then handle the message, for example by establishing a suitable page mapping.

The page fault handler is set with the [L4::Thread::control](#) interface.

Definition at line 87 of file [pager](#).

16.189.2 Member Function Documentation

16.189.2.1 [page_fault\(\)](#)

```
l4_msgtag_t L4::Pager::page_fault (
    l4_umword_t pfa,
    l4_umword_t pc,
    L4::Ipc::Rcv_fpage rwin,
    L4::Ipc::Opt< L4::Ipc::Snd_fpage & > fp)
```

Page-fault protocol message.

Parameters

	<i>pfa</i>	Faulting address including failure reason: bits [0:2].
	<i>pc</i>	Faulting program counter.
	<i>rwin</i>	Receive window for a flexpage mapping resolving the page fault.
out	<i>fp</i>	Optional: flexpage descriptor to send to the task raising the page fault.

Returns

System call message tag; use [l4_error\(\)](#) to check for errors.

Page-fault messages are usually generated by the kernel and need to be handled by an appropriate handler function, potentially filling in `fp` for the reply.

`pfa` encoding is as shown:

[63/31 .. 3]	2	1	0
PFA	X	W	r

- **PFA** Bits 63/31..3 of `pfa` are the page fault address bits 63/31 to 3, bits 2..0 are masked.
- **X** Bit 2 of `pfa` if set, indicates a page fault during instruction fetch. Note, this bit is implementation-defined and might always be clear. Therefore, if this bit is clear it does not imply that the page fault is not due to an instruction fetch.
- **W** Bit 1 of `pfa` is set to 1 for a page fault due to a write operation.
- **r** Bit0: reserved, undefined.

The documentation for this class was generated from the following file:

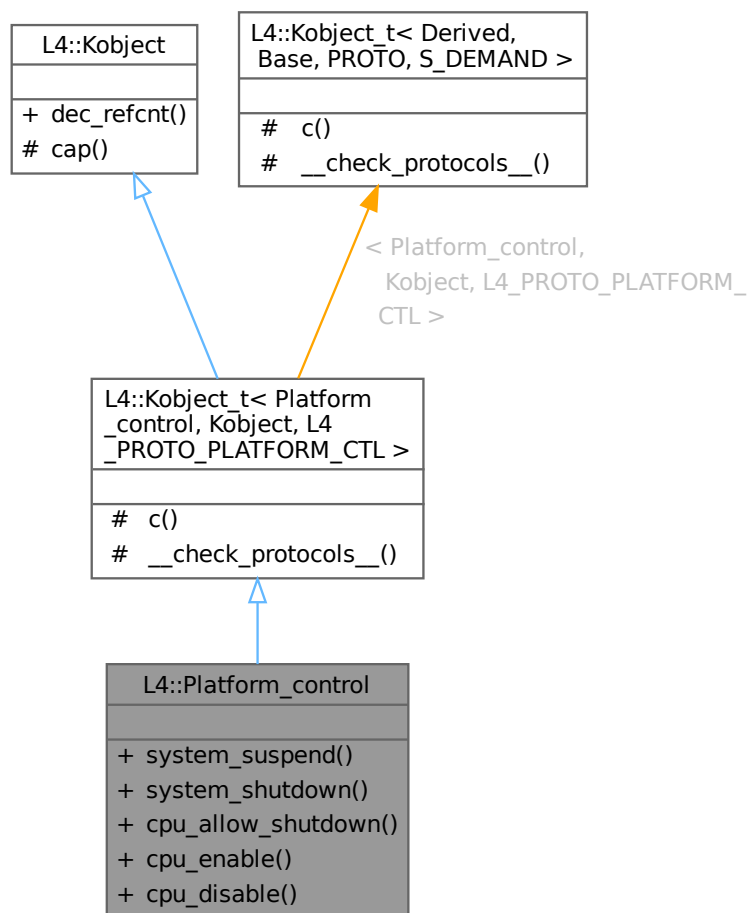
- [l4/sys/pager](#)

16.190 L4::Platform_control Class Reference

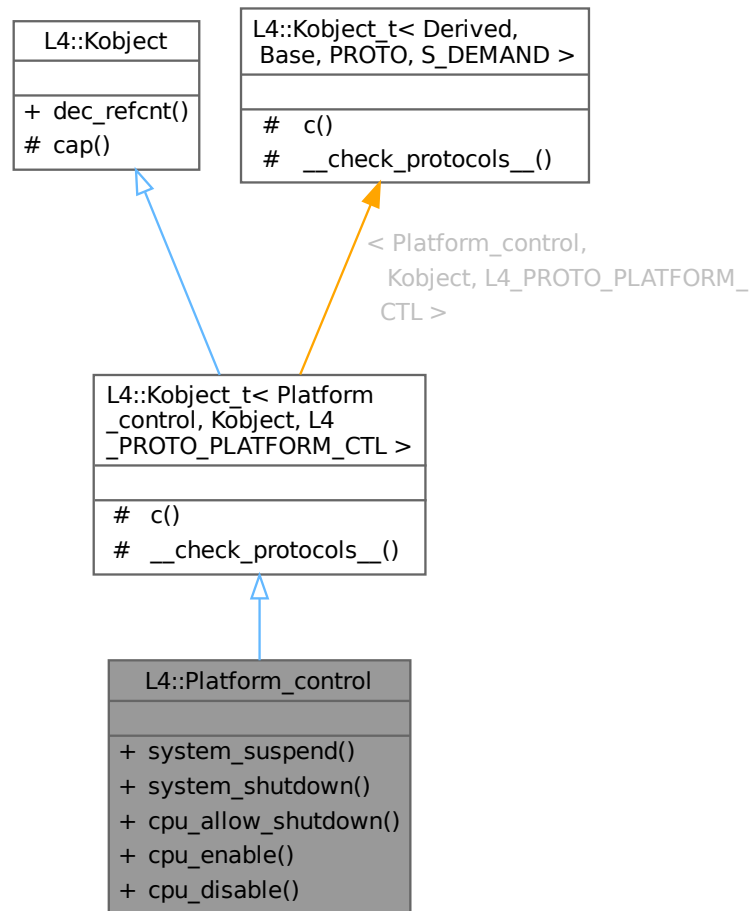
[L4](#) C++ interface for controlling platform-wide properties, see [Platform Control C API](#) for the C interface.

```
#include <platform_control>
```

Inheritance diagram for L4::Platform_control:



Collaboration diagram for L4::Platform_control:



Public Member Functions

- [l4_msgtag_t system_suspend](#) ([l4_umword_t](#) extras)
Enter suspend to RAM.
- [l4_msgtag_t system_shutdown](#) ([l4_umword_t](#) reboot)
Shutdown/Reboot the system.
- [l4_msgtag_t cpu_allow_shutdown](#) ([l4_umword_t](#) phys_id, [l4_umword_t](#) enable)
Allow CPU shutdown.
- [l4_msgtag_t cpu_enable](#) ([l4_umword_t](#) phys_id)
Enable an offline CPU.
- [l4_msgtag_t cpu_disable](#) ([l4_umword_t](#) phys_id)
Disable an online CPU.

Public Member Functions inherited from [L4::Kobject](#)

- [l4_msgtag_t dec_refcnt](#) ([l4_mword_t](#) diff, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#))
Decrement the in kernel reference counter for the object.

Additional Inherited Members

Protected Types inherited from

[L4::Kobject_t< Platform_control, Kobject, L4_PROTO_PLATFORM_CTL >](#)

- typedef [Platform_control](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< [PROTO](#), [Platform_control](#) > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< [__Iface](#) >, typename Kobject::__Iface_list > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from

[L4::Kobject_t< Platform_control, Kobject, L4_PROTO_PLATFORM_CTL >](#)

- [L4::Cap< Class > c](#) () const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from [L4::Kobject](#)

- [l4_cap_idx_t cap](#) () const noexcept
Return capability selector.

Static Protected Member Functions inherited from

[L4::Kobject_t< Platform_control, Kobject, L4_PROTO_PLATFORM_CTL >](#)

- static void [__check_protocols__](#) () noexcept
Helper to check for protocol conflicts.

16.190.1 Detailed Description

[L4](#) C++ interface for controlling platform-wide properties, see [Platform Control C API](#) for the C interface.

Add

```
#include <l4/sys/platform_control>
```

to your code to use the platform control functions. The API allows a client to suspend, reboot or shutdown the system.

For the C interface refer to the [Platform Control C API](#).

Definition at line 36 of file [platform_control](#).

16.190.2 Member Function Documentation

16.190.2.1 [cpu_allow_shutdown\(\)](#)

```
l4\_msgtag\_t L4::Platform_control::cpu_allow_shutdown (
    l4\_umword\_t phys_id,
    l4\_umword\_t enable)
```

Allow CPU shutdown.

Parameters

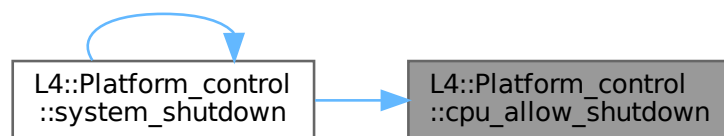
<i>phys↔ _id</i>	Physical CPU id of CPU (e.g. local APIC id) to disable.
<i>enable</i>	Allow shutdown when 1, disallow when 0.

Sets or unsets a hint that a CPU that is not currently used may be powered down.

References [L4_PLATFORM_CTL_CPU_ENABLE_OP](#).

Referenced by [system_shutdown\(\)](#).

Here is the caller graph for this function:



16.190.2.2 `cpu_disable()`

```
l4_msgtag_t L4::Platform_control::cpu_disable (
    l4_umword_t phys_id)
```

Disable an online CPU.

Parameters

<i>phys↔ _id</i>	Physical CPU id of CPU (e.g. local APIC id) to disable.
----------------------	---

Returns

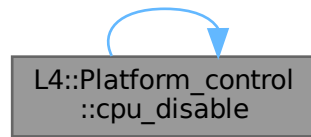
System call message tag

This function is currently only supported on the ARM EXYNOS platform.

References [cpu_disable\(\)](#).

Referenced by [cpu_disable\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.190.2.3 `cpu_enable()`

```
l4_msgtag_t L4::Platform_control::cpu_enable (  
    l4_umword_t phys_id)
```

Enable an offline CPU.

Parameters

<i>phys_id</i>	Physical CPU id of CPU (e.g. local APIC id) to enable.
----------------	--

Returns

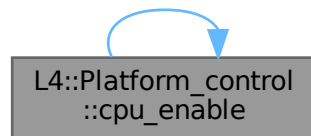
System call message tag

This function is currently only supported on the ARM EXYNOS platform.

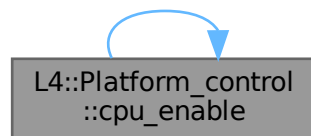
References [cpu_enable\(\)](#), and [L4_PLATFORM_CTL_CPU_DISABLE_OP](#).

Referenced by [cpu_enable\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.190.2.4 system_shutdown()

```
l4_msgtag_t L4::Platform_control::system_shutdown (
    l4_umword_t reboot)
```

Shutdown/Reboot the system.

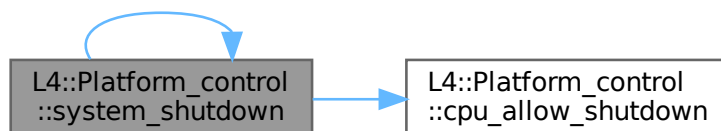
Parameters

<i>reboot</i>	1 for reboot, 0 for power off
---------------	-------------------------------

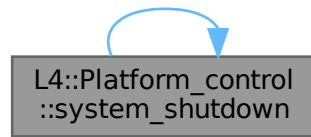
References [cpu_allow_shutdown\(\)](#), [L4_PLATFORM_CTL_CPU_ALLOW_SHUTDOWN_OP](#), and [system_shutdown\(\)](#).

Referenced by [system_shutdown\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.190.2.5 system_suspend()

```
l4_msgtag_t L4::Platform_control::system_suspend (  
    l4_umword_t extras)
```

Enter suspend to RAM.

Precondition

Must only be invoked on the boot CPU. Furthermore it must be ensured that the invoking thread is not migrated to a different CPU during the suspend.

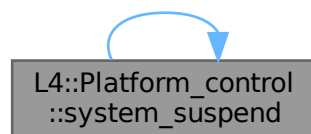
Parameters

<i>extras</i>	Some extra platform-specific information needed to enter suspend to RAM. On x86 platforms and when using the Platform_control object provided by Fiasco, the value defines the sleep state. The sleep states are defined in the ACPI table. Other platforms as well as lo's Platform_control object don't make use of this value at the moment.
---------------	---

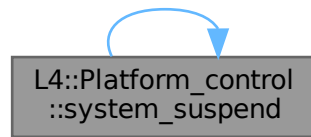
References [L4_PLATFORM_CTL_SYS_SHUTDOWN_OP](#), and [system_suspend\(\)](#).

Referenced by [system_suspend\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

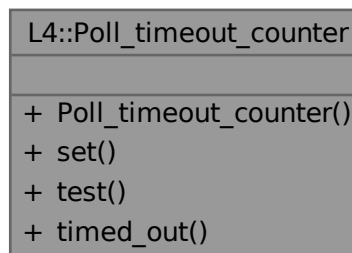
- [l4/sys/platform_control](#)

16.191 L4::Poll_timeout_counter Class Reference

Evaluate an expression for a maximum number of times.

```
#include <poll_timeout_counter.h>
```

Collaboration diagram for L4::Poll_timeout_counter:



Public Member Functions

- [Poll_timeout_counter](#) (unsigned counter_val)
Constructor.
- void [set](#) (unsigned counter_val)
Set the counter to a certain value.
- bool **test** (bool expression=true)
Evaluate the expression for a maximum number of times.
- bool [timed_out](#) () const
Indicator if the maximum number of tests was required.

16.191.1 Detailed Description

Evaluate an expression for a maximum number of times.

A typical use case is testing for a bit change in a hardware register for a maximum number of times (polling). For example:

```
Mmio_register_block regs;
Poll_timeout_counter i(3000000);
while (i.test(!(regs.read<14_uint32_t>(0x04) & 1)))
;
```

The following usage is **wrong**:

```
...
Poll_timeout_counter i(3000000);
while (!i.test((regs.read<14_uint32_t>(0x04) & 1)))
;
```

This loop would never terminate if the hardware register doesn't change!

Definition at line 34 of file [poll_timeout_counter.h](#).

16.191.2 Constructor & Destructor Documentation

16.191.2.1 Poll_timeout_counter()

```
L4::Poll_timeout_counter::Poll_timeout_counter (
    unsigned counter_val) [inline]
```

Constructor.

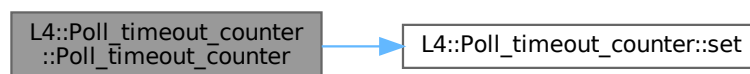
Parameters

<i>counter_val</i>	Maximum number of times to repeat the test.
--------------------	---

Definition at line 42 of file [poll_timeout_counter.h](#).

References [set\(\)](#).

Here is the call graph for this function:



16.191.3 Member Function Documentation

16.191.3.1 set()

```
void L4::Poll_timeout_counter::set (
    unsigned counter_val) [inline]
```

Set the counter to a certain value.

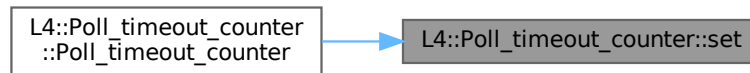
Parameters

<i>counter_val</i>	New counter value for maximum number of times to repeat the test.
--------------------	---

Definition at line 53 of file [poll_timeout_counter.h](#).

Referenced by [Poll_timeout_counter\(\)](#).

Here is the caller graph for this function:



16.191.3.2 timed_out()

```
bool L4::Poll_timeout_counter::timed_out () const [inline]
```

Indicator if the maximum number of tests was required.

Return values

<i>true, if</i>	the maximum number of tests was required or if the counter was initialized to zero.
-----------------	---

Definition at line 81 of file [poll_timeout_counter.h](#).

The documentation for this class was generated from the following file:

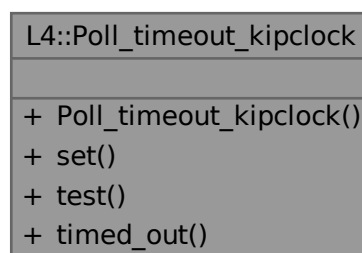
- pkg/drivers-frst/include/poll_timeout_counter.h

16.192 L4::Poll_timeout_kipclock Class Reference

A polling timeout based on the [L4Re](#) clock.

```
#include <poll_timeout_kipclock>
```

Collaboration diagram for L4::Poll_timeout_kipclock:



Public Member Functions

- [Poll_timeout_kipclock](#) (unsigned poll_time_us)
Initialise relative timeout in microseconds.
- void [set](#) (unsigned poll_time_us)
(Re-)Set relative timeout in microseconds
- bool [test](#) (bool expression=true)
Test whether timeout has expired.
- bool [timed_out](#) () const
Query whether timeout has expired.

16.192.1 Detailed Description

A polling timeout based on the [L4Re](#) clock.

This class allows to conveniently add a timeout to a polling loop.

The original

```
while (device.read(State) & Busy)
;
```

is converted to

```
Poll_timeout_kipclock timeout(10000);
while (timeout.test(device.read(State) & Busy))
;
if (timeout.timed_out())
    printf("ERROR: Device does not respond.\n");
```

Definition at line 35 of file [poll_timeout_kipclock](#).

16.192.2 Constructor & Destructor Documentation

16.192.2.1 Poll_timeout_kipclock()

```
L4::Poll_timeout_kipclock::Poll_timeout_kipclock (
    unsigned poll_time_us) [inline]
```

Initialise relative timeout in microseconds.

Parameters

<i>poll_time_us</i>	Polling timeout in microseconds.
---------------------	----------------------------------

Definition at line 42 of file [poll_timeout_kipclock](#).

References [set\(\)](#).

Here is the call graph for this function:



16.192.3 Member Function Documentation

16.192.3.1 set()

```
void L4::Poll_timeout_kipclock::set (
    unsigned poll_time_us) [inline]
```

(Re-)Set relative timeout in microseconds

Parameters

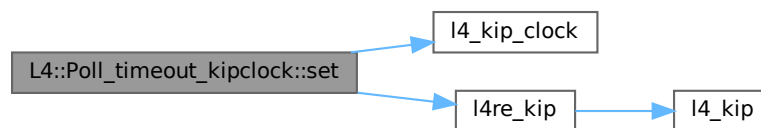
<i>poll_time_us</i>	Polling timeout in microseconds.
---------------------	----------------------------------

Definition at line 51 of file [poll_timeout_kipclock](#).

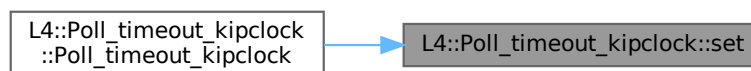
References [l4_kip_clock\(\)](#), and [l4re_kip\(\)](#).

Referenced by [Poll_timeout_kipclock\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.192.3.2 test()

```
bool L4::Poll_timeout_kipclock::test (
    bool expression = true) [inline]
```

Test whether timeout has expired.

Parameters

<i>expression</i>	Optional expression.
-------------------	----------------------

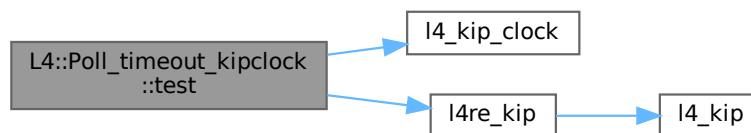
Return values

<i>false</i>	The timeout has expired or the given expression returned false.
<i>true</i>	The timeout has not expired and the optionally given expression returns true.

Definition at line 65 of file [poll_timeout_kipclock](#).

References [l4_kip_clock\(\)](#), and [l4re_kip\(\)](#).

Here is the call graph for this function:

**16.192.3.3 timed_out()**

```
bool L4::Poll_timeout_kipclock::timed_out () const [inline]
```

Query whether timeout has expired.

Returns

Expiry state of timeout

Definition at line 77 of file [poll_timeout_kipclock](#).

The documentation for this class was generated from the following file:

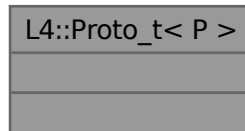
- `l4/re/util/poll_timeout_kipclock`

16.193 L4::Proto_t< P > Struct Template Reference

Data type for defining protocol numbers.

```
#include <__typeinfo.h>
```

Collaboration diagram for L4::Proto_t< P >:



16.193.1 Detailed Description

```
template<long P = PROTO_EMPTY>
struct L4::Proto_t< P >
```

Data type for defining protocol numbers.

Template Parameters

<i>P</i>	The protocol number itself
----------	----------------------------

This type must be used when specifying a protocol number with [Kobject_x](#).

Definition at line [1181](#) of file [__typeinfo.h](#).

The documentation for this struct was generated from the following file:

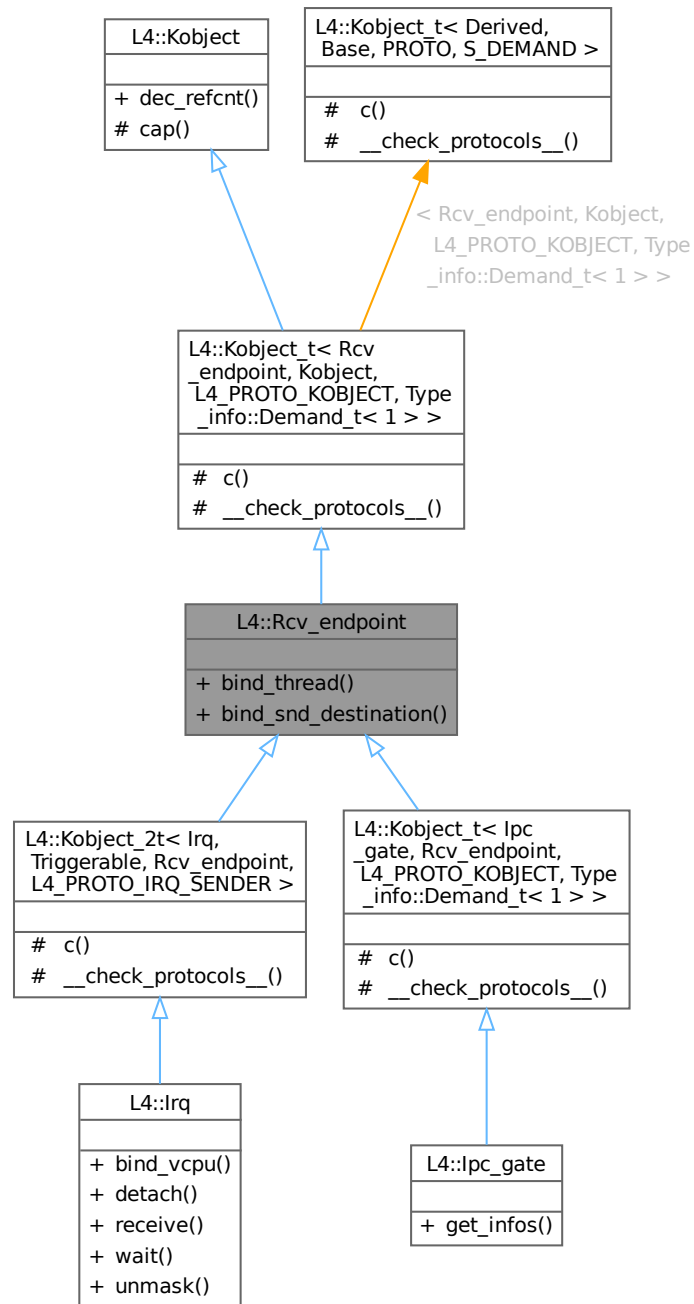
- [l4/sys/__typeinfo.h](#)

16.194 L4::Rcv_endpoint Class Reference

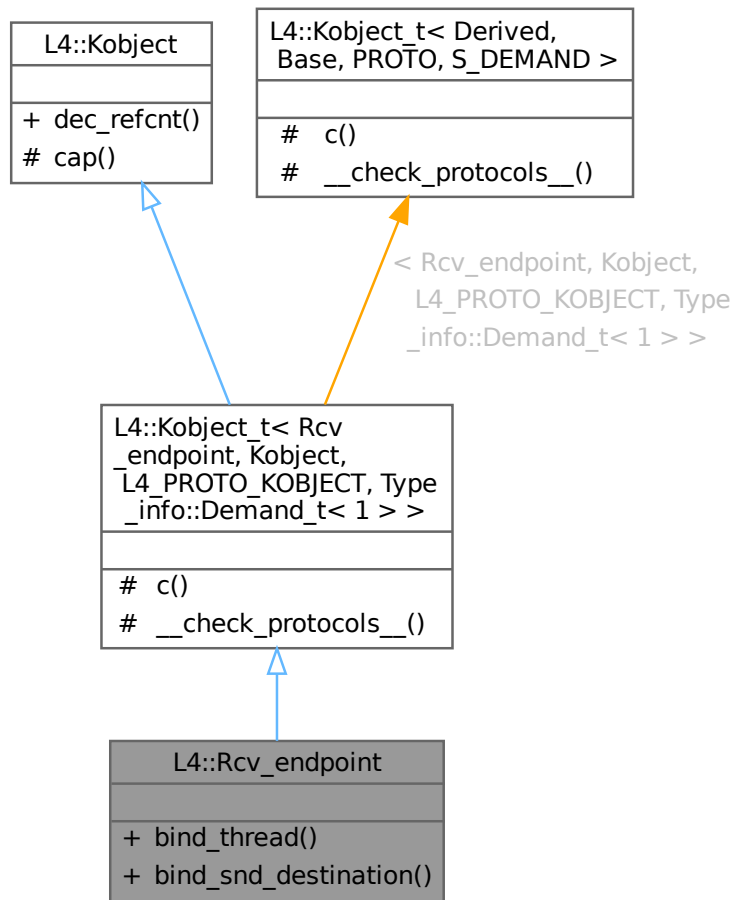
Interface for kernel objects that allow to receive IPC from them.

```
#include <rcv_endpoint>
```


Inheritance diagram for L4::Rcv_endpoint:



Collaboration diagram for L4::Rcv_endpoint:



Public Member Functions

- `l4_msgtag_t bind_thread (lpc::Cap< Thread > t, l4_umword_t label)`
Bind the IPC receive endpoint to a thread.
- `l4_msgtag_t bind_snd_destination (Cap< Snd_destination > snd_dst, l4_umword_t label)`
Bind a send destination (a thread or thread group) to an IPC receive endpoint.

Public Member Functions inherited from L4::Kobject

- `l4_msgtag_t dec_refcnt (l4_mword_t diff, l4_utcb_t *utcb=l4_utcb())`
Decrement the in kernel reference counter for the object.

Additional Inherited Members

Protected Types inherited from

[L4::Kobject_t< Rcv_endpoint, Kobject, L4_PROTO_KOBJECT, Type_info::Demand_t< 1 > >](#)

- typedef [Rcv_endpoint](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< [PROTO](#), [Rcv_endpoint](#) > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__Iface** >, typename Kobject::__Iface_list > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from

[L4::Kobject_t< Rcv_endpoint, Kobject, L4_PROTO_KOBJECT, Type_info::Demand_t< 1 > >](#)

- [L4::Cap< Class > c](#) () const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from [L4::Kobject](#)

- [l4_cap_idx_t cap](#) () const noexcept
Return capability selector.

Static Protected Member Functions inherited from

[L4::Kobject_t< Rcv_endpoint, Kobject, L4_PROTO_KOBJECT, Type_info::Demand_t< 1 > >](#)

- static void [__check_protocols__](#) () noexcept
Helper to check for protocol conflicts.

16.194.1 Detailed Description

Interface for kernel objects that allow to receive IPC from them.

Such an object is for example an [lpc_gate](#) (with server rights) or an [lrq](#). Those objects can be bound to a thread that shall receive IPC from these objects via [bind_thread\(\)](#) or [bind_snd_destination\(\)](#).

Definition at line 30 of file [rcv_endpoint](#).

16.194.2 Member Function Documentation

16.194.2.1 [bind_snd_destination\(\)](#)

```
l4_msgtag_t L4::Rcv_endpoint::bind_snd_destination (
    Cap< Snd_destination > snd_dst,
    l4_umword_t label) [inline]
```

Bind a send destination (a thread or thread group) to an IPC receive endpoint.

Parameters

<i>snd_dst</i>	Snd_destination object (a thread or thread group) that shall be bound to this receive endpoint. See bind_thread() and bind_snd_destination() for binding a thread or thread group object.
<i>label</i>	Label to assign to <code>this</code> receive endpoint. For IPC gates, the two least significant bits must be set to zero.

Returns

Syscall return tag containing one of the following return codes.

Return values

<i>L4_EOK</i>	Operation successful.
<i>-L4_EINVAL</i>	<code>snd_dst</code> is not a thread or thread group object or other arguments were malformed.
<i>-L4_EPERM</i>	No L4_CAP_FPAGE_S right on <code>snd_dst</code> or the capability used to invoke this operation.

Precondition

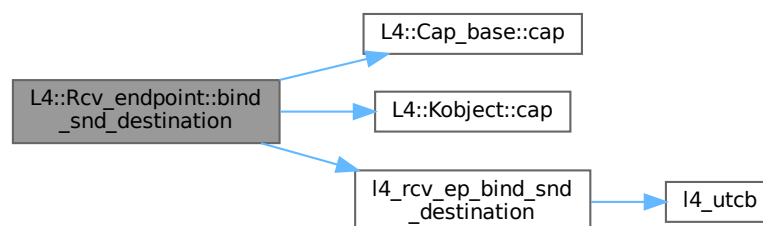
If this operation is invoked using an IPC gate capability without the [L4_FPAGE_C_IPCGATE_SVR](#) right, the kernel will not perform the operation. Instead, the underlying IPC message will be forwarded to the thread bound to the IPC gate, blocking the caller if no thread or thread group is bound yet.

The specified `label` is passed to the receiver of the incoming IPC. It is possible to re-bind a receive endpoint to the same or a different thread or thread group. In this case, IPC already in flight will be delivered with the old label to the previously bound thread or thread group unless [L4::Thread::modify_senders\(\)](#) is used to change these labels.

Definition at line 101 of file [rcv_endpoint](#).

References [L4::Cap_base::cap\(\)](#), [L4::Kobject::cap\(\)](#), and [l4_rcv_ep_bind_snd_destination\(\)](#).

Here is the call graph for this function:



16.194.2.2 bind_thread()

```

l4_msgtag_t L4::Rcv_endpoint::bind_thread (
    Ipc::Cap< Thread > t,
    l4_umword_t label)

```

Bind the IPC receive endpoint to a thread.

Parameters

<i>t</i>	Thread object this receive endpoint shall be bound to.
<i>label</i>	Label to assign to <code>this</code> receive endpoint. For IPC gates, the two least significant bits must be set to zero.

Returns

Syscall return tag containing one of the following return codes.

Return values

<i>L4_EOK</i>	Operation successful.
<i>-L4_EINVAL</i>	<code>t</code> is not a thread object or other arguments were malformed.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.

Precondition

The invoked capability and the capability `t` both must have the permission [L4_CAP_FPAGE_S](#).

Deprecated Use [bind_snd_destination\(\)](#) instead.

Precondition

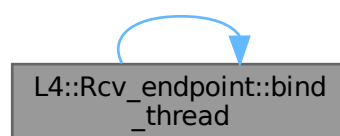
If this operation is invoked using an IPC gate capability without the [L4_FPAGE_C_IPCGATE_SVR](#) right, the kernel will not perform the operation. Instead, the underlying IPC message will be forwarded to the thread the IPC gate is bound to, blocking the caller if the IPC gate was not bound yet.

The specified `label` is passed to the receiver of the incoming IPC. It is possible to re-bind a receive endpoint to the same or a different thread. In this case, IPC already in flight will be delivered with the old label to the previously bound thread unless [L4::Thread::modify_senders\(\)](#) is used to change these labels.

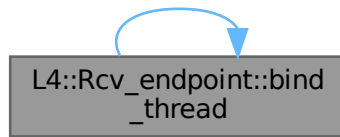
References [bind_thread\(\)](#).

Referenced by [bind_thread\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

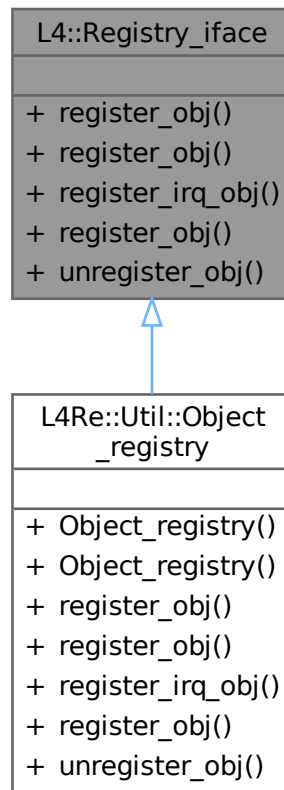
- [l4/sys/rcv_endpoint](#)

16.195 L4::Registry_iface Class Reference

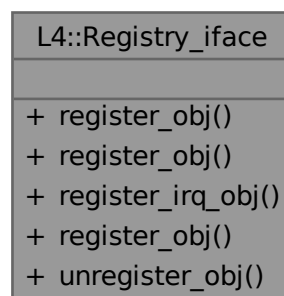
Abstract interface for object registries.

```
#include <ipc_epiface>
```

Inheritance diagram for L4::Registry_iface:



Collaboration diagram for L4::Registry_iface:



Public Member Functions

- virtual [L4::Cap](#)< void > [register_obj](#) ([L4::Epiface](#) *o, char const *service)=0

- Register an [L4::Epiface](#) for an IPC gate available in the applications environment under the name *service*.
virtual [L4::Cap](#)< void > [register_obj](#) ([L4::Epiface](#) *o)=0
Register o as server-side object for synchronous RPC.
- virtual [L4::Cap](#)< [L4::Irq](#) > [register_irq_obj](#) ([L4::Epiface](#) *o)=0
Register o as server-side object for asynchronous IRQs.
- virtual [L4::Cap](#)< [L4::Rcv_endpoint](#) > [register_obj](#) ([L4::Epiface](#) *o, [L4::Cap](#)< [L4::Rcv_endpoint](#) > ep)=0
Register o as server-side object for a pre-allocated capability.
- virtual void [unregister_obj](#) ([L4::Epiface](#) *o, bool unmap=true)=0
Unregister the given object o from the server.

16.195.1 Detailed Description

Abstract interface for object registries.

An object registry allows to register [L4::Epiface](#) objects at a server loop either for synchronous RPC messages or for asynchronous IRQ messages.

Definition at line 322 of file [ipc_epiface](#).

16.195.2 Member Function Documentation

16.195.2.1 [register_irq_obj\(\)](#)

```
virtual L4::Cap< L4::Irq > L4::Registry\_iface::register\_irq\_obj (  
    L4::Epiface * o) [pure virtual]
```

Register o as server-side object for asynchronous IRQs.

Parameters

<i>o</i>	Pointer to an Epiface object that shall be registered as server-side object for IRQs.
----------	---

Return values

L4::Cap < L4::Irq >	Capability to a new IRQ object on success.
L4::Cap < L4::Irq >:: Invalid	The allocation of the IRQ has failed.

After successful registration `o->obj_cap()` will be the capability of the allocated IRQ object.

The function may allocate a capability slot for the object. In that case [unregister_obj\(\)](#) is responsible for freeing the slot as well.

Implemented in [L4Re::Util::Object_registry](#).

16.195.2.2 [register_obj\(\)](#) [1/3]

```
virtual L4::Cap< void > L4::Registry\_iface::register\_obj (  
    L4::Epiface * o) [pure virtual]
```

Register o as server-side object for synchronous RPC.

Parameters

<i>o</i>	Pointer to an Epiface object that shall be registered as server-side object for RPC.
----------	--

Return values

L4::Cap<void>	A valid capability to a new IPC gate.
L4::Cap<void>::Invalid	The allocation of the IPC gate has failed.

After successful registration `o->obj_cap()` will be the capability of the allocated IPC gate.

The function may allocate a capability slot for the object. In that case [unregister_obj\(\)](#) is responsible for freeing the slot as well.

Implemented in [L4Re::Util::Object_registry](#).

16.195.2.3 register_obj() [2/3]

```
virtual L4::Cap< void > L4::Registry_iface::register_obj (
    L4::Epiface * o,
    char const * service) [pure virtual]
```

Register an [L4::Epiface](#) for an IPC gate available in the applications environment under the name `service`.

Parameters

<i>o</i>	Pointer to an Epiface object that shall be registered.
<i>service</i>	Name of the capability that shall be used to connect <i>o</i> to as a server-side object.

Return values

L4::Cap<void>	The capability known as <code>service</code> on success.
L4::Cap<void>::Invalid	No capability with the given name found.

After a successful call to this function `o->obj_cap()` is equal to the capability in the environment with the name given by `service`.

Implemented in [L4Re::Util::Object_registry](#).

16.195.2.4 register_obj() [3/3]

```
virtual L4::Cap< L4::Rcv\_endpoint > L4::Registry_iface::register_obj (
    L4::Epiface * o,
    L4::Cap< L4::Rcv\_endpoint > ep) [pure virtual]
```

Register *o* as server-side object for a pre-allocated capability.

Parameters

<i>o</i>	Pointer to an Epiface object that shall be registered as server-side object.
<i>ep</i>	Capability to an already allocated capability where <i>o</i> shall be attached as server-side handler. The capability may point to an IPC gate or an IRQ.

Return values

L4::Cap<L4::Rcv_endpoint>	Capability <i>ep</i> on success.
L4::Cap<L4::Rcv_endpoint>::Invalid	The IRQ attach operation has failed.

After successful registration `o->obj_cap()` will be equal to *ep*.

Implemented in [L4Re::Util::Object_registry](#).

16.195.2.5 unregister_obj()

```
virtual void L4::Registry_iface::unregister_obj (
    L4::Epiface * o,
    bool unmap = true) [pure virtual]
```

Unregister the given object *o* from the server.

Parameters

<i>o</i>	Pointer to the Epiface object that shall be unregistered. The object must have been registered with any of the register methods if Registry_iface .
<i>unmap</i>	If true the capability <code>o->obj_cap()</code> shall be unmapped from the local object space.

The function always unmaps and frees the capability if it was allocated by either [Registry_iface::register_irq_obj\(L4::Epiface *\)](#), or by [Registry_iface::register_obj\(L4::Epiface *\)](#).

Implemented in [L4Re::Util::Object_registry](#).

The documentation for this class was generated from the following file:

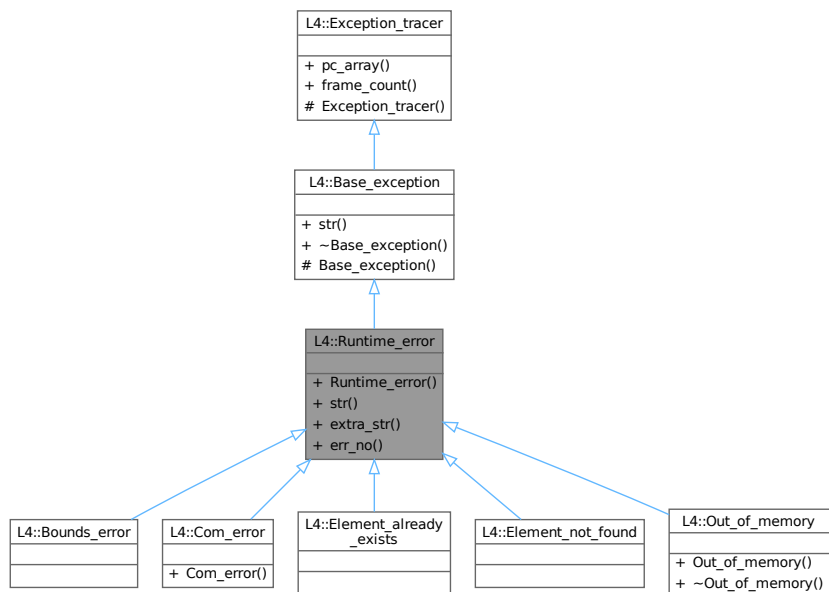
- `l4/sys/cxx/ipc_epiface`

16.196 L4::Runtime_error Class Reference

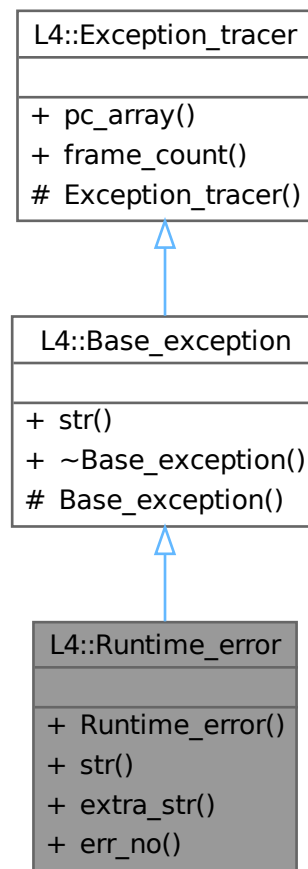
[Exception](#) for an abstract runtime error.

```
#include <l4/cxx/exceptions>
```

Inheritance diagram for L4::Runtime_error:



Collaboration diagram for L4::Runtime_error:



Public Member Functions

- [Runtime_error](#) (long [err_no](#), char const *extra=0) noexcept
Create a new [Runtime_error](#).
- char const * [str](#) () const noexcept override
Return a human readable string for the exception.
- char const * [extra_str](#) () const
Get the description text for this runtime error.
- long [err_no](#) () const noexcept
Get the error value for this runtime error.

Public Member Functions inherited from [L4::Base_exception](#)

- virtual `~Base_exception` () noexcept
Destruction.

Public Member Functions inherited from [L4::Exception_tracer](#)

- `void const *const * pc_array () const noexcept`
Get the array containing the call trace.
- `int frame_count () const noexcept`
Get the number of entries that are valid in the call trace.

Additional Inherited Members

Protected Member Functions inherited from [L4::Base_exception](#)

- `Base_exception () noexcept`
Create a base exception.

Protected Member Functions inherited from [L4::Exception_tracer](#)

- `Exception_tracer () noexcept`
Create a back trace.

16.196.1 Detailed Description

[Exception](#) for an abstract runtime error.

This is the base class for a set of exceptions that cover all errors that have a C error value (see [l4_error_code_t](#)).

Definition at line [128](#) of file [exceptions](#).

16.196.2 Constructor & Destructor Documentation

16.196.2.1 Runtime_error()

```
L4::Runtime_error::Runtime_error (
    long err_no,
    char const * extra = 0) [inline], [explicit], [noexcept]
```

Create a new [Runtime_error](#).

Parameters

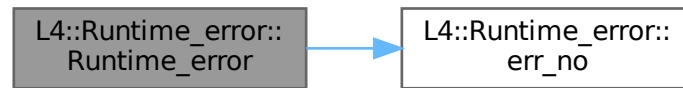
<i>err_no</i>	Error value for this runtime error.
<i>extra</i>	Description of what happened when the error occurred.

Definition at line [141](#) of file [exceptions](#).

References [err_no\(\)](#).

Referenced by [L4::Out_of_memory::Out_of_memory\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.196.3 Member Function Documentation

16.196.3.1 `err_no()`

```
long L4::Runtime_error::err_no () const [inline], [noexcept]
```

Get the error value for this runtime error.

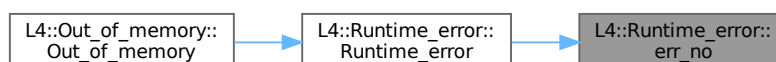
Returns

Error value.

Definition at line 170 of file [exceptions](#).

Referenced by [Runtime_error\(\)](#).

Here is the caller graph for this function:



16.196.3.2 extra_str()

```
char const * L4::Runtime_error::extra_str () const [inline]
```

Get the description text for this runtime error.

Returns

Pointer to the description string.

Definition at line 162 of file [exceptions](#).

The documentation for this class was generated from the following file:

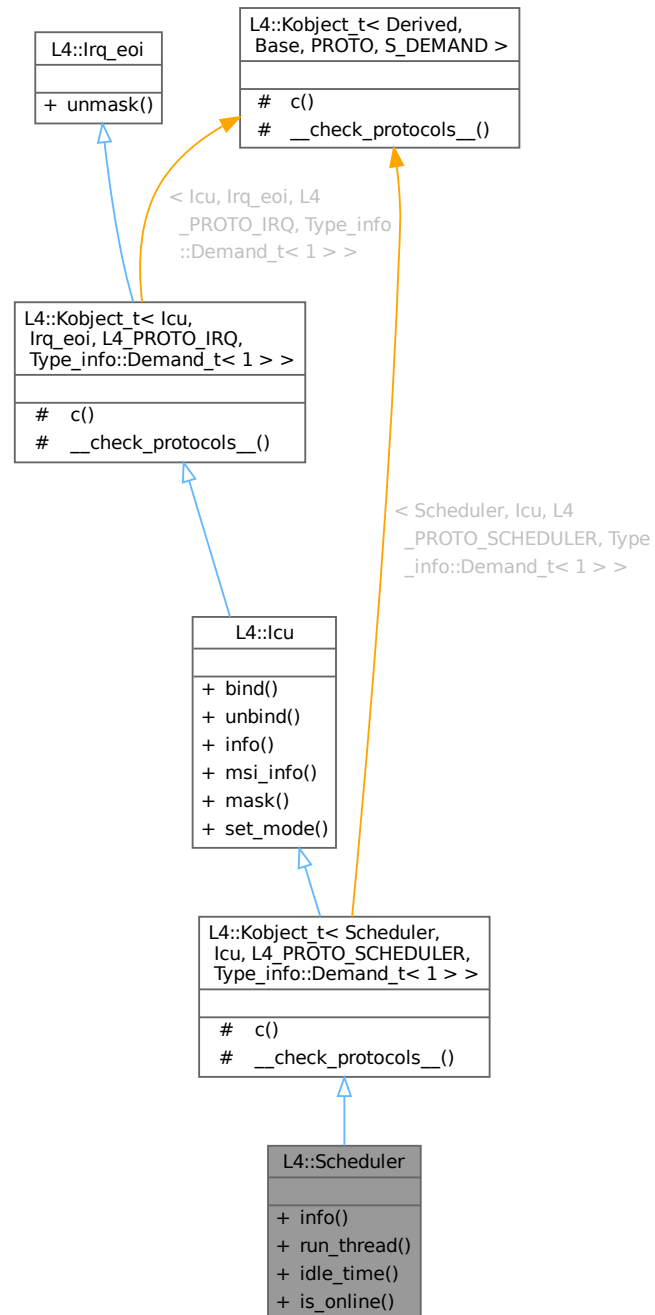
- [l4/cxx/exceptions](#)

16.197 L4::Scheduler Class Reference

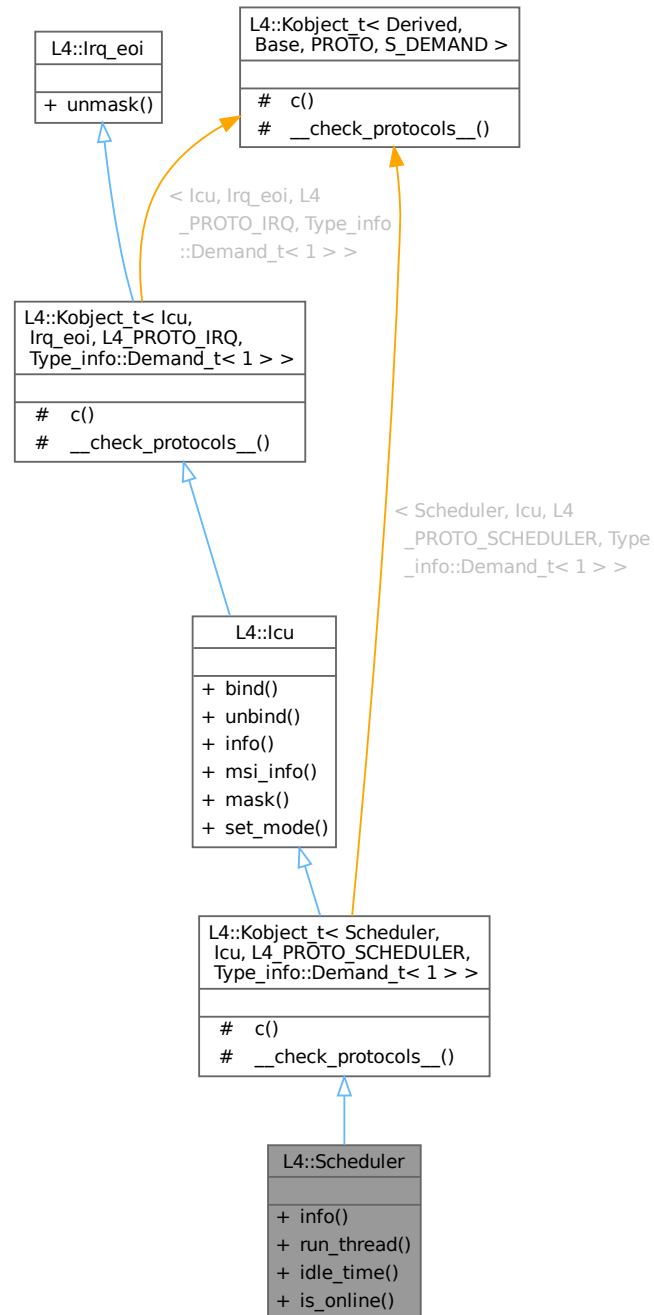
C++ interface of the [Scheduler](#) kernel object, see [Scheduler](#) for the C interface.

```
#include <scheduler>
```

Inheritance diagram for L4::Scheduler:



Collaboration diagram for L4::Scheduler:



Public Member Functions

- `l4_msgtag_t info (l4_umword_t *cpu_max, l4_sched_cpu_set_t *cpus, l4_umword_t *sched_classes=nullptr, l4_utcb_t *utcb=l4_utcb()) const noexcept`
Get scheduler information.
- `l4_msgtag_t run_thread (lpc::Cap< Thread > thread, l4_sched_param_t const &sp)`
Run a thread on a *Scheduler*.

- `l4_msgtag_t idle_time` (`l4_sched_cpu_set_t` const &cpus, `l4_kernel_clock_t` *us)
Query the idle time (in μ s) of a CPU.
- `bool is_online` (`l4_umword_t` cpu, `l4_utcb_t` *utcb=`l4_utcb`()) const noexcept
Query if a CPU is online.

Public Member Functions inherited from `L4::lcu`

- `l4_msgtag_t bind` (unsigned irqnum, `L4::Cap`< `Triggerable` > irq, `l4_utcb_t` *utcb=`l4_utcb`()) noexcept
Bind an interrupt line of an interrupt controller to an interrupt object.
- `l4_msgtag_t unbind` (unsigned irqnum, `L4::Cap`< `Triggerable` > irq, `l4_utcb_t` *utcb=`l4_utcb`()) noexcept
Remove binding of an interrupt line from the interrupt controller object.
- `l4_msgtag_t info` (`l4_icu_info_t` *info, `l4_utcb_t` *utcb=`l4_utcb`()) noexcept
Get information about the ICU features.
- `l4_msgtag_t msi_info` (`l4_umword_t` irqnum, `l4_uint64_t` source, `l4_icu_msi_info_t` *msi_info)
Get MSI info about IRQ.
- `l4_msgtag_t mask` (unsigned irqnum, `l4_umword_t` *label=0, `l4_timeout_t` to=`L4_IPC_NEVER`, `l4_utcb_t` *utcb=`l4_utcb`()) noexcept
Mask an IRQ line.
- `l4_msgtag_t set_mode` (unsigned irqnum, `l4_umword_t` mode, `l4_utcb_t` *utcb=`l4_utcb`()) noexcept
Set interrupt mode.

Public Member Functions inherited from `L4::lrq_eoi`

- `l4_msgtag_t unmask` (unsigned irqnum, `l4_umword_t` *label=0, `l4_timeout_t` to=`L4_IPC_NEVER`, `l4_utcb_t` *utcb=`l4_utcb`()) noexcept
Unmask the given interrupt line.

Additional Inherited Members

Protected Types inherited from

`L4::Kobject_t`< `Scheduler`, `lcu`, `L4_PROTO_SCHEDULER`, `Type_info::Demand_t`< 1 > >

- typedef `Scheduler` **Class**
The target interface type (inheriting from `Kobject_t`).
- typedef `Typeid::Iface`< `PROTO`, `Scheduler` > **__Iface**
The interface description for the derived class.
- typedef `Typeid::Merge_list`< `Typeid::Iface_list`< **__Iface** >, typename `lcu::__Iface_list` > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Types inherited from

`L4::Kobject_t`< `lcu`, `lrq_eoi`, `L4_PROTO_IRQ`, `Type_info::Demand_t`< 1 > >

- typedef `lcu` **Class**
The target interface type (inheriting from `Kobject_t`).
- typedef `Typeid::Iface`< `PROTO`, `lcu` > **__Iface**
The interface description for the derived class.
- typedef `Typeid::Merge_list`< `Typeid::Iface_list`< **__Iface** >, typename `lrq_eoi::__Iface_list` > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from**L4::Kobject_t< Scheduler, Icu, L4_PROTO_SCHEDULER, Type_info::Demand_t< 1 > >**

- **L4::Cap< Class > c ()** const noexcept

*Get the capability to ourselves.***Protected Member Functions inherited from****L4::Kobject_t< Icu, Irq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > >**

- **L4::Cap< Class > c ()** const noexcept

*Get the capability to ourselves.***Static Protected Member Functions inherited from****L4::Kobject_t< Scheduler, Icu, L4_PROTO_SCHEDULER, Type_info::Demand_t< 1 > >**

- **static void __check_protocols__ ()** noexcept

*Helper to check for protocol conflicts.***Static Protected Member Functions inherited from****L4::Kobject_t< Icu, Irq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > >**

- **static void __check_protocols__ ()** noexcept

*Helper to check for protocol conflicts.***16.197.1 Detailed Description**

C++ interface of the [Scheduler](#) kernel object, see [Scheduler](#) for the C interface.

The [Scheduler](#) interface allows a client to manage CPU resources. The API provides functions to query scheduler information, check the online state of CPUs, query CPU idle time and to start threads on defined CPU sets.

The scheduler offers IRQ number 0, which triggers when the number of online cores changes, e.g. due to hotplug events.

The [Scheduler](#) interface inherits from [L4::Icu](#) and [L4::Irq_eoi](#) for managing the scheduler virtual device IRQ which, in contrast to hardware IRQs, implements a limited functionality:

- Only IRQ line 0 is supported, no MSI vectors.
- The IRQ is edge-triggered and the IRQ mode cannot be changed.
- As the IRQ is edge-triggered, it does not have to be explicitly unmasked.

It depends on the platform, which hotplug events actually trigger the IRQ. Many platforms only support triggering the IRQ when a CPU core different from the boot CPU goes online.

Include File

```
#include <l4/sys/scheduler>
```

Definition at line 46 of file [scheduler](#).

16.197.2 Member Function Documentation

16.197.2.1 `idle_time()`

```
l4_msgtag_t L4::Scheduler::idle_time (  
    l4_sched_cpu_set_t const & cpus,  
    l4_kernel_clock_t * us)
```

Query the idle time (in μ s) of a CPU.

Parameters

	<i>cpus</i>	Set of CPUs to query. Only the idle time of the first selected CPU in <code>cpus.map</code> is queried.
out	<i>us</i>	Idle time of queried CPU in μ s.

Return values

0	Success.
-L4_EINVAL	Invalid CPU requested in cpu set.

This function retrieves the idle time in μ s of the first selected CPU in `cpus.map`. The idle time is the accumulated time a CPU has spent in the idle thread since its last reset. To calculate a load estimate l one has to retrieve the idle time at the beginning ($i1$) and the end ($i2$) of a known time interval t . The load is then calculated as $l = 1 - (i2 - i1)/t$.

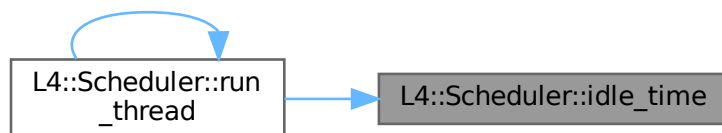
The idle time is only defined for online CPUs. Reading the idle time from offline CPUs is undefined and may result in either getting -L4_EINVAL or calculating an estimated (incorrect) load of 1.

Note

The idle time statistics of remote CPUs is updated on context switch events only, hence may not be up-to-date when requested cross-CPU. To get up-to-date idle time you should use a thread running on the same CPU of which the idle time is requested.

Referenced by [run_thread\(\)](#).

Here is the caller graph for this function:



16.197.2.2 info()

```

l4_msgtag_t L4::Scheduler::info (
    l4_umword_t * cpu_max,
    l4_sched_cpu_set_t * cpus,
    l4_umword_t * sched_classes = nullptr,
    l4_utcb_t * utcb = l4_utcb()) const [inline], [noexcept]
  
```

Get scheduler information.

Parameters

out	<i>cpu_max</i>	Maximum number of CPUs ever available. Optional, can be nullptr.
in, out	<i>cpus</i>	<i>cpus.offset</i> is first CPU of interest. <i>cpusgranularity</i> (see l4_sched_cpu_set_t). <i>cpus.map</i> Bitmap of online CPUs. Must not be nullptr.
out	<i>sched_classes</i>	A bitmap of available scheduling classes (see L4_scheduler_classes). Pass nullptr to omit this information.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

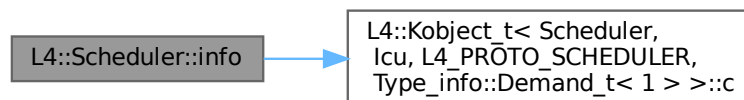
Return values

0	Success.
-L4_ERANGE	The given CPU offset is larger than the maximum number of CPUs.

Definition at line 74 of file [scheduler](#).

References [L4::Kobject_t< Scheduler, lcu, L4_PROTO_SCHEDULER, Type_info::Demand_t< 1 > >::c\(\)](#), [l4_sched_cpu_set_t::gran_offset](#), and [l4_sched_cpu_set_t::map](#).

Here is the call graph for this function:



16.197.2.3 is_online()

```

bool L4::Scheduler::is_online (
    l4_umword_t cpu,
    l4_utcb_t * utcb = l4_utcb()) const [inline], [noexcept]
  
```

Query if a CPU is online.

Parameters

<i>cpu</i>	CPU number whose online status should be queried.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

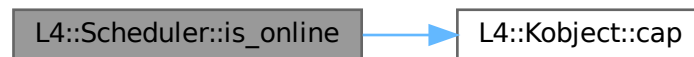
Return values

<i>true</i>	The CPU is online.
<i>false</i>	The CPU is offline

Definition at line 154 of file [scheduler](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



16.197.2.4 run_thread()

```

l4_msgtag_t L4::Scheduler::run_thread (
    ipc::Cap< Thread > thread,
    l4_sched_param_t const & sp)
  
```

Run a thread on a [Scheduler](#).

Parameters

<i>thread</i>	Capability of the thread to run.
<i>sp</i>	Scheduling parameters.

Return values

0	Success.
-L4_EINVAL	Invalid size of the scheduling parameter.

This function launches a thread on a CPU determined by the scheduling parameter `sp.affinity`. A thread can be intentionally stopped by migrating it on an offline or an invalid CPU. The thread is only guaranteed to run if the CPU it is migrated to is currently online.

Note

If the target CPU is currently not online, there is no guarantee that the thread will ever run, even if the CPU comes online later on.

A scheduler may impose a policy with regard to selecting CPUs. However the scheduler is required to ensure the following two properties:

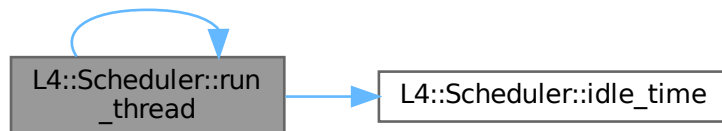
- Two threads with disjoint CPU sets must be scheduled to different CPUs.

- Two threads with identical CPU sets selecting only a single CPU must be scheduled to the same CPU.

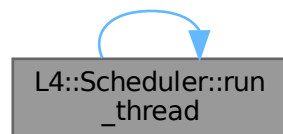
References [idle_time\(\)](#), [L4_SCHEDULER_IDLE_TIME_OP](#), and [run_thread\(\)](#).

Referenced by [run_thread\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

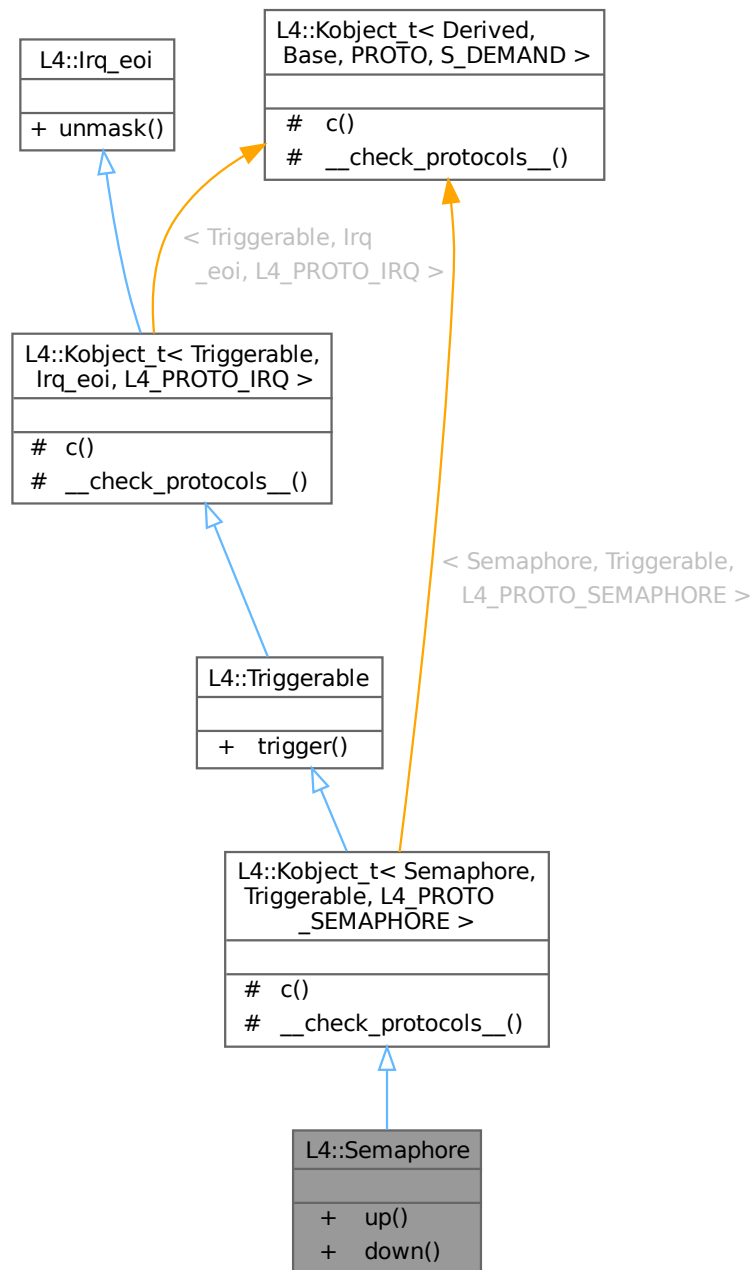
- [l4/sys/scheduler](#)

16.198 L4::Semaphore Struct Reference

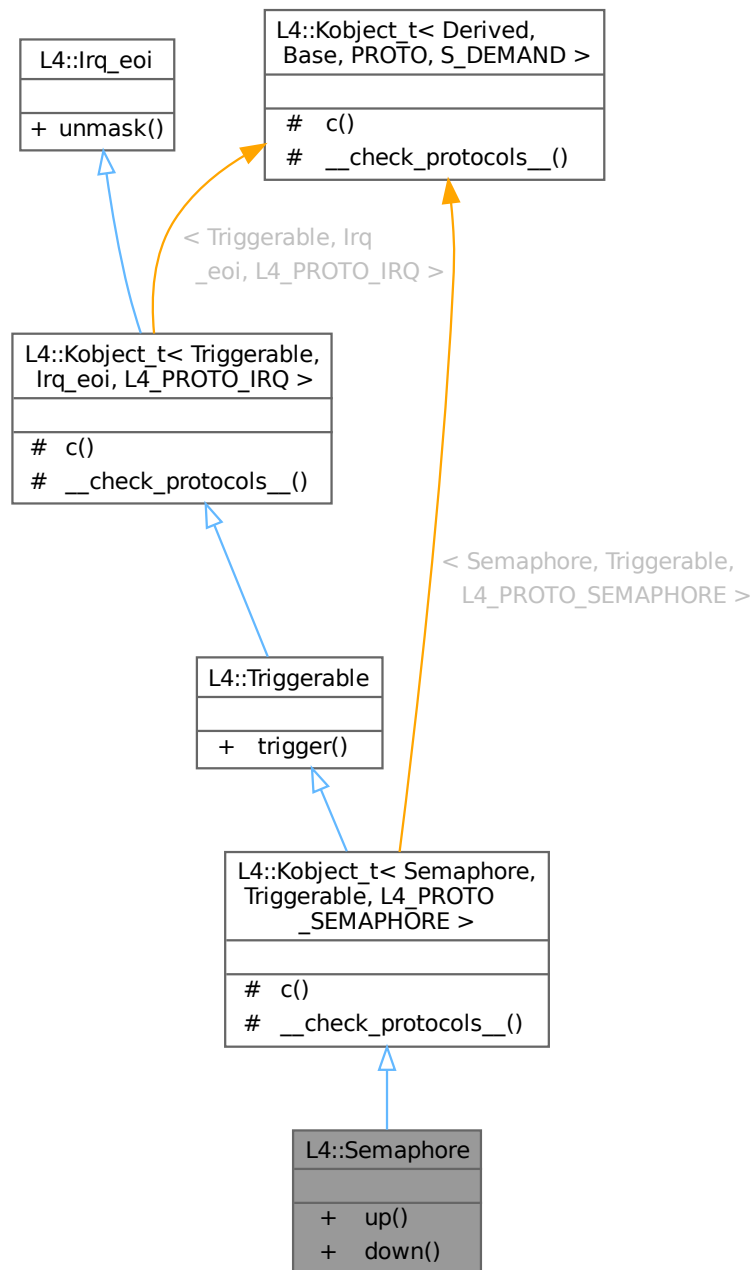
C++ Kernel-provided semaphore interface, see [Kernel-provided semaphore](#) for the C interface.

```
#include <semaphore>
```


Inheritance diagram for L4::Semaphore:



Collaboration diagram for L4::Semaphore:



Public Member Functions

- `l4_msgtag_t up (l4_utcb_t *utcb=l4_utcb()) noexcept`
Semaphore up operation (wrapper for `trigger()`).
- `l4_msgtag_t down (l4_timeout_t timeout=L4_IPC_NEVER, l4_utcb_t *utcb=l4_utcb()) noexcept`
Semaphore down operation.

Public Member Functions inherited from [L4::Triggerable](#)

- [l4_msgtag_t trigger](#) ([l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept
Trigger the object.

Public Member Functions inherited from [L4::Irq_eoi](#)

- [l4_msgtag_t unmask](#) (unsigned irqnum, [l4_umword_t](#) *label=0, [l4_timeout_t](#) to=[L4_IPC_NEVER](#), [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept
Unmask the given interrupt line.

Additional Inherited Members

Protected Types inherited from [L4::Kobject_t](#)< [Semaphore](#), [Triggerable](#), [L4_PROTO_SEMAPHORE](#) >

- typedef [Semaphore](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< [PROTO](#), [Semaphore](#) > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__Iface** >, typename [Triggerable](#)::__Iface_list > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Types inherited from [L4::Kobject_t](#)< [Triggerable](#), [Irq_eoi](#), [L4_PROTO_IRQ](#) >

- typedef [Triggerable](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< [PROTO](#), [Triggerable](#) > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__Iface** >, typename [Irq_eoi](#)::__Iface_list > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from [L4::Kobject_t](#)< [Semaphore](#), [Triggerable](#), [L4_PROTO_SEMAPHORE](#) >

- [L4::Cap](#)< [Class](#) > **c** () const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from [L4::Kobject_t](#)< [Triggerable](#), [Irq_eoi](#), [L4_PROTO_IRQ](#) >

- [L4::Cap](#)< [Class](#) > **c** () const noexcept
Get the capability to ourselves.

Static Protected Member Functions inherited from

[L4::Kobject_t](#) < [Semaphore](#), [Triggerable](#), [L4_PROTO_SEMAPHORE](#) >

- static void `__check_protocols__()` noexcept
Helper to check for protocol conflicts.

Static Protected Member Functions inherited from

[L4::Kobject_t](#) < [Triggerable](#), [Irq_eoi](#), [L4_PROTO_IRQ](#) >

- static void `__check_protocols__()` noexcept
Helper to check for protocol conflicts.

16.198.1 Detailed Description

C++ Kernel-provided semaphore interface, see [Kernel-provided semaphore](#) for the C interface.

This is the interface for kernel-provided semaphore objects. The object provides the classical functions `up()` and `down()` for counting the semaphore and blocking. The semaphore is a [Triggerable](#) with respect to the `up()` function, this means that a semaphore can be bound to an interrupt line at an ICU ([L4::lcu](#)) and incoming interrupts increment the semaphore counter.

The `down()` method decrements the semaphore counter and blocks if the counter is already zero. Blocking on a semaphore may—as all blocking operations—either return successfully, or be aborted due to an expired timeout provided to the `down()` operation, or due to an [L4::Thread::ex_regs\(\)](#) operation with the [L4_THREAD_EX_REGS_CANCEL](#) flag set.

A semaphore object is initialized with counter value 0.

The main reason for using a semaphore instead of an [L4::Irq](#) is to ensure that incoming trigger signals do not interfere with any open-wait operations, as used for example in a server loop.

Note that this is a kernel-level semaphore primitive that shall be used to implement user-level, application-usable synchronization primitives. For example, use `pthread_mutex` functions in applications if possible. When implementing a synchronization primitive, please ensure to only use [L4::Semaphore](#) in the case of contention, and use atomic operations for the non-contended case.

Definition at line 51 of file [semaphore](#).

16.198.2 Member Function Documentation

16.198.2.1 `down()`

```
l4_msgtag_t L4::Semaphore::down (
    l4_timeout_t timeout = L4_IPC_NEVER,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
```

[Semaphore](#) down operation.

Parameters

<i>timeout</i>	Timeout for blocking the semaphore down operation. Note: The receive timeout of this timeout-pair is significant for blocking, the send part is usually non-blocking.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Returns

Syscall return tag. Use [l4_error\(\)](#) to check for errors.

Return values

<code>-L4_EPERM</code>	Insufficient permissions; see precondition.
------------------------	---

Precondition

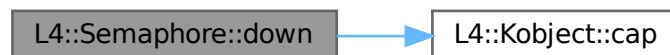
The invoked [Semaphore](#) capability must have the permission [L4_CAP_FPAGE_S](#).

This method decrements the semaphore counter by one, or blocks if the counter is already zero, until either a timeout or cancel condition hits or the counter is increased by an [up\(\)](#) operation.

Definition at line 89 of file [semaphore](#).

References [L4::Kobject::cap\(\)](#), and [L4_IPC_NEVER](#).

Here is the call graph for this function:

**16.198.2.2 up()**

```

l4_msgtag_t L4::Semaphore::up (
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

[Semaphore](#) up operation (wrapper for [trigger\(\)](#)).

Parameters

<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .
-------------	--

Returns

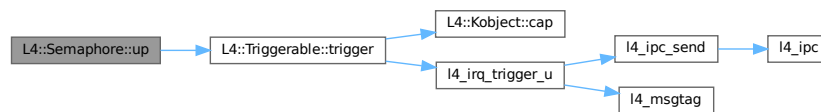
Syscall return tag for a send-only operation, this means there is no return value except [L4_MSGTAG_ERROR](#) indicating success or failure of the send operation. Use [l4_ipc_error\(\)](#) to check for errors and **do not** use [l4_error\(\)](#).

Increases the semaphore counter by one if it is smaller than an unspecified limit. The unspecified limit is guaranteed to be at least $2^{31}-1$.

Definition at line 67 of file [semaphore](#).

References [L4::Triggerable::trigger\(\)](#).

Here is the call graph for this function:



The documentation for this struct was generated from the following file:

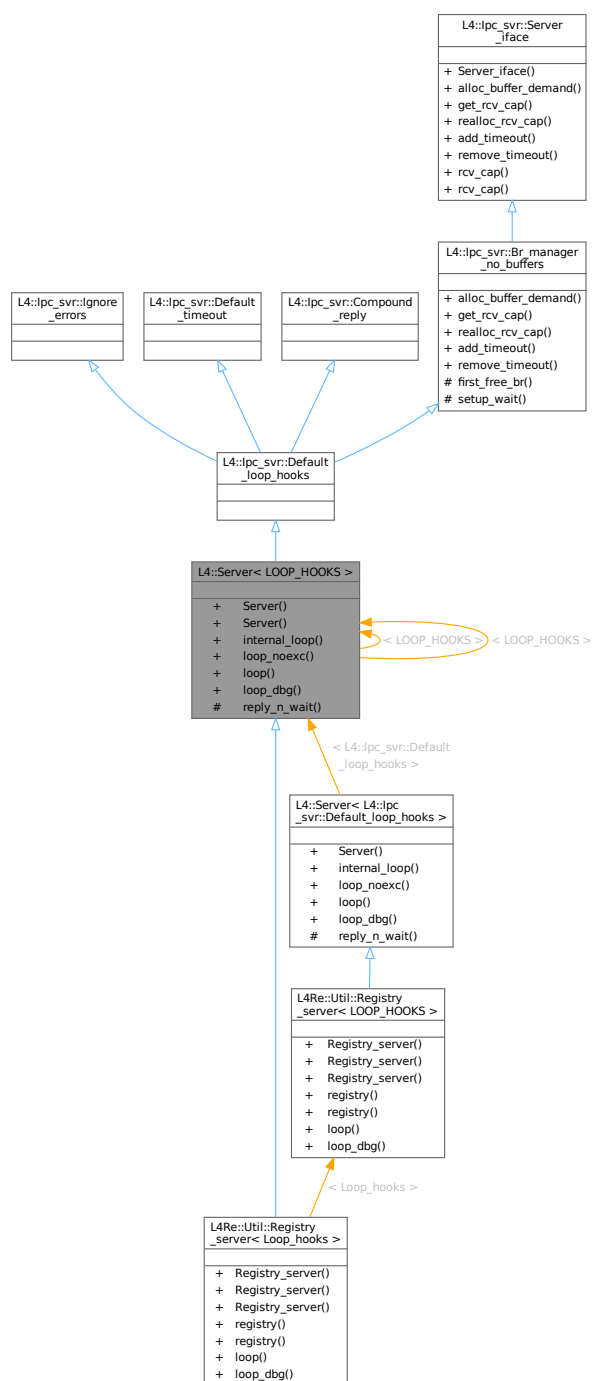
- [l4/sys/semaphore](#)

16.199 L4::Server< LOOP_HOOKS > Class Template Reference

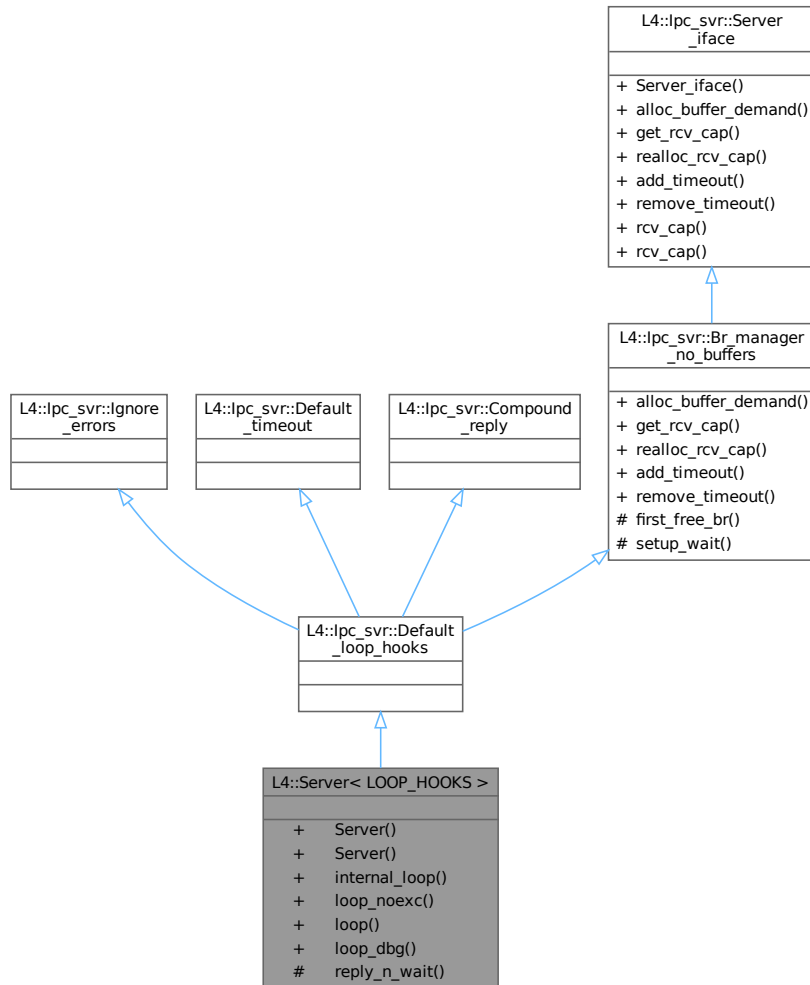
Basic server loop for handling client requests.

```
#include <ipc_server_loop>
```

Inheritance diagram for L4::Server< LOOP_HOOKS >:



Collaboration diagram for L4::Server< LOOP_HOOKS >:



Public Member Functions

- `Server (l4_utcb_t *)`
Initializes the server loop.
- `Server ()`
Initializes the server loop.
- `template<typename DISPATCH>`
`L4_NORETURN void internal_loop (DISPATCH dispatch, l4_utcb_t *)`
The server loop.
- `template<typename R>`
`L4_NORETURN void loop_noexc (R r, l4_utcb_t *u=l4_utcb())`
Server loop without exception handling.
- `template<typename EXC, typename R>`
`L4_NORETURN void loop (R r, l4_utcb_t *u=l4_utcb())`
Server loop with internal exception handling.
- `template<typename EXC, typename R, typename Printer>`
`L4_NORETURN void loop_dbg (R r, Printer p, l4_utcb_t *u=l4_utcb())`
Server loop with internal exception handling including message printing.

Public Member Functions inherited from L4::lpc_svr::Br_manager_no_buffers

- int **alloc_buffer_demand** ([Demand](#) const &demand) override
Tells the server to allocate buffers for the given demand.
- [L4::Cap](#)< void > **get_rcv_cap** (int) const override
Returns [L4::Cap](#)<void>::Invalid, we have no buffer management.
- int **realloc_rcv_cap** (int) override
Returns -L4_ENOMEM, we have no buffer management.
- int **add_timeout** ([Timeout](#) *, [l4_kernel_clock_t](#)) override
Returns -L4_ENOSYS, we have no timeout queue.
- int **remove_timeout** ([Timeout](#) *) override
Returns -L4_ENOSYS, we have no timeout queue.

Public Member Functions inherited from L4::lpc_svr::Server_iface

- **Server_iface** ()
Make a server interface.
- template<typename T>
[L4::Cap](#)< T > **rcv_cap** (int index) const
Get given receive buffer as typed capability.
- [L4::Cap](#)< void > **rcv_cap** (int index) const
Get receive cap with the given index as generic (void) type.

Protected Member Functions

- [l4_msgtag_t](#) **reply_n_wait** ([l4_msgtag_t](#) reply, [l4_umword_t](#) *p, [l4_utcb_t](#) *)
Internal implementation for reply and wait.

Protected Member Functions inherited from L4::lpc_svr::Br_manager_no_buffers

- unsigned **first_free_br** () const
Returns 1 as first free buffer.
- void **setup_wait** ([l4_utcb_t](#) *utcb, [L4::lpc_svr::Reply_mode](#))
Setup wait function for the server loop ([Server](#)<>).

Additional Inherited Members

Public Types inherited from L4::lpc_svr::Server_iface

- typedef [L4::Type_info::Demand](#) **Demand**
Data type expressing server-side demand for receive buffers.

16.199.1 Detailed Description

```
template<typename LOOP_HOOKS = lpc_svr::Default_loop_hooks>
class L4::Server< LOOP_HOOKS >
```

Basic server loop for handling client requests.

Parameters

<code>LOOP_HOOKS</code>	the server inherits from <code>LOOP_HOOKS</code> and calls the hooks defined in <code>LOOP_HOOKS</code> in the server loop. See lpc_svr::Default_loop_hooks , lpc_svr::Ignore_errors , lpc_svr::Default_timeout , lpc_svr::Compound_reply , and lpc_svr::Br_manager_no_buffers .
-------------------------	--

This is basically a simple server loop that uses a single message buffer for receiving requests and sending replies. The dispatcher determines how incoming messages are handled.

Definition at line 307 of file [ipc_server_loop](#).

16.199.2 Constructor & Destructor Documentation

16.199.2.1 Server()

```
template<typename LOOP_HOOKS = Ipc_svr::Default_loop_hooks>
L4::Server< LOOP_HOOKS >::Server (
    l4_utcb_t * ) [inline], [explicit]
```

Initializes the server loop.

Note that this variant is deprecated. The UTCB can be specified with the server loop function ([l4_utcb\(\)](#) is the default). (2021-10)

Definition at line 319 of file [ipc_server_loop](#).

16.199.3 Member Function Documentation

16.199.3.1 internal_loop()

```
template<typename L>
template<typename DISPATCH>
L4_NORETURN void L4::Server< L >::internal_loop (
    DISPATCH dispatch,
    l4_utcb_t * utcb) [inline]
```

The server loop.

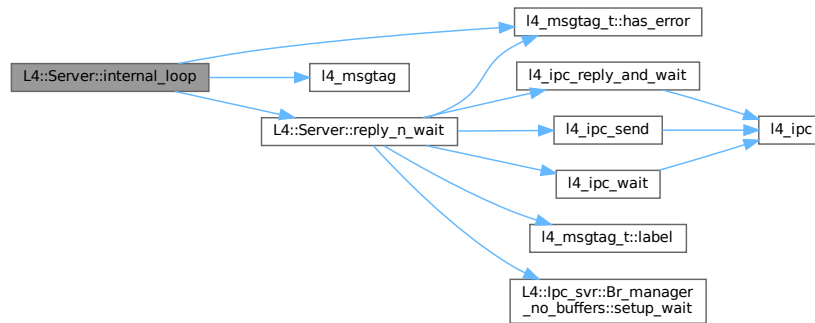
This function usually never returns, it waits for incoming messages calls the dispatcher, sends a reply and waits again.

Definition at line 405 of file [ipc_server_loop](#).

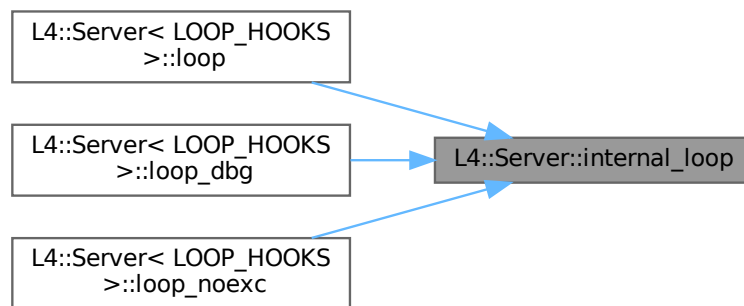
References [l4_msgtag_t::has_error\(\)](#), [L4_ENOREPLY](#), [l4_msgtag\(\)](#), and [reply_n_wait\(\)](#).

Referenced by [loop\(\)](#), [loop_dbg\(\)](#), and [loop_noexc\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.199.3.2 loop()

```

template<typename LOOP_HOOKS = Ipc_svr::Default_loop_hooks>
template<typename EXC, typename R>
L4_NORETURN void L4::Server< LOOP_HOOKS >::loop (
    R r,
    l4_utcb_t * u = l4_utcb()) [inline]

```

Server loop with internal exception handling.

This server loop translates [L4::Runtime_error](#) exceptions into negative error return codes sent to the caller.

Definition at line 355 of file [ipc_server_loop](#).

16.199.3.3 loop_dbg()

```
template<typename LOOP_HOOKS = Ipc_svr::Default_loop_hooks>
template<typename EXC, typename R, typename Printer>
L4_NORETURN void L4::Server< LOOP_HOOKS >::loop_dbg (
    R r,
    Printer p,
    l4_utcb_t * u = l4_utcb()) [inline]
```

[Server](#) loop with internal exception handling including message printing.

Exceptions are translated into error codes, just like in [loop\(\)](#). In addition, error messages are printed using the Printer argument.

Definition at line 368 of file [ipc_server_loop](#).

The documentation for this class was generated from the following file:

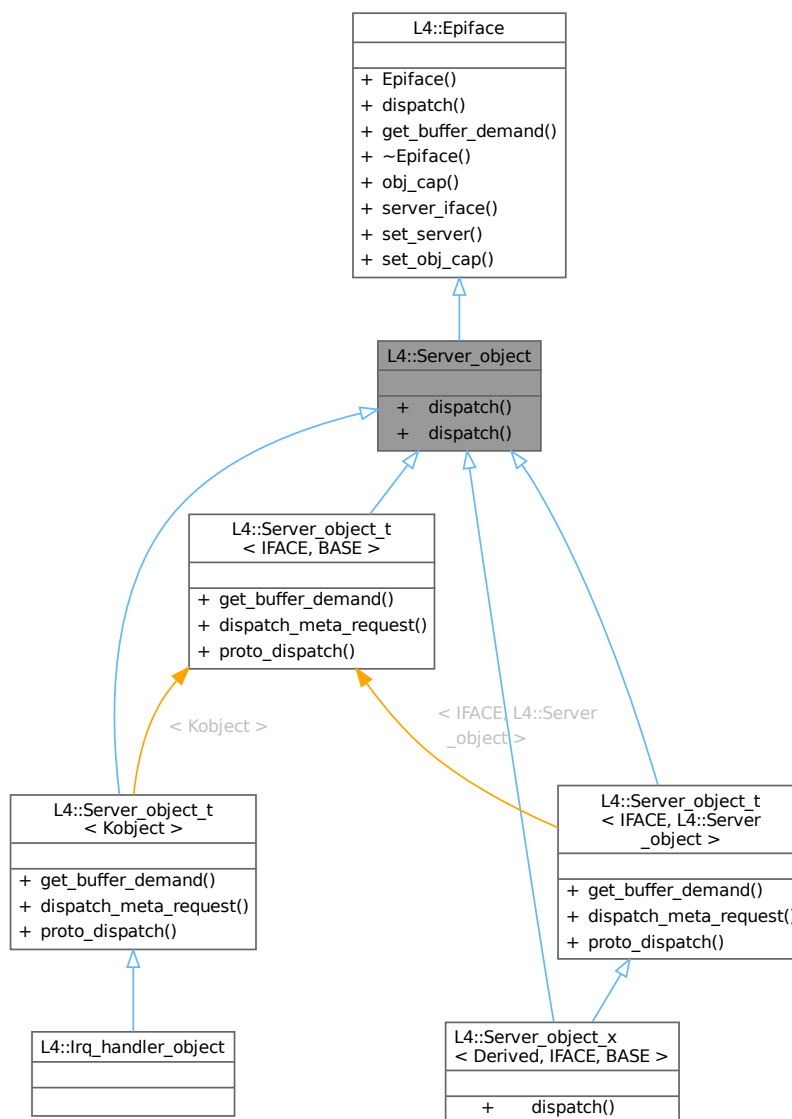
- l4/sys/cxx/ipc_server_loop

16.200 L4::Server_object Class Reference

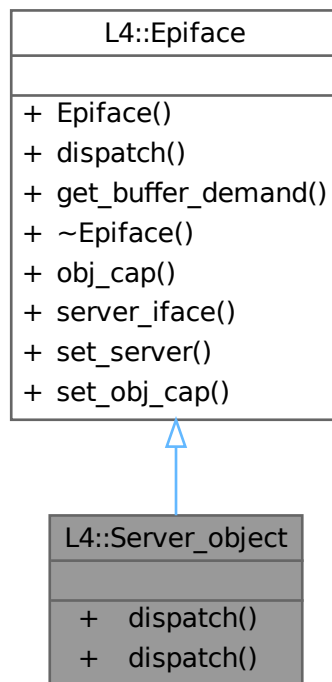
Abstract server object to be used with [L4::Server](#) and [L4::Basic_registry](#).

```
#include <ipc_server>
```

Inheritance diagram for L4::Server_object:



Collaboration diagram for L4::Server_object:



Public Member Functions

- virtual int `dispatch` (unsigned long rights, `lpc::lostream` &ios)=0
The abstract handler for client requests to the object.
- `l4_msgtag_t` `dispatch` (`l4_msgtag_t` tag, unsigned rights, `l4_utcb_t` *utcb) override
The abstract handler for client requests to the object.

Public Member Functions inherited from `L4::Epiface`

- `Epiface` ()
Make a server object.
- virtual `Demand` `get_buffer_demand` () const =0
Get the server-side receive buffer demand for this object.
- virtual `~Epiface` ()=0
Destroy the object.
- Stored_cap `obj_cap` () const
Get the capability to the kernel object belonging to this object.
- `Server_iface` * `server_iface` () const
Get pointer to server interface at which the object is currently registered.
- int `set_server` (`Server_iface` *srv, `Cap`< void > cap, bool managed=false)
Set server registration info for the object.
- void `set_obj_cap` (`Cap`< void > const &cap)
Deprecated server registration function.

Additional Inherited Members

Public Types inherited from L4::Epiface

- typedef [ipc_svr::Server_iface](#) **Server_iface**
Type for abstract server interface.
- typedef [ipc_svr::Server_iface::Demand](#) **Demand**
Type for server-side receive buffer demand.

16.200.1 Detailed Description

Abstract server object to be used with [L4::Server](#) and [L4::Basic_registry](#).

Note

Usually [L4::Server_object_t](#) is used as a base class when writing server objects.

This server object provides an abstract interface that is used by the [L4::Registry_dispatcher](#) model. You can derive subclasses from this interface and implement application specific server objects.

Definition at line 38 of file [ipc_server](#).

16.200.2 Member Function Documentation

16.200.2.1 dispatch() [1/2]

```
l4_msgtag_t L4::Server_object::dispatch (
    l4_msgtag_t tag,
    unsigned rights,
    l4_utcb_t * utcb) [inline], [override], [virtual]
```

The abstract handler for client requests to the object.

Parameters

<i>tag</i>	The message tag for this invocation.
<i>rights</i>	The rights bits in the invoked capability.
<i>utcb</i>	The UTCB used for the invocation.

Return values

<code>-L4_ENOREPLY</code>	No reply message is send.
<code>< 0</code>	Error, reply with error code.
<code>>= 0</code>	Success, reply with return value.

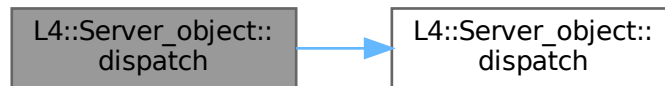
This function must be implemented by application specific server objects.

Implements [L4::Epiface](#).

Definition at line 60 of file [ipc_server](#).

References [dispatch\(\)](#).

Here is the call graph for this function:



16.200.2.2 `dispatch()` [2/2]

```
virtual int L4::Server_object::dispatch (
    unsigned long rights,
    Ipc::Iostream & ios) [pure virtual]
```

The abstract handler for client requests to the object.

Parameters

<i>rights</i>	The rights bits in the invoked capability.
<i>ios</i>	The lpc::lostream for reading the request and writing the reply.

Return values

<code>-L4_ENOREPLY</code>	Instructs the server loop to not send a reply.
<code>< 0</code>	Error, reply with error code.
<code>>= 0</code>	Success, reply with return value.

This function must be implemented by application specific server objects. The implementation must unmarshall data from the stream (`ios`) and create a reply by marshalling to the stream (`ios`). For details about the IPC stream see [IPC stream operators](#).

Note

You need to extract the complete message from the `ios` stream before inserting any reply data or before doing any function call that may use the UTCB. Otherwise, the incoming message may get lost.

Implemented in [L4::Server_object_x< Derived, IFACE, BASE >](#).

Examples

[examples/libs/l4re/c++/shared_ds/ds_srv.cc](#), and [examples/libs/l4re/streammap/server.cc](#).

Referenced by [dispatch\(\)](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

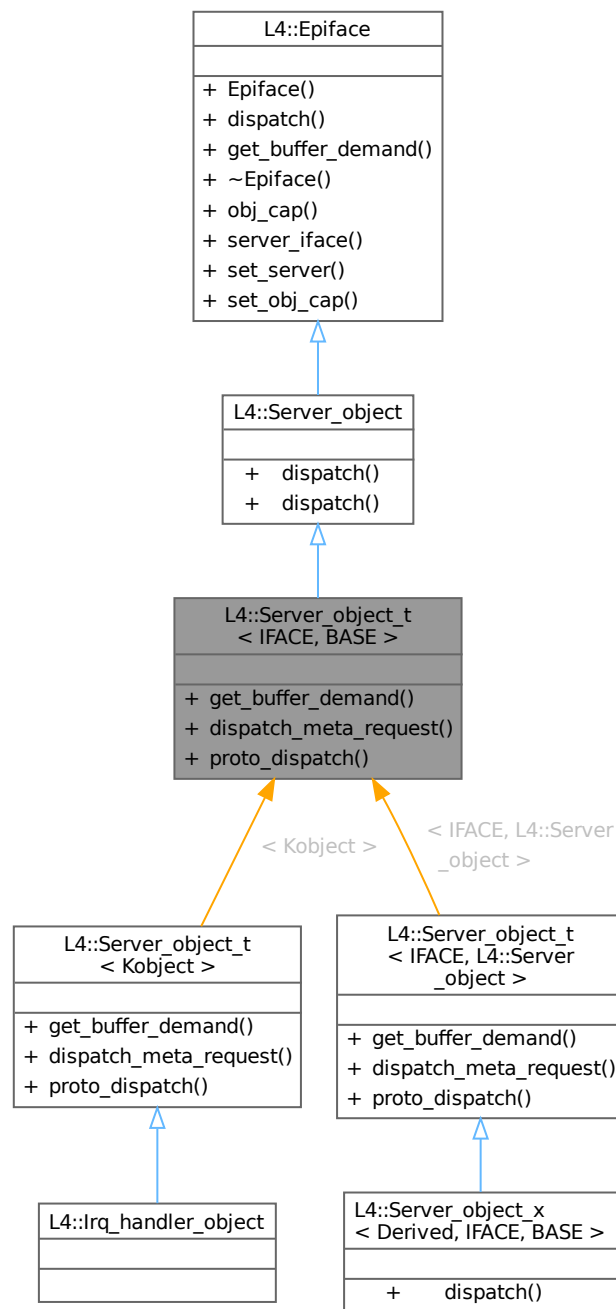
- [l4/cxx/ipc_server](#)

16.201 L4::Server_object_t< IFACE, BASE > Struct Template Reference

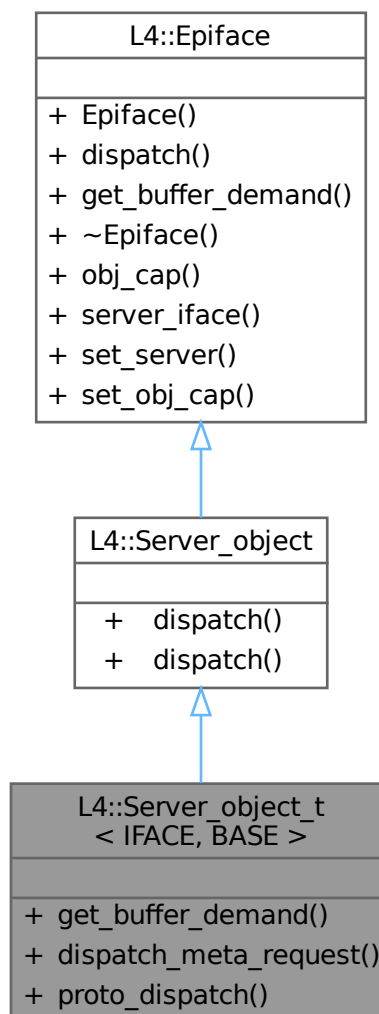
Base class (template) for server implementing server objects.

```
#include <ipc_server>
```

Inheritance diagram for L4::Server_object_t< IFACE, BASE >:



Collaboration diagram for L4::Server_object_t< IFACE, BASE >:



Public Types

- typedef **IFACE Interface**
Data type of the IPC interface definition.

Public Types inherited from **L4::Epiface**

- typedef **lpc_svr::Server_iface Server_iface**
Type for abstract server interface.
- typedef **lpc_svr::Server_iface::Demand Demand**
Type for server-side receive buffer demand.

Public Member Functions

- `BASE::Demand` `get_buffer_demand` () const override
- `int` `dispatch_meta_request` (`L4::lpc::lostream` &ios)

Implementation of the meta protocol based on IFACE.

Public Member Functions inherited from `L4::Server_object`

- `virtual int` `dispatch` (unsigned long rights, `lpc::lostream` &ios)=0
The abstract handler for client requests to the object.
- `l4_msgtag_t` `dispatch` (`l4_msgtag_t` tag, unsigned rights, `l4_utcb_t` *utcb) override
The abstract handler for client requests to the object.

Public Member Functions inherited from `L4::Epiface`

- `Epiface` ()
Make a server object.
- `virtual ~Epiface` ()=0
Destroy the object.
- `Stored_cap` `obj_cap` () const
Get the capability to the kernel object belonging to this object.
- `Server_iface` * `server_iface` () const
Get pointer to server interface at which the object is currently registered.
- `int` `set_server` (`Server_iface` *srv, `Cap`< void > cap, bool managed=false)
Set server registration info for the object.
- `void` `set_obj_cap` (`Cap`< void > const &cap)
Deprecated server registration function.

Static Public Member Functions

- `template<typename THIS>`
`static int` `proto_dispatch` (`THIS` *self, `l4_umword_t` rights, `L4::lpc::lostream` &ios)
Implementation of protocol-based dispatch for this server object.

16.201.1 Detailed Description

```
template<typename IFACE, typename BASE = L4::Server_object>
struct L4::Server_object_t< IFACE, BASE >
```

Base class (template) for server implementing server objects.

Template Parameters

<code>IFACE</code>	The IPC interface class that defines the interface that shall be implemented.
<code>BASE</code>	The server object base class (usually <code>L4::Server_object</code>).

Examples

`examples/libs/l4re/c++/shared_ds/ds_srv.cc`, and `examples/libs/l4re/streammap/server.cc`.

Definition at line 80 of file `ipc_server`.

16.201.2 Member Function Documentation

16.201.2.1 dispatch_meta_request()

```
template<typename IFACE, typename BASE = L4::Server_object>
int L4::Server_object_t< IFACE, BASE >::dispatch_meta_request (
    L4::Ipc::Iostream & ios) [inline]
```

Implementation of the meta protocol based on *IFACE*.

Parameters

<i>ios</i>	The IO stream used for receiving the message.
------------	---

This function can be used to handle incoming [L4_PROTO_META](#) protocol requests. The implementation uses the [L4::Type_info](#) of *IFACE* to handle the requests. Call this function in the implementation of [Server_object::dispatch\(\)](#) when the received message tag has protocol [L4_PROTO_META](#) ([L4::Meta::Protocol](#)).

Definition at line 99 of file [ipc_server](#).

16.201.2.2 get_buffer_demand()

```
template<typename IFACE, typename BASE = L4::Server_object>
BASE::Demand L4::Server_object_t< IFACE, BASE >::get_buffer_demand () const [inline], [override],
[virtual]
```

Returns

the server-side buffer demand based in *IFACE*.

Implements [L4::Epiface](#).

Definition at line 86 of file [ipc_server](#).

16.201.2.3 proto_dispatch()

```
template<typename IFACE, typename BASE = L4::Server_object>
template<typename THIS>
int L4::Server_object_t< IFACE, BASE >::proto_dispatch (
    THIS * self,
    l4_umword_t rights,
    L4::Ipc::Iostream & ios) [inline], [static]
```

Implementation of protocol-based dispatch for this server object.

Parameters

<i>self</i>	The this pointer for the object (inherits from Server_object_t).
-------------	---

<i>rights</i>	The rights from the received IPC (forwarded to <code>p_dispatch()</code>).
<i>ios</i>	The message stream for the incoming and the reply message.

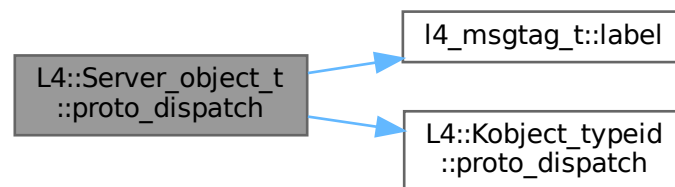
[Server](#) objects may call this function from their [dispatch\(\)](#) function. This function reads the protocol ID from the message tag and uses the `p_dispatch` code to dispatch to overloaded `p_dispatch` functions of self.

Definition at line 114 of file [ipc_server](#).

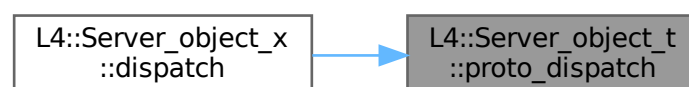
References [l4_msgtag_t::label\(\)](#), and [L4::Kobject_typeid< T >::proto_dispatch\(\)](#).

Referenced by [L4::Server_object_x< Derived, IFACE, BASE >::dispatch\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this struct was generated from the following file:

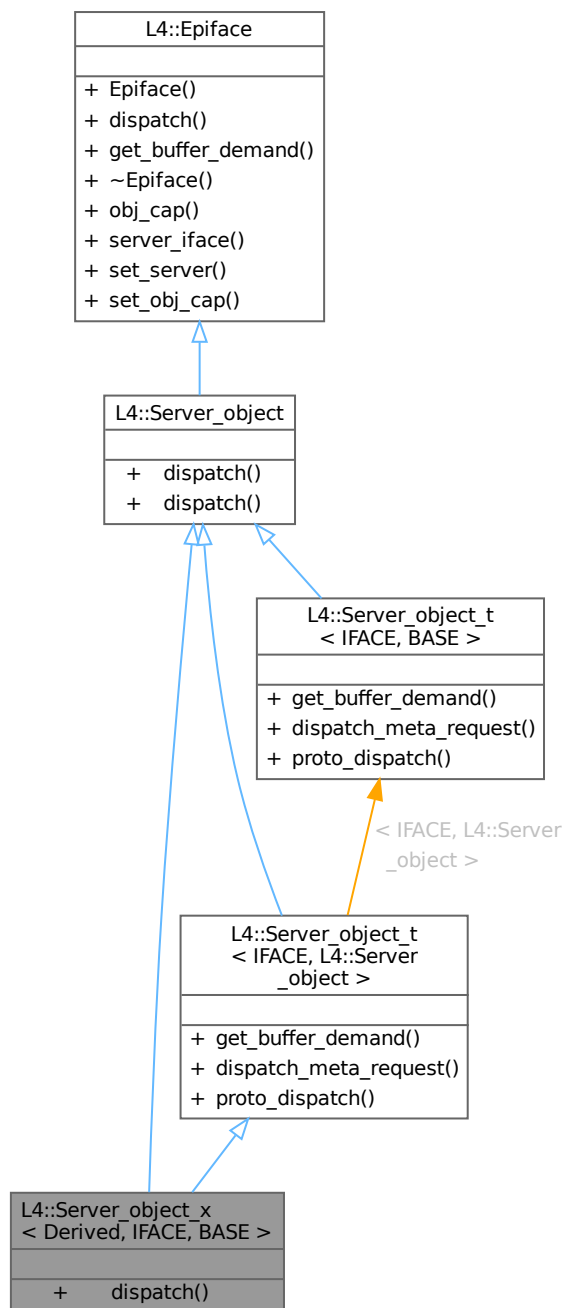
- [l4/cxx/ipc_server](#)

16.202 L4::Server_object_x< Derived, IFACE, BASE > Struct Template Reference

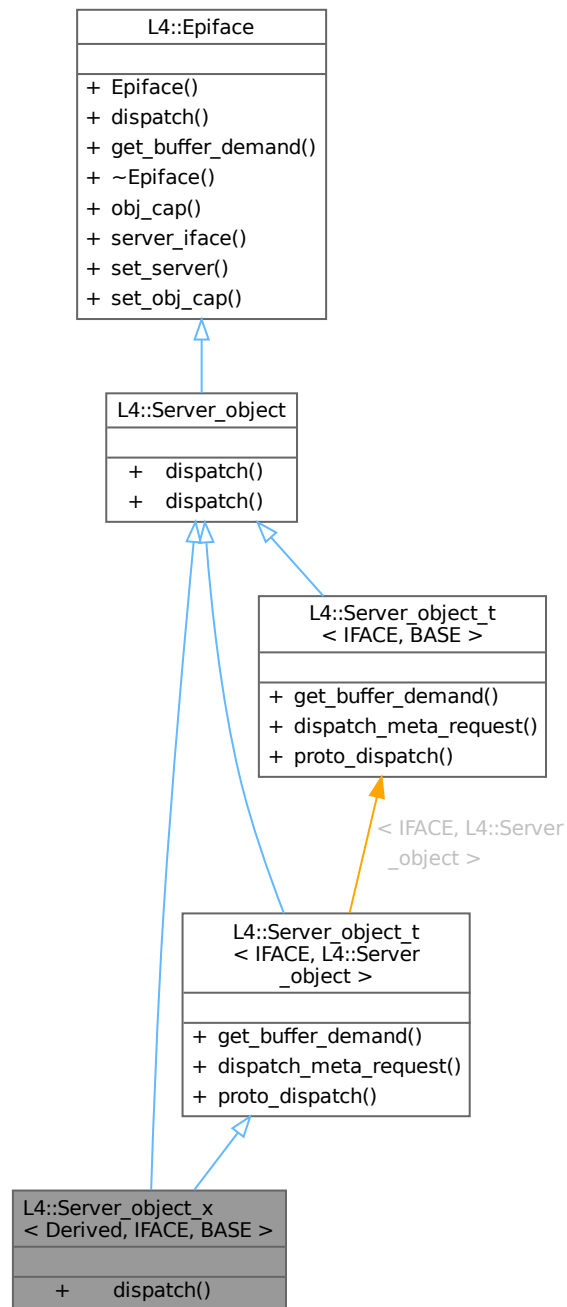
Helper class to implement p_dispatch based server objects.

```
#include <ipc_server>
```

Inheritance diagram for L4::Server_object_x< Derived, IFACE, BASE >:



Collaboration diagram for L4::Server_object_x< Derived, IFACE, BASE >:



Public Member Functions

- `int dispatch (l4_umword_t r, L4::lpc::lostream &ios)`
Implementation forwarding to `p_dispatch()`.

Public Member Functions inherited from L4::Server_object

- `l4_msgtag_t dispatch (l4_msgtag_t tag, unsigned rights, l4_utcb_t *utcb)` override

The abstract handler for client requests to the object.

Public Member Functions inherited from L4::Epiface

- **Epiface** ()
Make a server object.
- virtual **~Epiface** ()=0
Destroy the object.
- Stored_cap **obj_cap** () const
Get the capability to the kernel object belonging to this object.
- **Server_iface** * **server_iface** () const
Get pointer to server interface at which the object is currently registered.
- int **set_server** (**Server_iface** *srv, **Cap**< void > cap, bool managed=false)
Set server registration info for the object.
- void **set_obj_cap** (**Cap**< void > const &cap)
Deprecated server registration function.

Public Member Functions inherited from L4::Server_object_t< IFACE, L4::Server_object >

- L4::Server_object::Demand **get_buffer_demand** () const override
- int **dispatch_meta_request** (L4::lpc::lostream &ios)
Implementation of the meta protocol based on IFACE.

Additional Inherited Members

Public Types inherited from L4::Epiface

- typedef **lpc_svr::Server_iface** **Server_iface**
Type for abstract server interface.
- typedef **lpc_svr::Server_iface::Demand** **Demand**
Type for server-side receive buffer demand.

Public Types inherited from L4::Server_object_t< IFACE, L4::Server_object >

- typedef IFACE **Interface**
Data type of the IPC interface definition.

Static Public Member Functions inherited from L4::Server_object_t< IFACE, L4::Server_object >

- static int **proto_dispatch** (THIS *self, **l4_umword_t** rights, L4::lpc::lostream &ios)
Implementation of protocol-based dispatch for this server object.

16.202.1 Detailed Description

```
template<typename Derived, typename IFACE, typename BASE = L4::Server_object>
struct L4::Server_object_x< Derived, IFACE, BASE >
```

Helper class to implement p_dispatch based server objects.

Template Parameters

<i>Derived</i>	The data type of your server object class.
<i>IFACE</i>	The data type providing the interface definition for the object.
<i>BASE</i>	Optional data-type of the base server object (usually L4::Server_object)

This class implements the standard [dispatch\(\)](#) function of [L4::Server_object](#) and forwards incoming messages to a set of overloaded `p_dispatch()` functions. There must be a `p_dispatch()` function in *Derived* for each interface provided by *IFACE* with the signature

```
int p_dispatch(Iface *, unsigned rights, L4::Ipc::Iostream &)
```

that is called for messages with `protocol == Iface::Protocol`.

Example signature for [L4Re::Dataspace](#) is:

```
int p_dispatch(L4Re::Dataspace *, unsigned, L4::Ipc::Iostream &)
```

Definition at line [143](#) of file [ipc_server](#).

The documentation for this struct was generated from the following file:

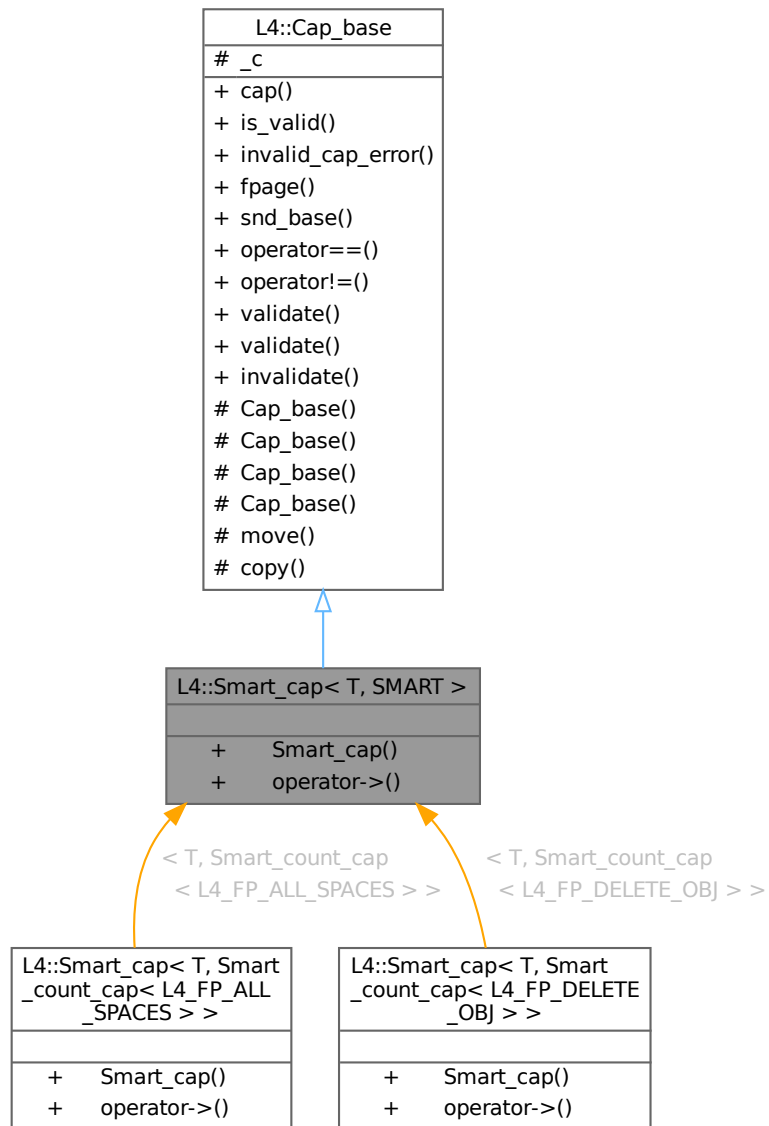
- [l4/cxx/ipc_server](#)

16.203 L4::Smart_cap< T, SMART > Class Template Reference

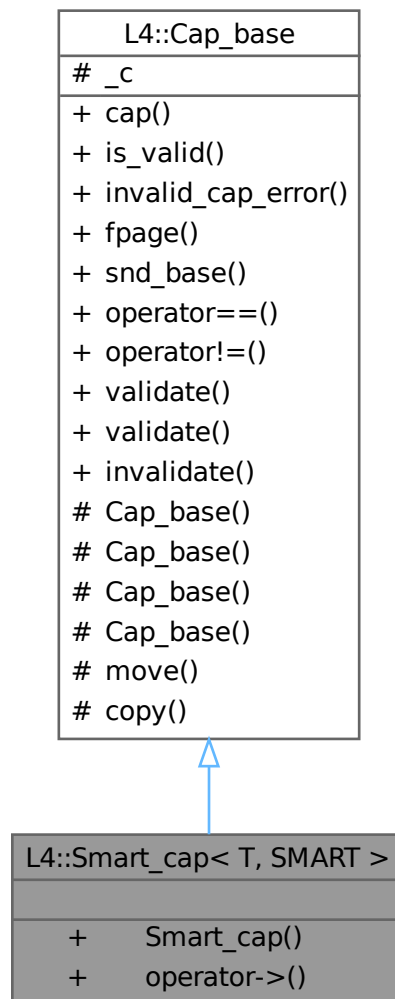
Smart capability class.

```
#include <smart_capability>
```

Inheritance diagram for L4::Smart_cap< T, SMART >:



Collaboration diagram for L4::Smart_cap< T, SMART >:



Public Member Functions

- `template<typename O>`
`Smart_cap (Cap< O > const &p) noexcept`
Internal constructor, use to generate a capability from a `this` pointer.
- `Cap< T > operator-> () const noexcept`
Member access of a `T`.

Public Member Functions inherited from L4::Cap_base

- `l4_cap_idx_t cap () const noexcept`
Return capability selector.
- `bool is_valid () const noexcept`

- *Test whether the capability is a valid capability index (i.e., not L4_INVALID_CAP).*
- `int invalid_cap_error () const noexcept`
Return the transported error code in an invalid capability index.
- `l4_fpage_t fpage` (unsigned rights=L4_CAP_FPAGE_RWS) const noexcept
Return flexpage for the capability.
- `l4_umword_t snd_base` (unsigned grant=L4_MAP_ITEM_MAP, l4_cap_idx_t base=L4_INVALID_CAP) const noexcept
Return send base.
- `bool operator== (Cap_base const &o) const noexcept`
Test if two capabilities are equal.
- `bool operator!= (Cap_base const &o) const noexcept`
Test if two capabilities are not equal.
- `l4_msgtag_t validate (l4_utcb_t *u=l4_utcb()) const noexcept`
Check whether a capability is present (refers to an object).
- `l4_msgtag_t validate (Cap< Task > task, l4_utcb_t *u=l4_utcb()) const noexcept`
Check whether a capability is present (refers to an object).
- `void invalidate () noexcept`
Set this capability to invalid (L4_INVALID_CAP).

Additional Inherited Members

Public Types inherited from L4::Cap_base

- `enum No_init_type { No_init }`
Special value for uninitialized capability objects.
- `enum Cap_type { Invalid = L4_INVALID_CAP }`
Invalid capability type.

Protected Member Functions inherited from L4::Cap_base

- `Cap_base (l4_cap_idx_t c) noexcept`
Generate a capability from its C representation.
- `Cap_base (Cap_type cap) noexcept`
Constructor to create an invalid capability.
- `Cap_base (l4_default_caps_t cap) noexcept`
Initialize capability with one of the default capabilities.
- `Cap_base () noexcept`
Create an uninitialized instance.
- `void move (Cap_base const &src) const`
Replace this capability with the contents of src.
- `void copy (Cap_base const &src) const`
Copy a capability.

Protected Attributes inherited from L4::Cap_base

- `l4_cap_idx_t _c`
The C representation of a capability selector.

16.203.1 Detailed Description

```
template<typename T, typename SMART>
class L4::Smart_cap< T, SMART >
```

Smart capability class.

Definition at line 25 of file [smart_capability](#).

16.203.2 Constructor & Destructor Documentation

16.203.2.1 Smart_cap()

```
template<typename T, typename SMART>
template<typename O>
L4::Smart_cap< T, SMART >::Smart_cap (
    Cap< O > const & p) [inline], [noexcept]
```

Internal constructor, use to generate a capability from a `this` pointer.

Attention

This constructor is only useful to generate a capability from the `this` pointer of an objected that is an [L4::Kobject](#). Do *never* use this constructor for something else!

Parameters

<i>p</i>	The <code>this</code> pointer of the Kobject or derived object
----------	--

Definition at line 62 of file [smart_capability](#).

The documentation for this class was generated from the following file:

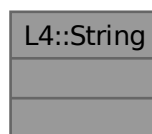
- [l4/sys/smart_capability](#)

16.204 L4::String Class Reference

A null-terminated string container class.

```
#include <string.h>
```

Collaboration diagram for L4::String:



16.204.1 Detailed Description

A null-terminated string container class.

Definition at line 22 of file [string.h](#).

The documentation for this class was generated from the following file:

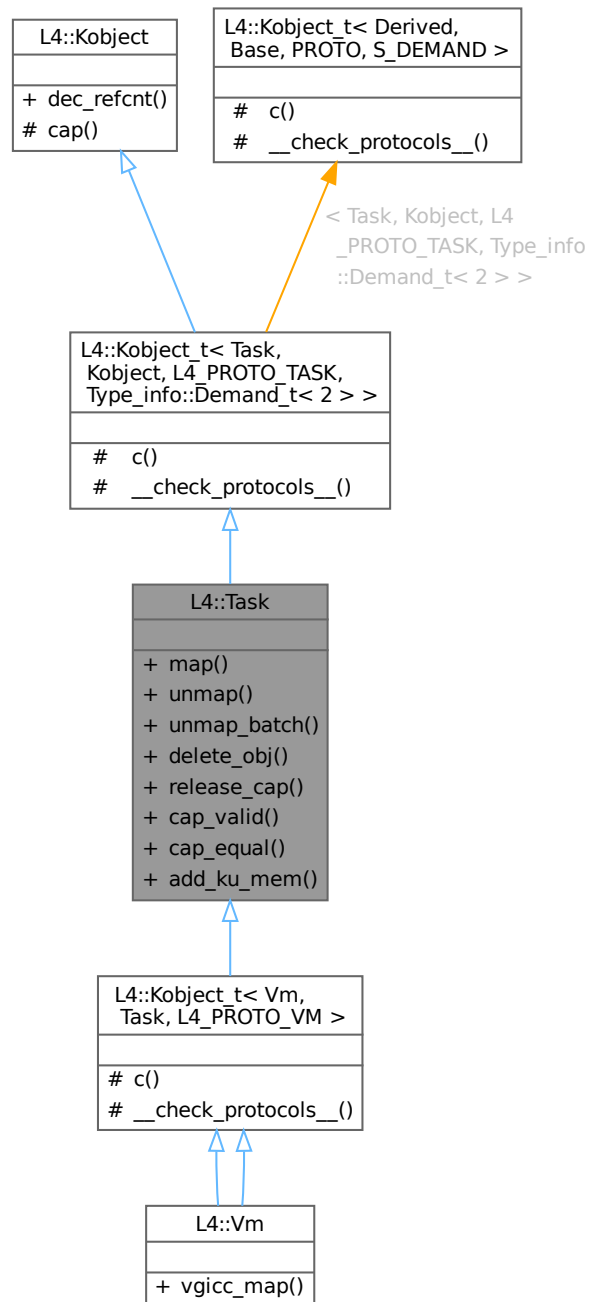
- [l4/cxx/string.h](#)

16.205 L4::Task Class Reference

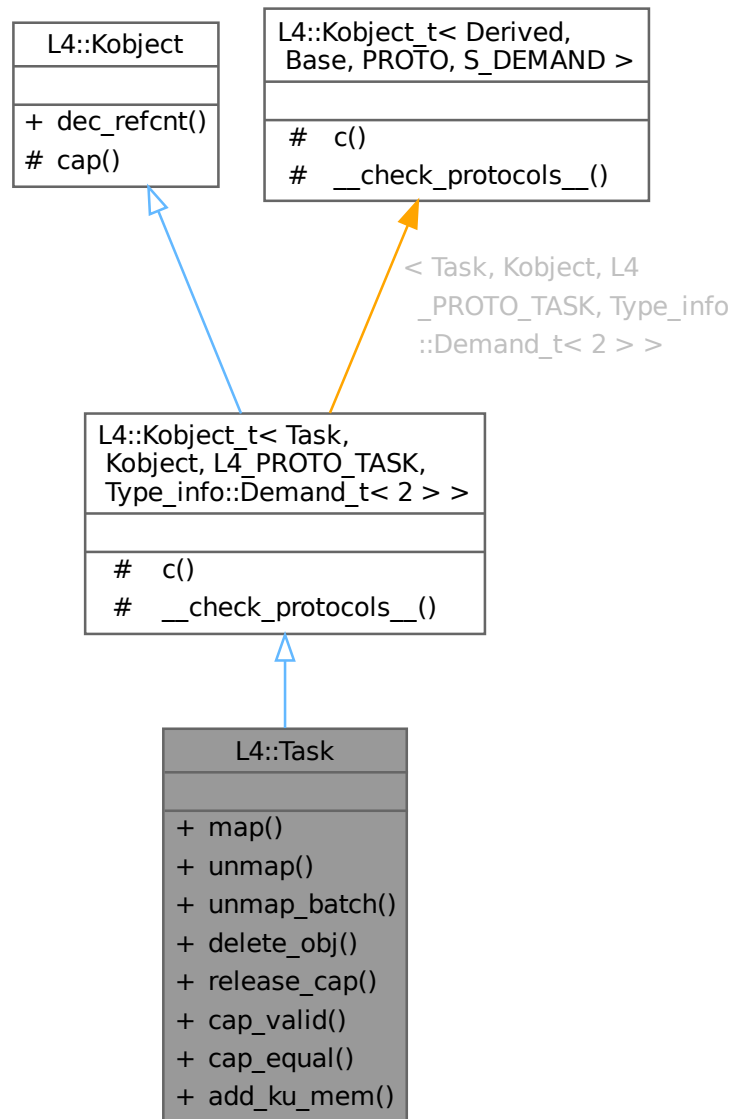
C++ interface of the [Task](#) kernel object, see [Task](#) for the C interface.

```
#include <task>
```

Inheritance diagram for L4::Task:



Collaboration diagram for L4::Task:



Public Member Functions

- `l4_msgtag_t map (Cap< Task > const &src_task, l4_fpage_t const &snd_fpage, l4_umword_t snd_base, l4_utcb_t *utcb=l4_utcb()) noexcept`
Map resources available in the source task to a destination task.
- `l4_msgtag_t unmap (l4_fpage_t const &fpage, l4_umword_t map_mask, l4_utcb_t *utcb=l4_utcb()) noexcept`
Revoke rights from the task.
- `l4_msgtag_t unmap_batch (l4_fpage_t const *fpages, unsigned num_fpages, l4_umword_t map_mask, l4_utcb_t *utcb=l4_utcb()) noexcept`
Revoke rights from a task.
- `l4_msgtag_t delete_obj (L4::Cap< void > obj, l4_utcb_t *utcb=l4_utcb()) noexcept`

Release capability and delete object.

- `l4_msgtag_t release_cap (L4::Cap< void > cap, l4_utcb_t *utcb=l4_utcb()) noexcept`

Release object capability.

- `l4_msgtag_t cap_valid (Cap< void > const &cap, l4_utcb_t *utcb=l4_utcb()) noexcept`

Check whether a capability is present (refers to an object).

- `l4_msgtag_t cap_equal (Cap< void > const &cap_a, Cap< void > const &cap_b, l4_utcb_t *utcb=l4_utcb()) noexcept`

Test whether two capabilities point to the same object with the same permissions (only considering selected permissions).

- `l4_msgtag_t add_ku_mem (l4_fpage_t *fpage, l4_utcb_t *utcb=l4_utcb()) noexcept`

Add kernel-user memory.

Public Member Functions inherited from `L4::Kobject`

- `l4_msgtag_t dec_refcnt (l4_mword_t diff, l4_utcb_t *utcb=l4_utcb())`

Decrement the in kernel reference counter for the object.

Additional Inherited Members

Protected Types inherited from

`L4::Kobject_t< Task, Kobject, L4_PROTO_TASK, Type_info::Demand_t< 2 > >`

- typedef `Task` **Class**

The target interface type (inheriting from `Kobject_t`).

- typedef `Typeid::Iface< PROTO, Task > __Iface`

The interface description for the derived class.

- typedef `Typeid::Merge_list< Typeid::Iface_list< __Iface >, typename Kobject::__Iface_list > __Iface_list`

The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from

`L4::Kobject_t< Task, Kobject, L4_PROTO_TASK, Type_info::Demand_t< 2 > >`

- `L4::Cap< Class > c () const noexcept`

Get the capability to ourselves.

Protected Member Functions inherited from `L4::Kobject`

- `l4_cap_idx_t cap () const noexcept`

Return capability selector.

Static Protected Member Functions inherited from

`L4::Kobject_t< Task, Kobject, L4_PROTO_TASK, Type_info::Demand_t< 2 > >`

- static void `__check_protocols__ () noexcept`

Helper to check for protocol conflicts.

16.205.1 Detailed Description

C++ interface of the [Task](#) kernel object, see [Task](#) for the C interface.

The [L4::Task](#) class represents a combination of the address spaces provided by the [L4Re](#) micro kernel. A task consists of at least a memory address space and an object address space. On IA32 there is also an IO-port address space associated with an [L4::Task](#).

[L4::Task](#) objects are created using the [L4::Factory](#) interface.

Include File

```
#include <l4/sys/task>
```

Definition at line 33 of file [task](#).

16.205.2 Member Function Documentation

16.205.2.1 add_ku_mem()

```
l4_msgtag_t L4::Task::add_ku_mem (
    l4_fpage_t * fpage,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
```

Add kernel-user memory.

Parameters

<i>in, out</i>	<i>fpage</i>	Flexpage describing the virtual area the memory goes to. On systems without MMU, the flexpage is adjusted to reflect the actually allocated physical address.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

Syscall return tag

Kernel-user memory (ku_mem) is memory that is shared between the kernel and user-space. It is needed for the UTCB area of threads (see [L4::Thread::Attr::bind\(\)](#)) and for (extended) vCPU state. Note that existing kernel-user memory cannot be unmapped or mapped somewhere else.

Note

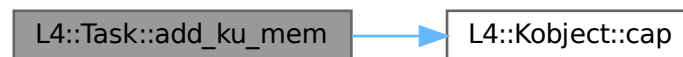
The amount of kernel-user memory that can be allocated at once is limited by the used kernel implementation. The minimum allocatable amount is one page ([L4_PAGESIZE](#)). A portable implementation should not depend on allocations greater than 16KiB to succeed.

This function is only guaranteed to work on [L4::Task](#) objects. It might or might not work on [L4::Vm](#) objects or on [L4Re::Dma_space](#) objects but there is no practical use for adding kernel-user memory to [L4::Vm](#) objects or to [L4Re::Dma_space](#) objects.

Definition at line 281 of file [task](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:

**16.205.2.2 cap_equal()**

```

l4_msgtag_t L4::Task::cap_equal (
    Cap< void > const & cap_a,
    Cap< void > const & cap_b,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Test whether two capabilities point to the same object with the same permissions (only considering selected permissions).

Parameters

<i>cap_a</i>	Capability selector for the first capability to compare.
<i>cap_b</i>	Capability selector for the second capability to compare.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Return values

<i>l4_msgtag_t::label()</i> = 1	The compared capabilities point to the same object with same considered permission.
<i>l4_msgtag_t::label()</i> = 0	The compared capabilities do not point to the same object or differ in the considered permission.

- For [L4::lpc_gate](#) objects, only the permissions [L4_CAP_FPAGE_W](#), [L4_CAP_FPAGE_S](#), and [L4_FPAGE_C_OBJ_RIGHT1](#) are considered for the comparison. Differences in other permissions are ignored.

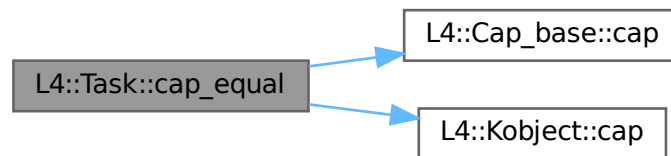
- For other objects, only the permissions [L4_CAP_FPAGE_W](#) and [L4_CAP_FPAGE_S](#) are considered for the comparison. Differences in other permissions are ignored.

Note that having the [L4_CAP_FPAGE_R](#) permission is implicit in possessing the capability.

Definition at line 251 of file [task](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



16.205.2.3 cap_valid()

```

l4_msgtag_t L4::Task::cap_valid (
    Cap< void > const & cap,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Check whether a capability is present (refers to an object).

Parameters

<i>cap</i>	Valid capability to check for presence.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Return values

<i>l4_msgtag_t::label()</i> > 0	Capability is present (refers to an object).
<i>l4_msgtag_t::label()</i> == 0	No capability present (void object).

A capability is considered present when it refers to an existing kernel object.

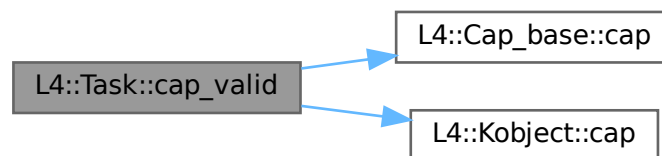
Precondition

`cap` must be a valid capability (i.e. `cap.is_valid() == true`). If you are unsure about the validity of your capability use [L4::Cap.validate\(\)](#) instead.

Definition at line 222 of file [task](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:

**16.205.2.4 delete_obj()**

```

l4_msgtag_t L4::Task::delete_obj (
    L4::Cap< void > obj,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Release capability and delete object.

Parameters

<i>obj</i>	Capability index of the object to delete.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

Syscall return tag

If `obj` has the delete permission, initiates the deletion of the object. This implies that all capabilities for that object are gone afterwards. However, kernel-internally, objects are not destroyed until all other kernel objects holding a reference to it drop the reference. Hence, quota used by that object might not be freed immediately.

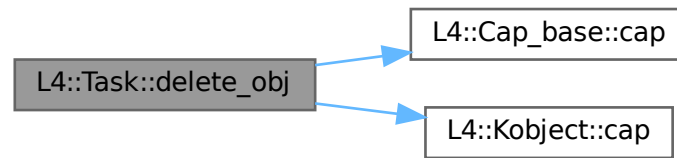
If `obj` does not have the delete permission, no error will be reported and only the capability `obj` is removed. (Note that, depending on the object's reference counter, this might still imply initiation of deletion.)

This operation is equivalent to [unmap\(\)](#) with [L4_FP_DELETE_OBJ](#) flag.

Definition at line 180 of file [task](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



16.205.2.5 map()

```

l4_msgtag_t L4::Task::map (
    Cap< Task > const & src_task,
    l4_fpage_t const & snd_fpage,
    l4_umword_t snd_base,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Map resources available in the source task to a destination task.

Parameters

<i>src_task</i>	Capability selector of the source task.
<i>snd_fpage</i>	Send flexpage that describes an area in the address space or object space of the source task.
<i>snd_base</i>	Send base that describes an offset in the receive window of the destination task. The lower bits contain additional map control flags (see l4_fpage_cacheability_opt_t for memory mappings, L4_obj_fpage_ctl for object mappings, and L4_MAP_ITEM_GRANT ; also see l4_map_control() and l4_map_obj_control()).
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

Syscall return tag. The function [l4_error\(\)](#) shall be used to test if the map operation was successful.

Return values

<i>L4_EOK</i>	Operation successful (but see notes below).
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i>-L4_EINVAL</i>	Invalid source task capability.
<i>-L4_IPC_SEMAPFAILED</i>	The map operation failed due to limited quota.

Precondition

The invoked [Task](#) capability must have the permission [L4_CAP_FPAGE_W](#).

This method allows for asynchronous transfer of capabilities, memory mappings, and IO-port mappings (on IA32) from one task to another. The destination task is the task referenced by the capability on which the map is invoked, and the receive window is the whole address space of that task. By specifying proper rights in the `snd_fpage` and `snd_base`, it is possible to remove rights during transfer.

Note

If the send flexpage is of type [L4_FPAGE_OBJ](#), the [L4_CAP_FPAGE_S](#) right is removed from the transferred capability unless both the source and destination task capabilities possess the [L4_CAP_FPAGE_S](#) right themselves.

Even with [l4_error\(\)](#) returning `L4_EOK` there might be cases where not all pages of the send flexpage were mapped respectively granted to the destination task, for instance, if the corresponding mapping in the destination task does already exist.

For more information on spaces and mappings, see [Spaces and Mappings](#). The flexpage API is described in more detail at [Flexpages](#).

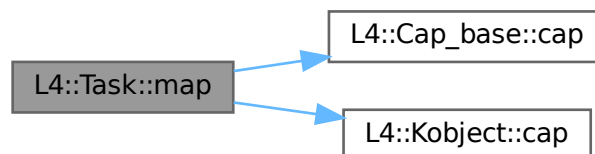
Note

For peculiarities when using grant, see [L4_MAP_ITEM_GRANT](#).

Definition at line 85 of file [task](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:

**16.205.2.6 release_cap()**

```

l4_msgtag_t L4::Task::release_cap (
    L4::Cap< void > cap,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Release object capability.

Parameters

<i>cap</i>	Capability selector of the object to release.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

Syscall return tag.

This operation unmaps the capability from `this` task. This operation is equivalent to unmapping a single object capability by specifying all object rights as unmap mask.

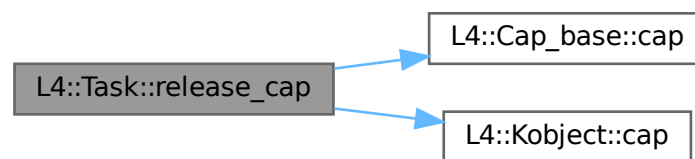
Note

If the reference counter of the kernel object referenced by [cap](#) goes down to zero, the deletion of the object is initiated. Objects are not destroyed until all other kernel objects holding a reference to it drop the reference.

Definition at line 201 of file [task](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:

**16.205.2.7 unmap()**

```

l4_msgtag_t L4::Task::unmap (
    l4_fpage_t const & fpage,
    l4_umword_t map_mask,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Revoke rights from the task.

Parameters

<i>fpage</i>	Flexpage that describes an area in one capability space of <code>this</code> task and the rights to revoke.
<i>map_mask</i>	Unmap mask, see l4_unmap_flags_t
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

Syscall return tag

This method allows to revoke rights from the destination task. The rights to revoke are specified in the flexpage, see [l4_fpage_rights\(\)](#). For a flexpage describing IO ports or memory, it also revokes rights from all the tasks that got the rights delegated from the destination task (i.e., this operation does a recursive rights revocation). The capability is unmapped if certain rights are specified, see below for details. It is guaranteed that the rights revocation is completed before this function returns.

Note that this function cannot be used to revoke the reference counting permission (see [L4_FPAGE_C_REF_CNT](#)) or the IPC-gate server permission (see [L4_FPAGE_C_IPCGATE_SVR](#)) from object capabilities.

It depends on the platform and the object type which rights need to be specified in the `rights` field of `fpage` to unmap a capability:

- An object capability is unmapped if and only if the [L4_CAP_FPAGE_R](#) right bit is set.
- An IO port is unmapped if and only if any right bit is set.
- Memory is unmapped if and only if the [L4_FPAGE_RO](#) right bit is set.

Note

Depending on the page-table features supported by the hardware, revocation of certain rights from a memory capability can be a no-op (i.e., the rights are not revoked). Further, revocation of certain rights may grant other rights which were not present before. For instance, on an architecture without support for NX, revoking X does nothing. For another example, revoking only X from an execute-only page grants read permission (because the mapping remains present in the page table).

If the reference counter of a kernel object referenced in `fpage` goes down to zero (as a result of deleting capabilities), the deletion of the object is initiated. Objects are not destroyed until all other kernel objects holding a reference to it drop the reference.

Definition at line [135](#) of file [task](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:

**16.205.2.8 unmap_batch()**

```

l4_msgtag_t L4::Task::unmap_batch (
    l4_fpage_t const * fpages,
    unsigned num_fpages,
    l4_umword_t map_mask,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Revoke rights from a task.

Parameters

<i>fpages</i>	An array of flexpages. Each item describes an area in one capability space of <code>this</code> task.
<i>num_fpages</i>	Number of fpages in the <code>fpages</code> array.
<i>map_mask</i>	Unmap mask, see l4_unmap_flags_t .
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Revoke rights for an array of flexpages, see [unmap](#) for details.

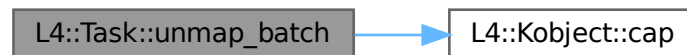
Precondition

The caller needs to take care that `num_fpages` is not bigger than `L4_UTCB_GENERIC_DATA_SIZE - 2`.

Definition at line 154 of file [task](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

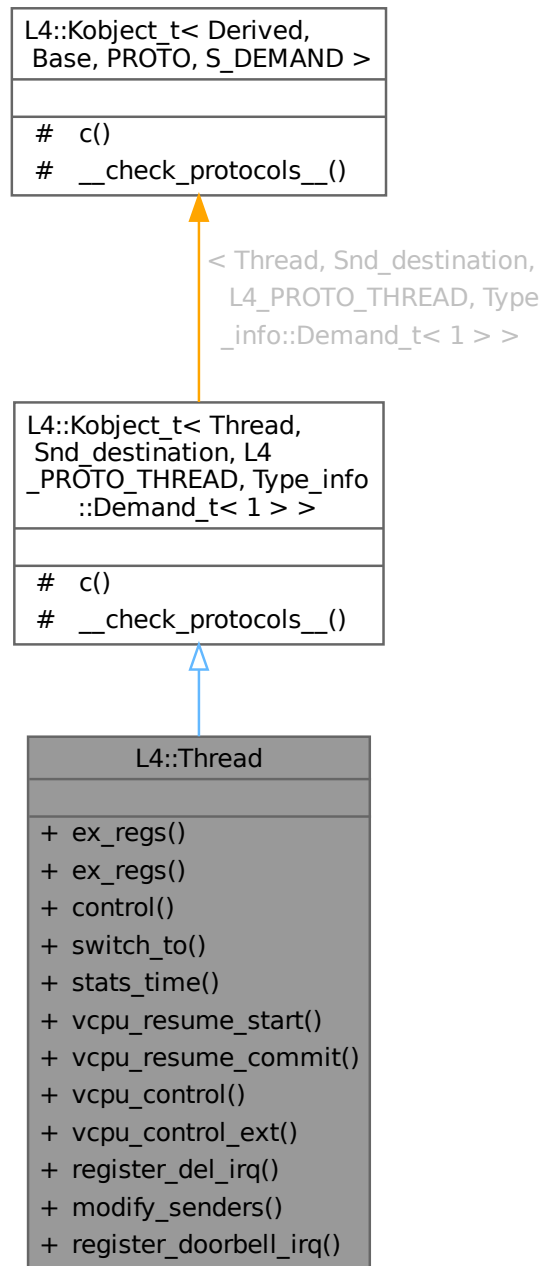
- [l4/sys/task](#)

16.206 L4::Thread Class Reference

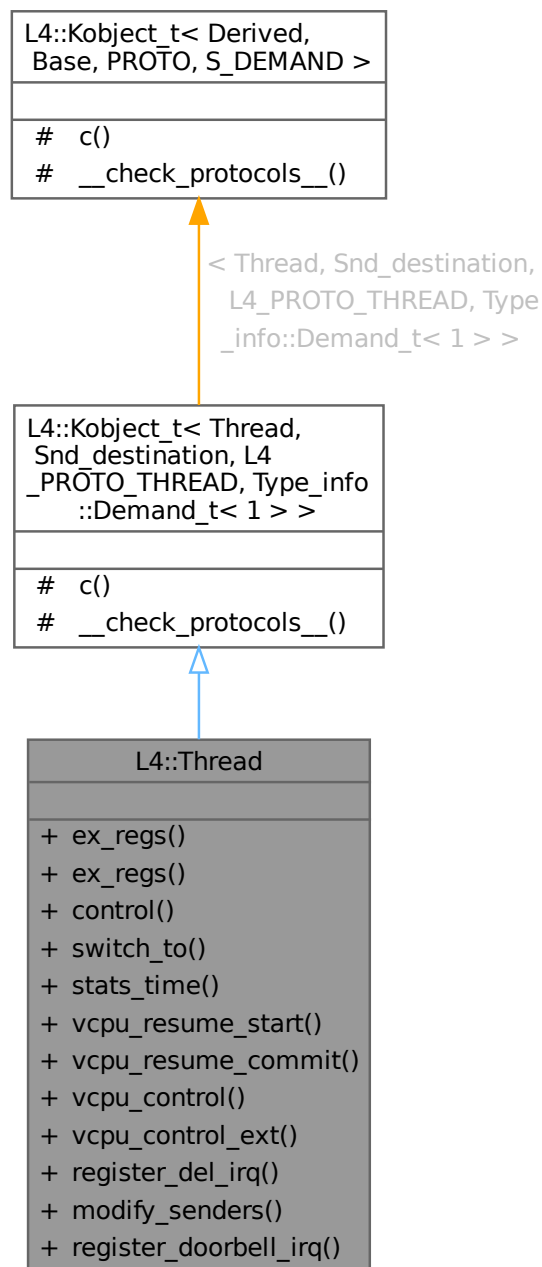
C++ [L4](#) kernel thread interface, see [Thread](#) for the C interface.

```
#include <thread>
```

Inheritance diagram for L4::Thread:



Collaboration diagram for L4::Thread:



Data Structures

- class [Attr](#)
Thread attributes used for control().
- class [Modify_senders](#)
Class wrapping a list of rules which modify the sender label of IPC messages inbound to this thread.

Public Member Functions

- [l4_msgtag_t ex_regs](#) ([l4_addr_t ip](#), [l4_addr_t sp](#), [l4_umword_t flags](#), [l4_utcb_t *utcb=l4_utcb\(\)](#)) noexcept
Exchange basic thread registers.
- [l4_msgtag_t ex_regs](#) ([l4_addr_t *ip](#), [l4_addr_t *sp](#), [l4_umword_t *flags](#), [l4_utcb_t *utcb=l4_utcb\(\)](#)) noexcept
Exchange basic thread registers and return previous values.
- [l4_msgtag_t control](#) ([Attr](#) const &attr) noexcept
Commit the given thread-attributes object.
- [l4_msgtag_t switch_to](#) ([l4_utcb_t *utcb=l4_utcb\(\)](#)) noexcept
Switch execution to this thread.
- [l4_msgtag_t stats_time](#) ([l4_kernel_clock_t *us](#), [l4_utcb_t *utcb=l4_utcb\(\)](#)) noexcept
Get consumed time of thread in us.
- [l4_msgtag_t vcpu_resume_start](#) ([l4_utcb_t *utcb=l4_utcb\(\)](#)) noexcept
Resume from vCPU asynchronous IPC handler, start.
- [l4_msgtag_t vcpu_resume_commit](#) ([l4_msgtag_t tag](#), [l4_utcb_t *utcb=l4_utcb\(\)](#)) noexcept
Resume from vCPU asynchronous IPC handler, commit.
- [l4_msgtag_t vcpu_control](#) ([l4_addr_t vcpu_state](#), [l4_utcb_t *utcb=l4_utcb\(\)](#)) noexcept
Enable the vCPU feature for the thread.
- [l4_msgtag_t vcpu_control_ext](#) ([l4_addr_t ext_vcpu_state](#), [l4_utcb_t *utcb=l4_utcb\(\)](#)) noexcept
Enable the extended vCPU feature for the thread.
- [l4_msgtag_t register_del_irq](#) ([Cap< l4r_irq > irq](#), [l4_utcb_t *u=l4_utcb\(\)](#)) noexcept
Register an IRQ that will trigger upon deletion events.
- [l4_msgtag_t modify_senders](#) ([Modify_senders](#) const &todo) noexcept
Apply sender modification rules.
- [l4_msgtag_t register_doorbell_irq](#) ([Cap< l4r_irq > irq](#), [l4_utcb_t *u=l4_utcb\(\)](#)) noexcept
Register an IRQ that will trigger when a forwarded virtual interrupt is pending.

Additional Inherited Members

Protected Types inherited from

[L4::Kobject_t< Thread, Snd_destination, L4_PROTO_THREAD, Type_info::Demand_t< 1 > >](#)

- typedef [Thread](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef [Typeid::Iface< PROTO, Thread >](#) **__Iface**
The interface description for the derived class.
- typedef [Typeid::Merge_list< Typeid::Iface_list< __Iface >, typename Snd_destination::__Iface_list >](#) **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from

[L4::Kobject_t< Thread, Snd_destination, L4_PROTO_THREAD, Type_info::Demand_t< 1 > >](#)

- [L4::Cap< Class > c](#) () const noexcept
Get the capability to ourselves.

Static Protected Member Functions inherited from

[L4::Kobject_t< Thread, Snd_destination, L4_PROTO_THREAD, Type_info::Demand_t< 1 > >](#)

- static void [__check_protocols](#) () noexcept
Helper to check for protocol conflicts.

16.206.1 Detailed Description

C++ [L4](#) kernel thread interface, see [Thread](#) for the C interface.

The [Thread](#) class defines a thread of execution in the [L4](#) context. Usually user-level and kernel threads are mapped 1:1 to each other. [Thread](#) kernel objects are created using a factory, see the [L4::Factory](#) API ([L4::Factory::create\(\)](#)).

Amongst other things an [L4::Thread](#) encapsulates:

- CPU state
 - General-purpose registers
 - Program counter
 - Stack pointer
- FPU state
- Scheduling parameters, see the [L4::Scheduler](#) API
- Execution state
 - Blocked, Runnable, Running

[Thread](#) objects provide an API for

- [Thread](#) configuration and manipulation
- [Thread](#) switching.

On ARM newly created threads run in EL0 by default and the exception level can be changed there with [ex_regs\(\)](#).

Include File

```
#include <l4/sys/thread>
```

For the C interface see the [Thread](#) API. For more elaborated documentation on the vCPU feature see [vCPU API](#).

Definition at line 52 of file [thread](#).

16.206.2 Member Function Documentation

16.206.2.1 control()

```
l4_msgtag_t L4::Thread::control (
    Attr const & attr) [inline], [noexcept]
```

Commit the given thread-attributes object.

Parameters

<i>attr</i>	the attribute object to commit to the thread.
-------------	---

Returns

Syscall return tag containing one of the following return codes.

Return values

<i>L4_EOK</i>	Operation successful.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i>-L4_EINVAL</i>	Malformed thread-attributes.

Precondition

The invoked [Thread](#) capability must have the permission [L4_CAP_FPAGE_S](#). When using `#Attr::bind()`, also the respective [Task](#) capability must have the permission [L4_CAP_FPAGE_S](#).

Definition at line 243 of file [thread](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



16.206.2.2 `ex_regs()` [1/2]

```

l4_msgtag_t L4::Thread::ex_regs (
    l4_addr_t * ip,
    l4_addr_t * sp,
    l4_umword_t * flags,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Exchange basic thread registers and return previous values.

Parameters

in, out	<i>ip</i>	New instruction pointer, use <code>~0UL</code> to leave the instruction pointer unchanged, return previous instruction pointer.
in, out	<i>sp</i>	New stack pointer, use <code>~0UL</code> to leave the stack pointer unchanged, returns previous stack pointer.
in, out	<i>flags</i>	Ex-regs flags, see L4_thread_ex_regs_flags , return previous CPU flags of the thread.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

System call return tag. [out] parameters are only valid if the function returns successfully. Use [l4_error\(\)](#) to check.

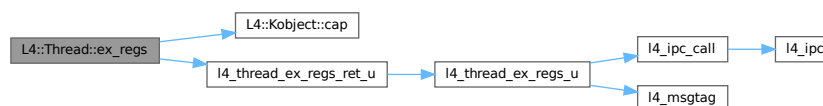
This method allows to manipulate and start a thread. The basic functionality is to set the instruction pointer and the stack pointer of a thread. Additionally, this method allows also to cancel ongoing IPC operations and to force the thread to raise an artificial exception (see `flags`). If the thread is in an IPC operation or if [L4_THREAD_EX_REGS_TRIGGER_EXCEPTION](#) forces an IPC then changes in IP and SP take effect directly after returning from this IPC. On ARM this method allows to change the exception level, see [L4_thread_ex_regs_flags_arm](#) and [L4_thread_ex_regs_flags_arm64](#).

The thread is started using [L4::Scheduler::run_thread\(\)](#). However, if at the time [L4::Scheduler::run_thread\(\)](#) is called, the instruction pointer of the thread is invalid, a later call to [ex_regs\(\)](#) with a valid instruction pointer might start the thread.

Definition at line 119 of file [thread](#).

References [L4::Kobject::cap\(\)](#), and [l4_thread_ex_regs_ret_u\(\)](#).

Here is the call graph for this function:

16.206.2.3 `ex_regs()` [2/2]

```

l4_msgtag_t L4::Thread::ex_regs (
    l4_addr_t ip,
    l4_addr_t sp,
    l4_umword_t flags,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]

```

Exchange basic thread registers.

Parameters

<i>ip</i>	New instruction pointer, use ~0UL to leave the instruction pointer unchanged.
<i>sp</i>	New stack pointer, use ~0UL to leave the stack pointer unchanged.
<i>flags</i>	Ex-regs flags, see L4_thread_ex_regs_flags .
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

System call return tag.

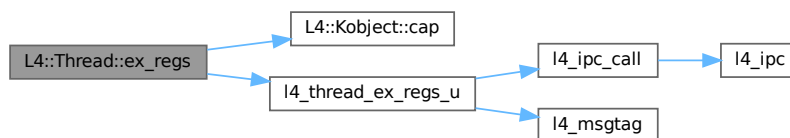
This method allows to manipulate and start a thread. The basic functionality is to set the instruction pointer and the stack pointer of a thread. Additionally, this method allows also to cancel ongoing IPC operations and to force the thread to raise an artificial exception (see `flags`). If the thread is in an IPC operation or if `L4_THREAD_EX_REGS_TRIGGER_EXCEPTION` forces an IPC then changes in IP and SP take effect directly after returning from this IPC. On ARM this method allows to change the exception level, see `L4_thread_ex_regs_flags_arm` and `L4_thread_ex_regs_flags_arm64`.

The thread is started using `L4::Scheduler::run_thread()`. However, if at the time `L4::Scheduler::run_thread()` is called, the instruction pointer of the thread is invalid, a later call to `ex_regs()` with a valid instruction pointer might start the thread.

Definition at line 84 of file `thread`.

References `L4::Kobject::cap()`, and `l4_thread_ex_regs_u()`.

Here is the call graph for this function:

**16.206.2.4 modify_senders()**

```
l4_msgtag_t L4::Thread::modify_senders (
    Modify_senders const & todo) [inline], [noexcept]
```

Apply sender modification rules.

Parameters

<i>todo</i>	Prepared sender modification rules.
-------------	-------------------------------------

Returns

System call return tag.

The modification rules are applied to all IPCs to the thread (whether directly or by IPC gate) that are already in flight, that is that the sender is already blocking on.

See `Modify_senders` for a detailed description when applying sender modification rules is required.

Note

Modifying the senders of a thread running on a different CPU core is not supported.

To ensure that no in-flight senders are missed, either the thread itself must execute `modify_senders`, or the thread executing the `modify_senders` must synchronize with the target thread. This synchronization must ensure the following:

1. Before `modify_senders` is executed the target thread must execute at least shortly (so that pending DRQs are handled).
2. The target thread must pause its IPC dispatch, until `modify_senders` is completed. In other words, the target thread must not be receive ready, because otherwise an IPC message with an unmodified label can be transferred to its UTCB or vCPU state.

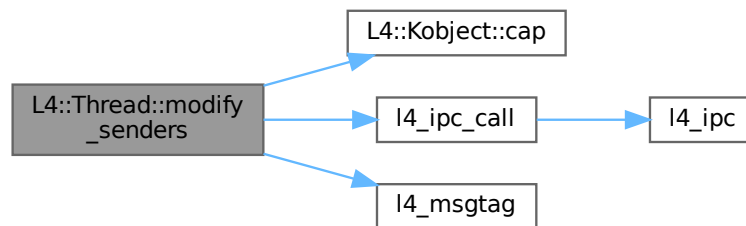
See also

[l4_thread_modify_sender_commit\(\)](#)

Definition at line 525 of file [thread](#).

References [L4::Kobject::cap\(\)](#), [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), [l4_msgtag\(\)](#), and [L4_PROTO_THREAD](#).

Here is the call graph for this function:

**16.206.2.5 register_del_irq()**

```

l4_msgtag_t L4::Thread::register_del_irq (
    Cap< Irq > irq,
    l4_utcb_t * u = l4_utcb()) [inline], [noexcept]

```

Register an IRQ that will trigger upon deletion events.

Parameters

<i>irq</i>	Capability selector for the IRQ object to be triggered.
<i>u</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

System call return tag containing the return code.

Return values

-L4_BUSY	A deletion IRQ is already bound to this thread.
-L4_EPERM	Insufficient permissions; see precondition.

Precondition

The capability `irq` must have the permission [L4_CAP_FPAGE_W](#).

In case the `irq` is already bound to an interrupt source, it is unbound first. When `irq` is deleted, it will be deregistered first. A registered deletion [irq](#) can only be deregistered by deleting the [irq](#) or the thread.

List of deletion events:

- deletion of one or several IPC gates bound to this thread.

When the deletion event is delivered, there is no indication about which IPC gate was deleted.

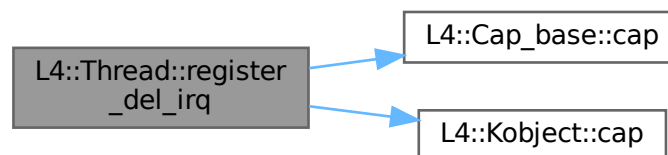
See also

[l4_thread_register_del_irq](#)

Definition at line [427](#) of file [thread](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



16.206.2.6 register_doorbell_irq()

```

l4_msgtag_t L4::Thread::register_doorbell_irq (
    Cap< Irq > irq,
    l4_utcb_t * u = l4_utcb())  [inline], [noexcept]
  
```

Register an IRQ that will trigger when a forwarded virtual interrupt is pending.

Parameters

<i>irq</i>	Capability selector for the IRQ object to be triggered.
<i>u</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

System call return tag containing the return code.

Return values

-L4_BUSY	A doorbell IRQ is already bound to this thread.
-L4_EPERM	Insufficient permissions; see precondition.

Precondition

The capability `irq` must have the permission [L4_CAP_FPAGE_W](#).

See [Irq::bind_vcpu\(\)](#) for more details about how interrupts can be forwarded directly by the kernel to extended vCPU user mode.

In case the `irq` is already bound to an interrupt source, it is unbound first. When `irq` is deleted, it will be deregistered first. A registered doorbell [irq](#) can only be deregistered by deleting the [irq](#) or the thread.

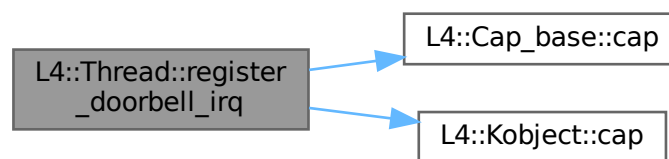
See also

[l4_thread_register_doorbell_irq](#)

Definition at line 553 of file [thread](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:

**16.206.2.7 stats_time()**

```

l4_msgtag_t L4::Thread::stats_time (
    l4_kernel_clock_t * us,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Get consumed time of thread in us.

Parameters

out	us	Consumed time in μ s.
	utcb	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

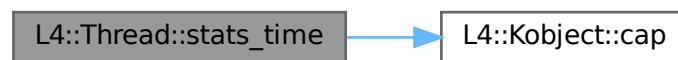
Returns

Syscall return tag.

Definition at line [264](#) of file [thread](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:

**16.206.2.8 switch_to()**

```

l4_msgtag_t L4::Thread::switch_to (
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Switch execution to this thread.

Parameters

utcb	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .
------	--

Note

The current time slice is inherited to this thread.

Definition at line [253](#) of file [thread](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



16.206.2.9 vcpu_control()

```
l4_msgtag_t L4::Thread::vcpu_control (
    l4_addr_t vcpu_state,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
```

Enable the vCPU feature for the thread.

Parameters

<i>vcpu_state</i>	A virtual address pointing to a l4_vcpu_state_t . It must be a valid kernel-user-memory address (see L4::Task::add_ku_mem()).
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

Syscall return tag.

This function enables the vCPU feature of `this` thread

The kernel-user memory starting at `vcpu_state` must be at least 128-byte aligned and must cover the size of [l4_vcpu_state_t](#).

The asynchronous IPC handling is described at [vCPU API](#).

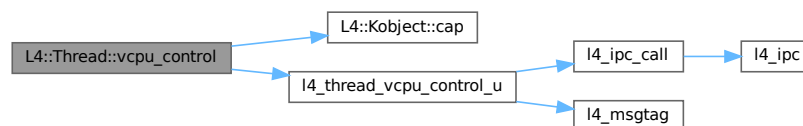
Note

Disabling of the vCPU feature is optional and currently not supported.

Definition at line 358 of file [thread](#).

References [L4::Kobject::cap\(\)](#), and [l4_thread_vcpu_control_u\(\)](#).

Here is the call graph for this function:



16.206.2.10 vcpu_control_ext()

```
l4_msgtag_t L4::Thread::vcpu_control_ext (
    l4_addr_t ext_vcpu_state,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
```

Enable the extended vCPU feature for the thread.

Parameters

<i>ext_vcpu_state</i>	The virtual address where the kernel shall store the vCPU state in case of vCPU exits. The address must be a valid kernel-user-memory address (see L4::Task::add_ku_mem()).
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

Syscall return tag.

The extended vCPU feature allows the use of hardware-virtualization features such as Intel's VT-x (VMX) or AMD's AMD-V (SVM).

This function enables the extended vCPU feature of `this` thread. Enabling the extended vCPU feature also enables the vCPU feature.

The kernel-user memory area starting at `ext_vcpu_state` must be at least 4 KiB aligned and must cover a size of `L4_PAGESIZE`. It includes the data of [l4_vcpu_state_t](#) at offset 0, the extended vCPU state at offset `L4_VCPU_OFFSET_EXT_STATE`, and, on some platforms, the extended vCPU information at offset `L4_VCPU_OFFSET_EXT_INFOS`.

On Intel's VT-x (VMX), the extended vCPU state is [l4_vm_vmx_vcpu_vmcs_t](#) and the extended vCPU information is [l4_vm_vmx_vcpu_infos_t](#). Furthermore, the extended vCPU state needs to be associated with a vCPU context (see [l4_vm_vmx_set_hw_vmcs\(\)](#)).

On AMD's AMD-V (SVM), the extended vCPU state is [l4_vm_svm_vmcb_t](#).

Note

Enabling the extended vCPU feature for a thread running on a different CPU core is currently not supported.

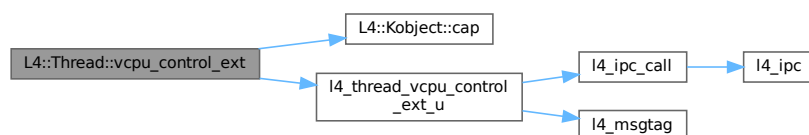
Disabling of the extended vCPU feature is currently not supported.

Upgrading from non-extended vCPU feature to extended vCPU feature is currently not supported.

Definition at line 398 of file [thread](#).

References [L4::Kobject::cap\(\)](#), and [l4_thread_vcpu_control_ext_u\(\)](#).

Here is the call graph for this function:



16.206.2.11 vcpu_resume_commit()

```

l4_msgtag_t L4::Thread::vcpu_resume_commit (
    l4_msgtag_t tag,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]

```

Resume from vCPU asynchronous IPC handler, commit.

Parameters

<i>tag</i>	Tag to use, returned by l4_thread_vcpu_resume_start() .
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

Syscall return tag containing one of the following return codes.

Return values

<i>0</i>	Indicates a VM exit, provided that <i>thread</i> is in extended vCPU mode with virtual interrupts cleared.
<i>1</i>	Indicates an incoming IPC message, provided that the <i>thread</i> is in extended vCPU mode with virtual interrupts cleared.
<i>-L4_EPERM</i>	The user task capability set in the vCPU state is missing the L4_CAP_FPAGE_S right. On Intel's VT-x (VMX): The vCPU context capability set in the extended vCPU state is missing the L4_CAP_FPAGE_S right.
<i>-L4_ENOENT</i>	The user task capability set in the vCPU state is invalid.
<i>-L4_EINVAL</i>	<i>thread</i> is not the current running thread, or does not have the vCPU feature enabled. On Intel's VT-x (VMX): No vCPU context associated with the extended vCPU state.
<i>-L4_EBUSY</i>	On Intel's VT-x (VMX): The vCPU context associated with the extended vCPU state is already active on a different CPU.
<i>-L4_ENODEV</i>	On Intel's VT-x (VMX): The vCPU context associated with the extended vCPU state cannot be initialized or activated.
<i><0</i>	A supplied mapping failed.

All flexpages in the UTCB (added with [l4_sndfpage_add\(\)](#) after [l4_thread_vcpu_resume_start\(\)](#)) are unconditionally mapped into the user task configured in the vCPU state.

To resume into another address space, the capability to the target [Task](#) (or [L4::Vm](#)) must be set in [l4_vcpu_state_t::user_task](#) together with [L4_VCPU_F_USER_MODE](#). The capability selector must have all lower bits clear (see [L4_CAP_MASK](#)). The kernel adds the [L4_SYSF_SEND](#) flag there to indicate that the capability has been referenced in the kernel. Consecutive resumes will not reference the task capability again until all lower bits are cleared again. To release a task use a different task capability or use an invalid capability with the [L4_SYSF_REPLY](#) flag set.

The asynchronous IPC handling is described at [vCPU API](#).

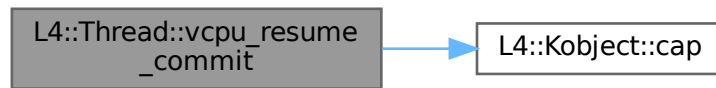
See also

[l4_thread_vcpu_resume_commit](#)

Definition at line 334 of file [thread](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



16.206.2.12 vcpu_resume_start()

```
l4_msgtag_t L4::Thread::vcpu_resume_start (
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
```

Resume from vCPU asynchronous IPC handler, start.

Parameters

<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .
-------------	--

Returns

Message tag to be used for [l4_sndfpage_add\(\)](#) and [l4_thread_vcpu_resume_commit\(\)](#)

The vCPU resume functionality is split in multiple functions to allow the specification of additional send-flexpages using [l4_sndfpage_add\(\)](#).

The asynchronous IPC handling is described at [vCPU API](#).

See also

[l4_thread_vcpu_resume_start](#)

Definition at line 283 of file [thread](#).

The documentation for this class was generated from the following file:

- [l4/sys/thread](#)

16.207 L4::Thread::Attr Class Reference

[Thread](#) attributes used for [control\(\)](#).

```
#include <thread>
```

Collaboration diagram for L4::Thread::Attr:

L4::Thread::Attr
<ul style="list-style-type: none"> + Attr() + pager() + pager() + exc_handler() + exc_handler() + bind() + alien()

Public Member Functions

- [Attr](#) ([l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept
Create a thread-attribute object with the given UTCB.
- void [pager](#) ([Cap](#)< void > const &pager) noexcept
Set the pager capability selector.
- [Cap](#)< void > [pager](#) () noexcept
Get the capability selector used for page-fault messages.
- void [exc_handler](#) ([Cap](#)< void > const &exc_handler) noexcept
Set the exception-handler capability selector.
- [Cap](#)< void > [exc_handler](#) () noexcept
Get the capability selector used for exception messages.
- void [bind](#) ([l4_utcb_t](#) *thread_utcb, [Cap](#)< [Task](#) > const &task) noexcept
Bind the thread to a task.
- void [alien](#) (int on) noexcept
Enable alien mode.

Friends

- class **L4::Thread**

16.207.1 Detailed Description

[Thread](#) attributes used for [control\(\)](#).

This class is responsible for initializing various attributes of a thread in a UTCB for the [control\(\)](#) method.

Note

Instantiation of this class starts the preparation of the UTCB. Do not invoke any non-Attr functions between the instantiation and the call to [L4::Thread::control\(\)](#).

See also

[Thread control](#) for some more details.

Definition at line 137 of file [thread](#).

16.207.2 Constructor & Destructor Documentation

16.207.2.1 Attr()

```
L4::Thread::Attr::Attr (
    l4_utcb_t * utcb = l4_utcb()) [inline], [explicit], [noexcept]
```

Create a thread-attribute object with the given UTCB.

Parameters

<i>utcb</i>	The UTCB to use for the later L4::Thread::control() function. Usually this is the UTCB of the calling thread. See l4_utcb() .
-------------	---

Definition at line 151 of file [thread](#).

16.207.3 Member Function Documentation

16.207.3.1 alien()

```
void L4::Thread::Attr::alien (
    int on) [inline], [noexcept]
```

Enable alien mode.

Parameters

<i>on</i>	Boolean value defining the state of the feature.
-----------	--

For a thread in alien mode the kernel produces just an exception IPC for each IPC and exception caused by the alien thread instead of handling these events regularly. (Page faults of alien threads and interrupts occurring while the alien thread is running are always handled regularly.) While the alien thread is blocking, the exception handler can inspect and modify the state of the alien thread and potentially also the system call arguments. If the exception handler replies with [L4_PROTO_ALLOW_SYSCALL](#) as message tag, the kernel handles the next IPC or exception of the alien thread in a regular way. If the exception handler leaves certain thread state unchanged (in particular the instruction pointer), this will be the IPC or exception that caused the call of the exception handler. For a regularly processed IPC or exception of the alien thread the kernel also performs an exception IPC on kernel exit.

This feature can be used to attach a debugger to a thread and trace all object invocations and their results. It could also be used to handle other systems that use the same syscall instruction as [L4Re](#).

Definition at line 224 of file [thread](#).

16.207.3.2 bind()

```
void L4::Thread::Attr::bind (
    l4_utcb_t * thread_utcb,
    Cap< Task > const & task) [inline], [noexcept]
```

Bind the thread to a task.

Parameters

<i>thread_utcb</i>	The thread's UTCB address within the task it shall be bound to. The address must be aligned (architecture dependent; at least word aligned) and it must point to at least L4_UTCB_OFFSET bytes of kernel-user memory.
<i>task</i>	The task the thread shall be bound to.

Precondition

The thread must not be bound to a task yet.

The capability `task` must have the permission [L4_CAP_FPAGE_S](#), otherwise the later call to [L4::Thread::control\(\)](#) with this [Attr](#) object will fail with [L4_EPERM](#).

A thread may execute code in the context of a task if and only if the thread is bound to the task. To actually start execution, [L4::Thread::ex_regs\(\)](#) needs to be used. Execution in the context of the task means that the code has access to all the task's resources (and only those). The executed code itself must be one of those resources. A thread can be bound at most once to a task.

Note

The UTCBs of different threads in the same task should not overlap in order to prevent data corruption.

Definition at line [218](#) of file [thread](#).

16.207.3.3 exc_handler() [1/2]

```
Cap< void > L4::Thread::Attr::exc_handler () [inline], [noexcept]
```

Get the capability selector used for exception messages.

Returns

The capability selector used to send exception messages. The selector is valid in the task the thread is bound to.

Definition at line [189](#) of file [thread](#).

16.207.3.4 exc_handler() [2/2]

```
void L4::Thread::Attr::exc_handler (
    Cap< void > const & exc_handler) [inline], [noexcept]
```

Set the exception-handler capability selector.

Parameters

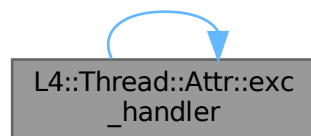
<i>exc_handler</i>	The capability selector that shall be used for exception messages. This capability selector must be valid within the task the thread is bound to.
--------------------	---

Definition at line 180 of file [thread](#).

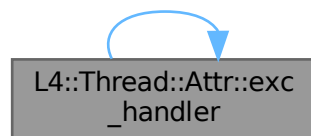
References [exc_handler\(\)](#).

Referenced by [exc_handler\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.207.3.5 pager() [1/2]

```
Cap< void > L4::Thread::Attr::pager () [inline], [noexcept]
```

Get the capability selector used for page-fault messages.

Returns

The capability selector used to send page-fault messages. The selector is valid in the task the thread is bound to.

Definition at line 170 of file [thread](#).

16.207.3.6 pager() [2/2]

```
void L4::Thread::Attr::pager (
    Cap< void > const & pager) [inline], [noexcept]
```

Set the pager capability selector.

Parameters

<i>pager</i>	The capability selector that shall be used for page-fault messages. This capability selector must be valid within the task the thread is bound to.
--------------	--

Definition at line 161 of file [thread](#).

References [pager\(\)](#).

Referenced by [pager\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

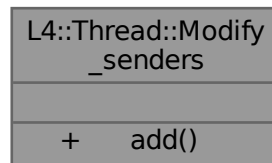
- [l4/sys/thread](#)

16.208 L4::Thread::Modify_senders Class Reference

[Class](#) wrapping a list of rules which modify the sender label of IPC messages inbound to this thread.

```
#include <thread>
```

Collaboration diagram for L4::Thread::Modify_senders:



Public Member Functions

- `int add (l4_umword_t match_mask, l4_umword_t match, l4_umword_t del_bits, l4_umword_t add_bits) noexcept`

Add a rule.

16.208.1 Detailed Description

[Class](#) wrapping a list of rules which modify the sender label of IPC messages inbound to this thread.

Use the [add\(\)](#) function to add modification rules, and use [modify_senders\(\)](#) to commit. Do not use the UTCB in between as it is used by [add\(\)](#) and [modify_senders\(\)](#).

This mechanism shall be used to change the source object labels of every pending IPC of an IPC gate or an IRQ if the labels in such pending IPC become invalid for the receiving thread, potentially because:

- an IPC gate / IRQ was unbound from a thread, or
- an IPC gate / IRQ was removed, or
- the label of an IPC gate / IRQ bound to a thread was changed.

It is not required to perform the modify_sender mechanism after an IPC gate or an IRQ was bound to a thread for the first time.

Definition at line [448](#) of file [thread](#).

16.208.2 Member Function Documentation

16.208.2.1 add()

```

int L4::Thread::Modify_senders::add (
    l4_umword_t match_mask,
    l4_umword_t match,
    l4_umword_t del_bits,
    l4_umword_t add_bits) [inline], [noexcept]
  
```

Add a rule.

Parameters

<i>match_mask</i>	Bitmask of bits to match the label.
<i>match</i>	Bitmask that must be equal to the label after applying match_mask.
<i>del_bits</i>	Bits to be deleted from the label.
<i>add_bits</i>	Bits to be added to the label.

Returns

0 on success, <0 on error

In pseudo code: if ((sender_label & match_mask) == match) { sender_label = (sender_label & ~del_bits) | add_bits; }

Only the first match is applied.

See also

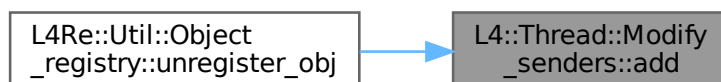
[l4_thread_modify_sender_add\(\)](#)

Definition at line 481 of file [thread](#).

References [L4_ENOMEM](#), and [l4_msg_regs_t::mr](#).

Referenced by [L4Re::Util::Object_registry::unregister_obj\(\)](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

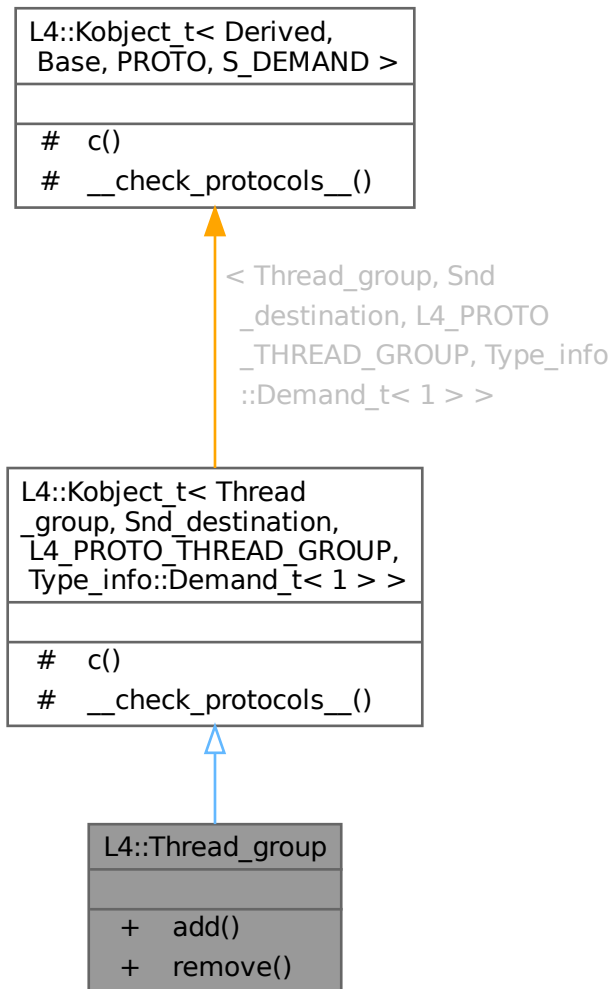
- [l4/sys/thread](#)

16.209 L4::Thread_group Class Reference

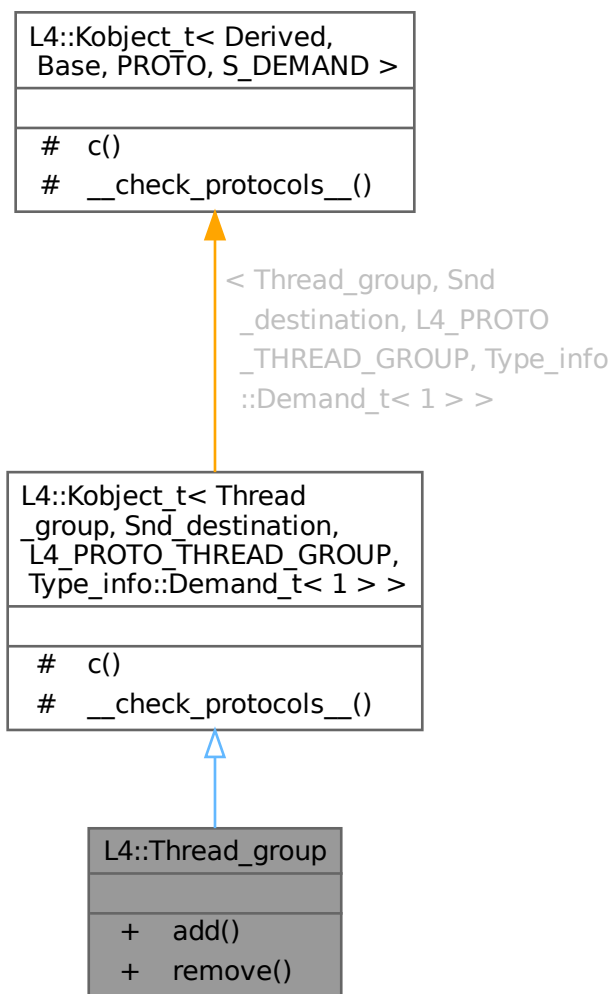
C++ [L4](#) kernel thread group interface, see [Thread groups](#) for the C interface.

```
#include <thread_group>
```

Inheritance diagram for L4::Thread_group:



Collaboration diagram for L4::Thread_group:



Public Member Functions

- `l4_msgtag_t add (Cap< Thread > thread, l4_utcb_t *utcb=l4_utcb()) noexcept`
Add thread to a thread group.
- `l4_msgtag_t remove (Cap< Thread > thread, l4_utcb_t *utcb=l4_utcb()) noexcept`
Remove thread from a thread group.

Additional Inherited Members

Protected Types inherited from

L4::Kobject_t< Thread_group, Snd_destination, L4_PROTO_THREAD_GROUP, Type_info::Demand_t<

- typedef `Thread_group` **Class**

The target interface type (inheriting from [Kobject_t](#)).

- typedef Typeid::Iface< PROTO, [Thread_group](#) > __Iface

The interface description for the derived class.

- typedef Typeid::Merge_list< Typeid::Iface_list< __Iface >, typename Snd_destination::Iface_list > __Iface_list

The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from

[L4::Kobject_t< Thread_group, Snd_destination, L4_PROTO_THREAD_GROUP, Type_info::Demand_t<](#)

- [L4::Cap< Class > c \(\)](#) const noexcept

Get the capability to ourselves.

Static Protected Member Functions inherited from

[L4::Kobject_t< Thread_group, Snd_destination, L4_PROTO_THREAD_GROUP, Type_info::Demand_t<](#)

- static void [__check_protocols__ \(\)](#) noexcept

Helper to check for protocol conflicts.

16.209.1 Detailed Description

C++ [L4](#) kernel thread group interface, see [Thread groups](#) for the C interface.

An [L4](#) thread group is a collection of threads used as indirection for IPC gate and IRQ objects such that these objects can have multiple receivers, from which the kernel selects one according to a policy.

The primary use case for thread groups are multi-threaded servers and CPU core local IRQ / IPC delivery.

A thread can be bound to at most one thread group. Before binding a thread to a thread group, the thread must be bound to a task. All threads bound to the same thread group must belong to the same task.

Definition at line [34](#) of file [thread_group](#).

16.209.2 Member Function Documentation

16.209.2.1 add()

```
l4\_msgtag\_t L4::Thread_group::add (
    Cap< Thread > thread,
    l4\_utcb\_t * utcb = l4\_utcb\(\)) [inline], [noexcept]
```

Add thread to a thread group.

Parameters

<i>thread</i>	Thread to add to the thread group.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

Syscall return tag containing one of the following return codes.

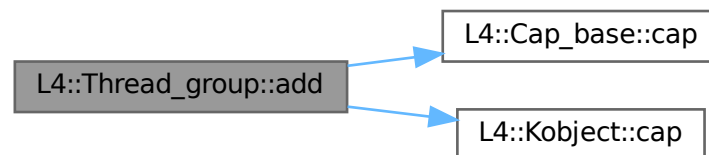
Return values

<i>L4_EOK</i>	No error occurred.
<i>-L4_EINVAL</i>	<code>thread</code> is not a thread object.
<i>-L4_EEXIST</i>	<code>thread</code> already bound to this thread group.
<i>-L4_EBUSY</i>	<code>thread</code> already bound to a different thread group.
<i>-L4_ENOENT</i>	Thread group doesn't exist.

Definition at line 53 of file [thread_group](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



16.209.2.2 remove()

```

l4_msgtag_t L4::Thread_group::remove (
    Cap< Thread > thread,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
  
```

Remove thread from a thread group.

Parameters

<i>thread</i>	Thread to remove from the thread group.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Returns

Syscall return tag containing one of the following return codes.

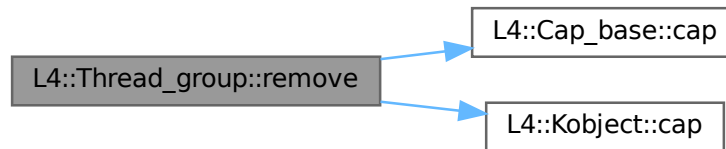
Return values

<i>L4_EOK</i>	No error occurred.
<i>-L4_EINVAL</i>	<code>thread</code> is not a thread object.

Definition at line 67 of file [thread_group](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

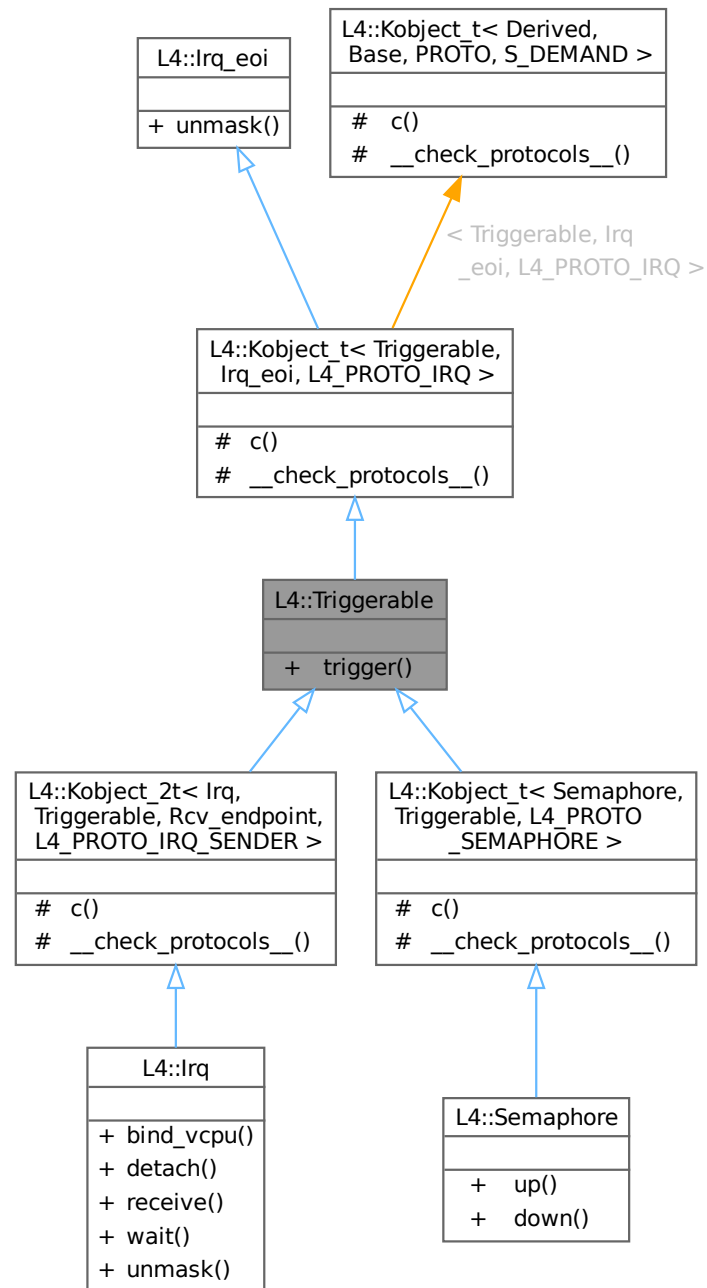
- [l4/sys/thread_group](#)

16.210 L4::Triggerable Struct Reference

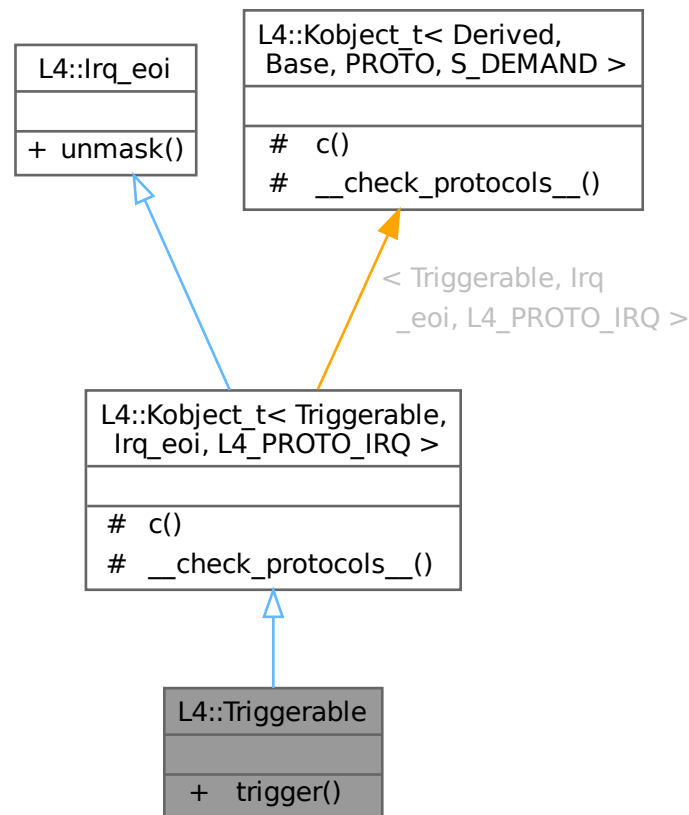
Interface that allows an object to be triggered by some source.

```
#include <irq>
```

Inheritance diagram for L4::Triggerable:



Collaboration diagram for L4::Triggerable:



Public Member Functions

- [l4_msgtag_t trigger](#) ([l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept
Trigger the object.

Public Member Functions inherited from [L4::Irq_eoi](#)

- [l4_msgtag_t unmask](#) (unsigned irqnum, [l4_umword_t](#) *label=0, [l4_timeout_t](#) to=[L4_IPC_NEVER](#), [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept
Unmask the given interrupt line.

Additional Inherited Members

Protected Types inherited from [L4::Kobject_t< Triggerable, Irq_eoi, L4_PROTO_IRQ >](#)

- typedef [Triggerable](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< PROTO, [Triggerable](#) > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< [__Iface](#) >, typename Irq_eoi::__Iface_list > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from [L4::Kobject_t< Triggerable, Irq_eoi, L4_PROTO_IRQ >](#)

- [L4::Cap< Class > c \(\)](#) const noexcept
Get the capability to ourselves.

Static Protected Member Functions inherited from [L4::Kobject_t< Triggerable, Irq_eoi, L4_PROTO_IRQ >](#)

- static void [__check_protocols__ \(\)](#) noexcept
Helper to check for protocol conflicts.

16.210.1 Detailed Description

Interface that allows an object to be triggered by some source.

The interface specifies no semantics for the trigger operation, this is defined by derived objects.

This interface is usually used in conjunction with [L4::lcu](#).

Definition at line 79 of file [irq](#).

16.210.2 Member Function Documentation

16.210.2.1 trigger()

```
l4_msgtag_t L4::Triggerable::trigger (
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
```

Trigger the object.

Parameters

<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .
-------------	--

Returns

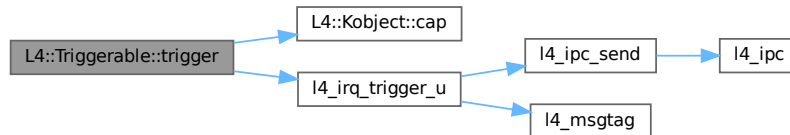
Syscall return tag for a send-only operation, this means there is no return value except [L4_MSGTAG_ERROR](#) indicating success or failure of the send operation. Use [l4_ipc_error\(\)](#) to check for errors and **do not** use [l4_error\(\)](#).

Definition at line 91 of file [irq](#).

References [L4::Kobject::cap\(\)](#), and [l4_irq_trigger_u\(\)](#).

Referenced by [L4::Semaphore::up\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this struct was generated from the following file:

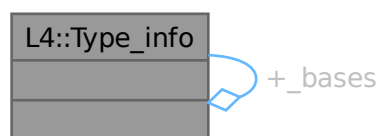
- [l4/sys/irq](#)

16.211 L4::Type_info Struct Reference

Dynamic Type Information for [L4Re](#) Interfaces.

```
#include <l4/sys/capability>
```

Collaboration diagram for `L4::Type_info`:



Data Structures

- class [Demand](#)
Data type for expressing the needed receive buffers at the server-side of an interface.
- struct [Demand_t](#)
Template type statically describing demand of receive buffers.
- struct [Demand_union_t](#)
Template type statically describing the combination of two [Demand](#) object.

16.211.1 Detailed Description

Dynamic Type Information for [L4Re](#) Interfaces.

This class represents the runtime-dynamic type information for [L4Re](#) interfaces, and is not intended to be used directly by applications.

Note

The interface of is subject to changes.

The main use for this info is to be used by the implementation of the [L4::cap_dynamic_cast\(\)](#) function.

Definition at line 499 of file [__typeinfo.h](#).

The documentation for this struct was generated from the following file:

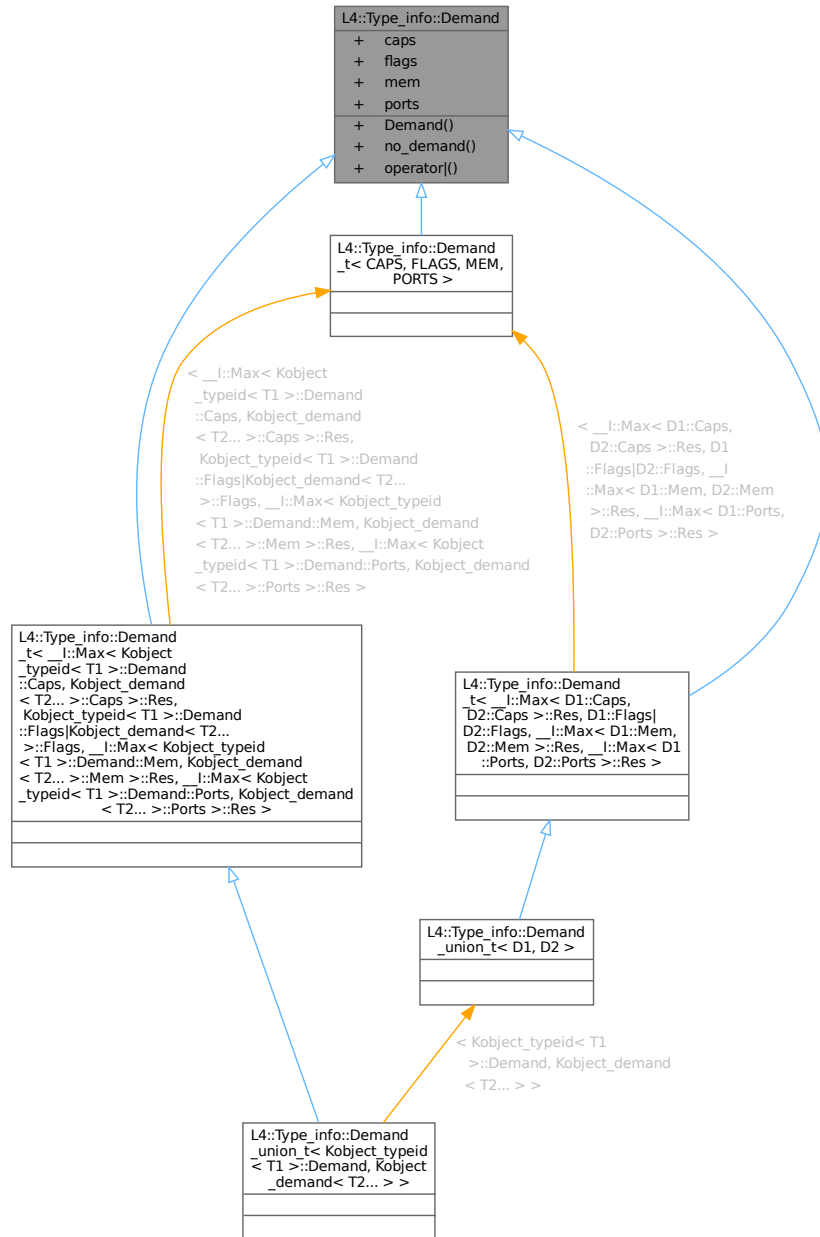
- [l4/sys/__typeinfo.h](#)

16.212 L4::Type_info::Demand Class Reference

Data type for expressing the needed receive buffers at the server-side of an interface.

```
#include <l4/sys/capability>
```

Inheritance diagram for L4::Type_info::Demand:



Collaboration diagram for L4::Type_info::Demand:

L4::Type_info::Demand	
+	caps
+	flags
+	mem
+	ports
+	Demand()
+	no_demand()
+	operator ()

Public Member Functions

- [Demand](#) (unsigned char [caps](#)=0, unsigned char [flags](#)=0, unsigned char [mem](#)=0, unsigned char [ports](#)=0) noexcept
Make [Demand](#) object.
- bool [no_demand](#) () const noexcept
- [Demand operator|](#) ([Demand](#) const &rhs) const noexcept
get the combined demand of this and rhs

Data Fields

- unsigned char **caps**
number of capability receive buffers.
- unsigned char **flags**
flags, such as the need for timeouts (TBD).
- unsigned char **mem**
number of memory receive buffers.
- unsigned char **ports**
number of IO-port receive buffers.

16.212.1 Detailed Description

Data type for expressing the needed receive buffers at the server-side of an interface.

Definition at line 506 of file [__typeinfo.h](#).

16.212.2 Constructor & Destructor Documentation

16.212.2.1 Demand()

```
L4::Type_info::Demand::Demand (  
    unsigned char caps = 0,  
    unsigned char flags = 0,  
    unsigned char mem = 0,  
    unsigned char ports = 0) [inline], [explicit], [noexcept]
```

Make [Demand](#) object.

Parameters

<i>caps</i>	number of capability receive buffers
<i>flags</i>	flags, such as the need for timeouts (TBD).
<i>mem</i>	number of memory receive windows.
<i>ports</i>	number of IO-port receive windows.

Definition at line [527](#) of file [__typeinfo.h](#).

References [caps](#), [flags](#), [mem](#), and [ports](#).

Referenced by [operator|\(\)](#).

Here is the caller graph for this function:



16.212.3 Member Function Documentation

16.212.3.1 no_demand()

```
bool L4::Type_info::Demand::no_demand () const [inline], [noexcept]
```

Returns

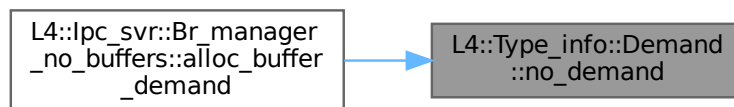
true if there is no demand at all

Definition at line 532 of file [__typeinfo.h](#).

References [caps](#), [flags](#), [mem](#), and [ports](#).

Referenced by [L4::lpc_svr::Br_manager_no_buffers::alloc_buffer_demand\(\)](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

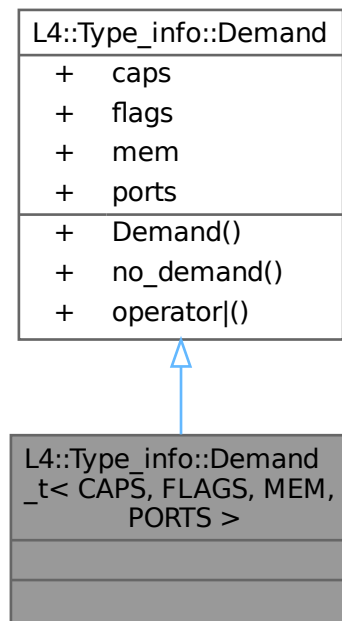
- [l4/sys/__typeinfo.h](#)

16.213 L4::Type_info::Demand_t< CAPS, FLAGS, MEM, PORTS > Struct Template Reference

Template type statically describing demand of receive buffers.

```
#include <l4/sys/capability>
```


Collaboration diagram for L4::Type_info::Demand_t< CAPS, FLAGS, MEM, PORTS >:



Public Types

- enum { `Caps` = CAPS , `Flags` = FLAGS , `Mem` = MEM , `Ports` = PORTS }

Additional Inherited Members

Public Member Functions inherited from L4::Type_info::Demand

- `Demand` (unsigned char `caps`=0, unsigned char `flags`=0, unsigned char `mem`=0, unsigned char `ports`=0) noexcept
Make `Demand` object.
- bool `no_demand` () const noexcept
- `Demand operator|` (`Demand` const &rhs) const noexcept
get the combined demand of this and rhs

Data Fields inherited from L4::Type_info::Demand

- unsigned char `caps`
number of capability receive buffers.
- unsigned char `flags`
flags, such as the need for timeouts (TBD).
- unsigned char `mem`
number of memory receive buffers.
- unsigned char `ports`
number of IO-port receive buffers.

16.213.1 Detailed Description

```
template<unsigned char CAPS = 0, unsigned char FLAGS = 0, unsigned char MEM = 0, unsigned char
PORTS = 0>
struct L4::Type_info::Demand_t< CAPS, FLAGS, MEM, PORTS >
```

Template type statically describing demand of receive buffers.

Template Parameters

<i>CAPS</i>	number of capability receive buffers needed.
<i>FLAGS</i>	flags, such as the need for timeouts (TBD).
<i>MEM</i>	number of memory receive windows needed.
<i>PORTS</i>	number of IO-port receive windwows needed.

Definition at line 553 of file [__typeinfo.h](#).

16.213.2 Member Enumeration Documentation

16.213.2.1 anonymous enum

```
template<unsigned char CAPS = 0, unsigned char FLAGS = 0, unsigned char MEM = 0, unsigned char
PORTS = 0>
anonymous enum
```

Enumerator

Caps	number of capability receive buffers.
Flags	flags, such as the need for timeouts.
Mem	number of memory receive windows.
Ports	number of IO-port receive windows.

Definition at line 555 of file [__typeinfo.h](#).

The documentation for this struct was generated from the following file:

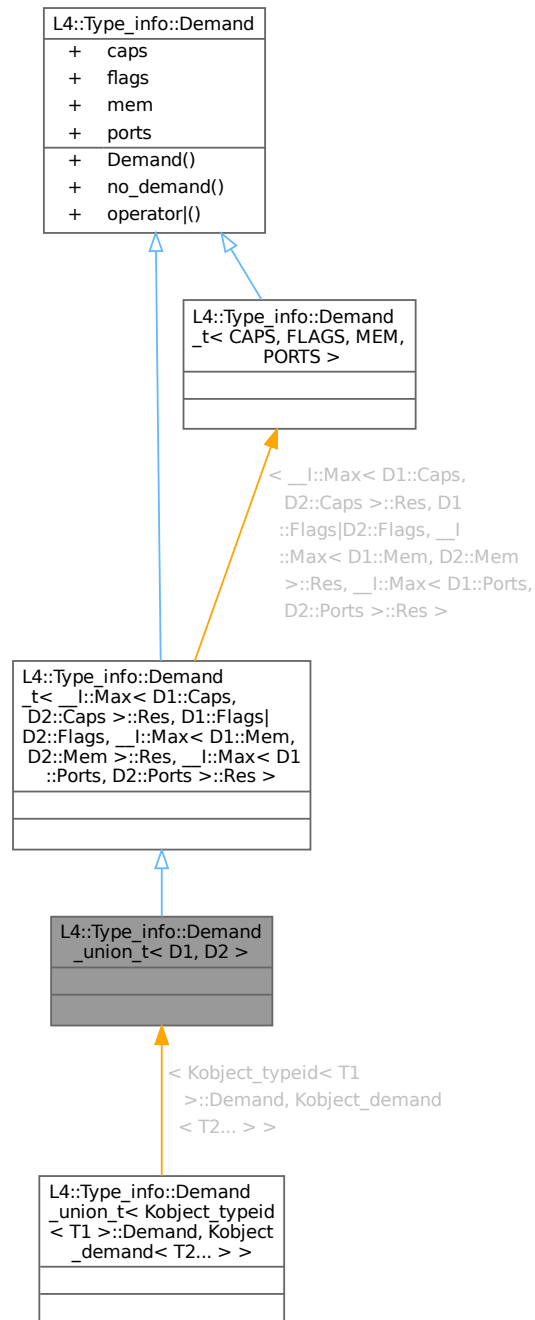
- [l4/sys/__typeinfo.h](#)

16.214 L4::Type_info::Demand_union_t< D1, D2 > Struct Template Reference

Template type statically describing the combination of two [Demand](#) object.

```
#include <l4/sys/capability>
```

Inheritance diagram for L4::Type_info::Demand_union_t< D1, D2 >:



Additional Inherited Members

Public Types inherited from

[L4::Type_info::Demand_t< __l::Max< D1::Caps, D2::Caps >::Res, D1::Flags|D2::Flags, __l::Max< D1::](#)

Public Member Functions inherited from [L4::Type_info::Demand](#)

- [Demand](#) (unsigned char [caps](#)=0, unsigned char [flags](#)=0, unsigned char [mem](#)=0, unsigned char [ports](#)=0) noexcept
Make [Demand](#) object.
- bool [no_demand](#) () const noexcept
- [Demand](#) operator| ([Demand](#) const &rhs) const noexcept
get the combined demand of this and rhs

Data Fields inherited from [L4::Type_info::Demand](#)

- unsigned char **caps**
number of capability receive buffers.
- unsigned char **flags**
flags, such as the need for timeouts (TBD).
- unsigned char **mem**
number of memory receive buffers.
- unsigned char **ports**
number of IO-port receive buffers.

16.214.1 Detailed Description

```
template<typename D1, typename D2>
struct L4::Type_info::Demand_union_t< D1, D2 >
```

Template type statically describing the combination of two [Demand](#) object.

Template Parameters

<i>D1</i>	first demand object.
<i>D2</i>	second demand object.

Definition at line [573](#) of file [__typeinfo.h](#).

The documentation for this struct was generated from the following file:

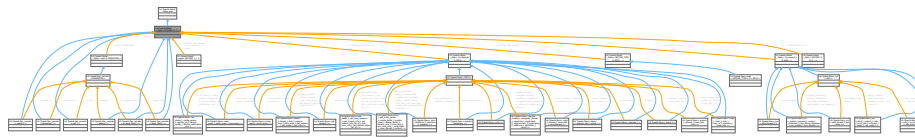
- [l4/sys/__typeinfo.h](#)

16.215 L4::Typeid::Detail::_Rpc< OPCODE, O, X > Struct Template Reference

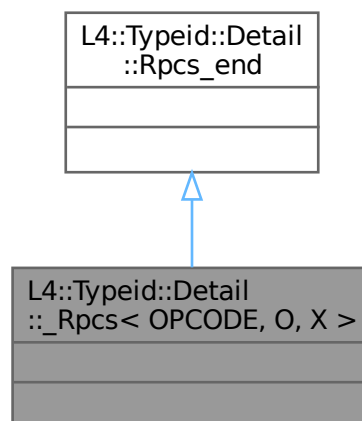
Empty list of RPCs.

```
#include <__typeinfo.h>
```

Inheritance diagram for L4::Typeid::Detail::_Rpc< OPCODE, O, X >:



Collaboration diagram for L4::Typeid::Detail::_Rpc< OPCODE, O, X >:



16.215.1 Detailed Description

```
template<typename OPCODE, unsigned O, typename ... X>
struct L4::Typeid::Detail::_Rpc< OPCODE, O, X >
```

Empty list of RPCs.

Definition at line 365 of file [__typeinfo.h](#).

The documentation for this struct was generated from the following file:

- [l4/sys/__typeinfo.h](#)

16.216 L4::Typeid::Detail::_Rpc< OPCODE, O, Default_op< R > >::Rpc< Y > Struct Template Reference

Find the given RPC in the list.

```
#include <__typeinfo.h>
```

Collaboration diagram for L4::Typeid::Detail::_Rpc< OPCODE, O, Default_op< R > >::Rpc< Y >:

L4::Typeid::Detail :: _Rpc< OPCODE, O, Default_op< R > >::Rpc< Y >

16.216.1 Detailed Description

```
template<typename OPCODE, unsigned O, typename R>
template<typename Y>
struct L4::Typeid::Detail::_Rpc< OPCODE, O, Default_op< R > >::Rpc< Y >
```

Find the given RPC in the list.

Definition at line 399 of file [__typeinfo.h](#).

The documentation for this struct was generated from the following file:

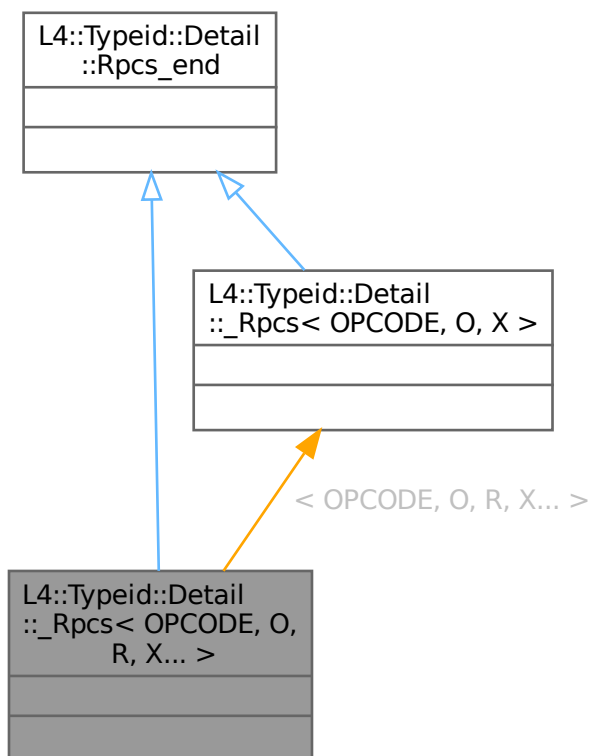
- l4/sys/[__typeinfo.h](#)

16.217 L4::Typeid::Detail::_Rpc< OPCODE, O, R, X... > Struct Template Reference

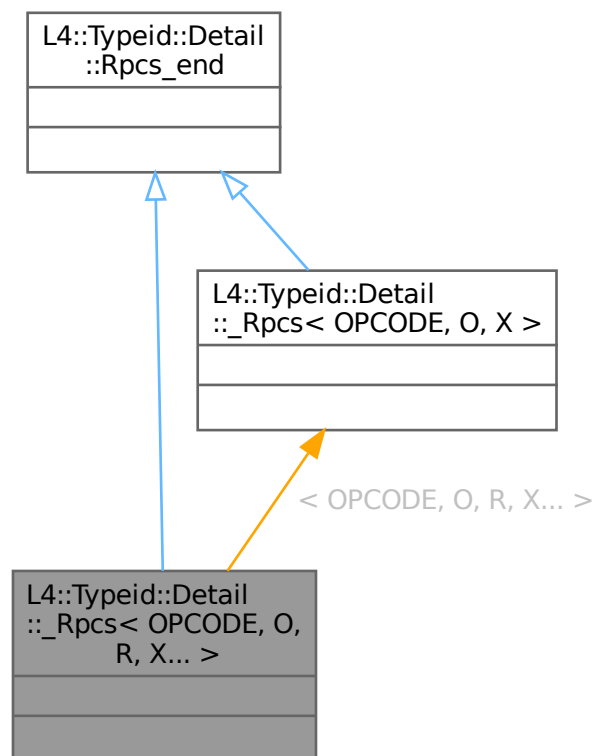
Non-empty list of RPCs.

```
#include <__typeinfo.h>
```

Inheritance diagram for L4::Typeid::Detail::_Rpccs< OPCODE, O, R, X... >:



Collaboration diagram for L4::Typeid::Detail::_Rpccs< OPCODE, O, R, X... >:



Data Structures

- struct [Rpc](#)

Find the given RPC in the list.

Public Types

- enum
The opcode value to use for this RPC, may be bogus if the [opcode_type](#) is void.
- typedef [_Rpccs](#) **type**
The list element itself.
- typedef OPCODE **opcode_type**
The data type for the opcode.
- typedef R **rpc**
The RPC type L4::lpc::Msg::Rpc_call or L4::lpc::Msg::Rpc_inline_call.
- typedef [_Rpccs](#)< OPCODE, _Get_opcode< R, O >::value+1, X... > **::type next**
The next RPC in the list or [Rpccs_end](#) if this is the last.

16.217.1 Detailed Description

```
template<typename OPCODE, unsigned O, typename R, typename ... X>
struct L4::Typeid::Detail::_Rpc< OPCODE, O, R, X... >
```

Non-empty list of RPCs.

Definition at line 369 of file [__typeinfo.h](#).

The documentation for this struct was generated from the following file:

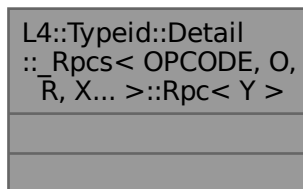
- [l4/sys/__typeinfo.h](#)

16.218 L4::Typeid::Detail::_Rpc< OPCODE, O, R, X... >::Rpc< Y > Struct Template Reference

Find the given RPC in the list.

```
#include <__typeinfo.h>
```

Collaboration diagram for L4::Typeid::Detail::_Rpc< OPCODE, O, R, X... >::Rpc< Y >:



16.218.1 Detailed Description

```
template<typename OPCODE, unsigned O, typename R, typename ... X>
template<typename Y>
struct L4::Typeid::Detail::_Rpc< OPCODE, O, R, X... >::Rpc< Y >
```

Find the given RPC in the list.

Definition at line 382 of file [__typeinfo.h](#).

The documentation for this struct was generated from the following file:

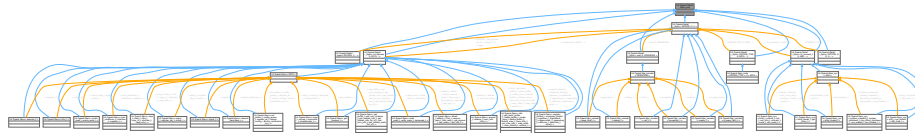
- [l4/sys/__typeinfo.h](#)

16.219 L4::Typeid::Detail::Rpc_end Struct Reference

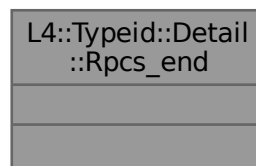
Internal end-of-list marker.

```
#include <__typeinfo.h>
```

Inheritance diagram for L4::Typeid::Detail::Rpc_end:



Collaboration diagram for L4::Typeid::Detail::Rpc_end:



16.219.1 Detailed Description

Internal end-of-list marker.

Definition at line 317 of file [__typeinfo.h](#).

The documentation for this struct was generated from the following file:

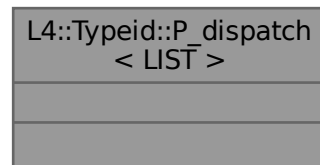
- [l4/sys/__typeinfo.h](#)

16.220 L4::Typeid::P_dispatch< LIST > Struct Template Reference

Use for protocol based dispatch stage.

```
#include <__typeinfo.h>
```

Collaboration diagram for L4::Typeid::P_dispatch< LIST >:



16.220.1 Detailed Description

```
template<typename LIST>
struct L4::Typeid::P_dispatch< LIST >
```

Use for protocol based dispatch stage.

Definition at line 308 of file [__typeinfo.h](#).

The documentation for this struct was generated from the following file:

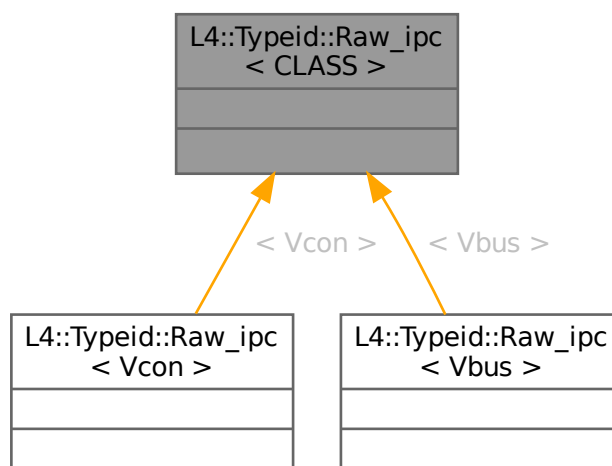
- [l4/sys/__typeinfo.h](#)

16.221 L4::Typeid::Raw_ipc< CLASS > Struct Template Reference

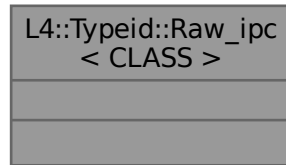
RPCs list for passing raw incoming IPC to the server object.

```
#include <l4/sys/capability>
```

Inheritance diagram for L4::Typeid::Raw_ipc< CLASS >:



Collaboration diagram for L4::Typeid::Raw_ipc< CLASS >:



16.221.1 Detailed Description

```
template<typename CLASS>
struct L4::Typeid::Raw_ipc< CLASS >
```

RPCs list for passing raw incoming IPC to the server object.

Template Parameters

<i>CLASS</i>	The type of the interface (e.g., L4::lcu)
--------------	--

This template allows to have fully handcrafted IPC protocols.

Definition at line [412](#) of file [__typeinfo.h](#).

The documentation for this struct was generated from the following file:

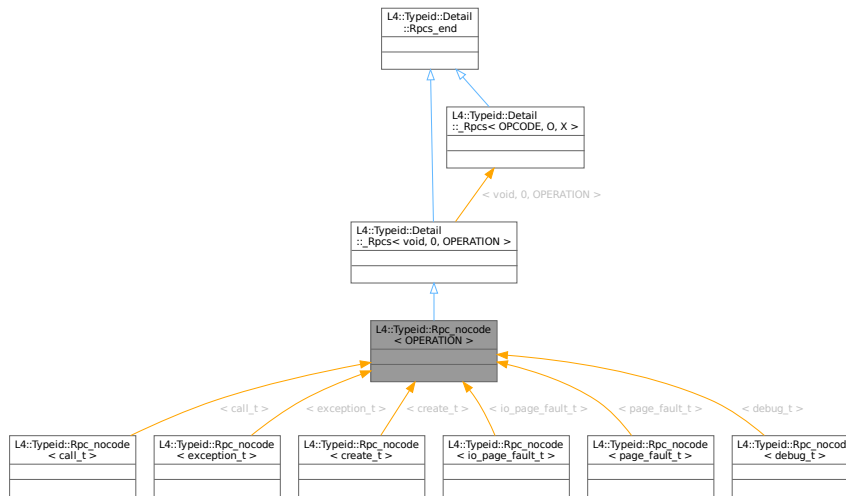
- [l4/sys/__typeinfo.h](#)

16.222 L4::Typeid::Rpc_nocode< OPERATION > Struct Template Reference

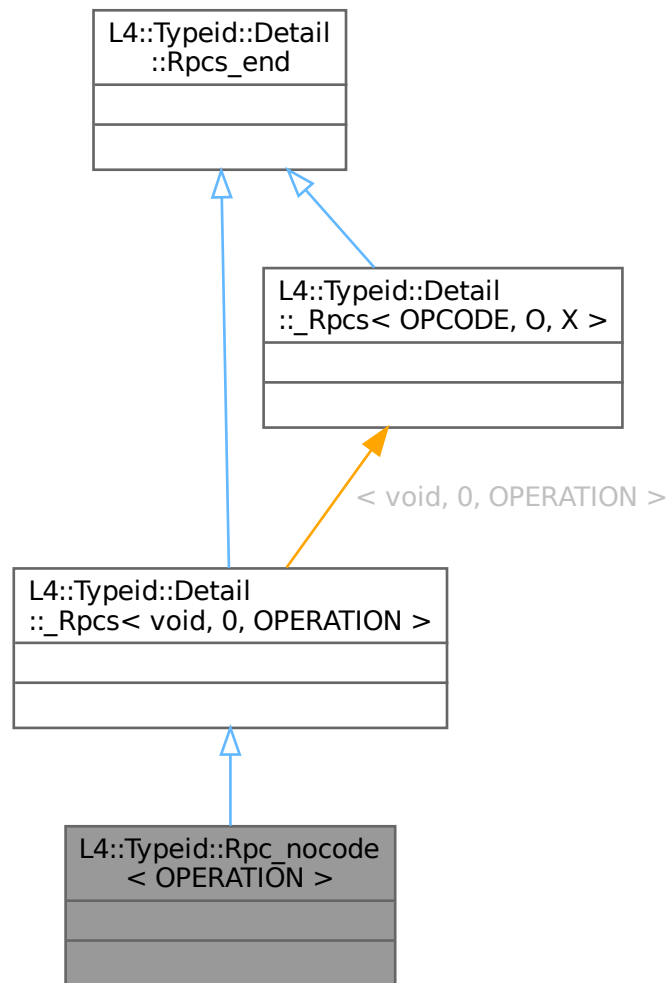
List of RPCs of an interface using a single operation without an opcode.

```
#include <l4/sys/capability>
```

Inheritance diagram for L4::Typeid::Rpc_nocode< OPERATION >:



Collaboration diagram for L4::Typeid::Rpc_nocode< OPERATION >:



16.222.1 Detailed Description

```
template<typename OPERATION>
struct L4::Typeid::Rpc_nocode< OPERATION >
```

List of RPCs of an interface using a single operation without an opcode.

Template Parameters

<i>OPERATION</i>	The RPC operation as defined by L4_RPC etc.
------------------	---

Definition at line 454 of file [__typeinfo.h](#).

The documentation for this struct was generated from the following file:

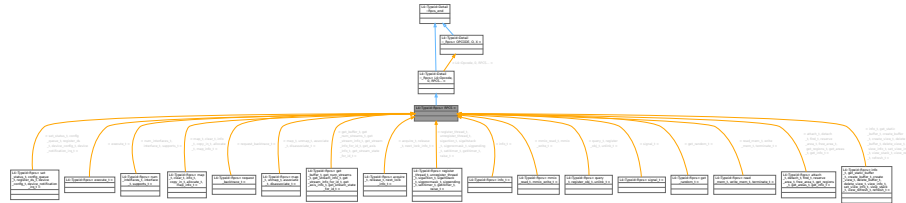
- [l4/sys/__typeinfo.h](#)

16.223 L4::Typeid::Rpc< RPCS > Struct Template Reference

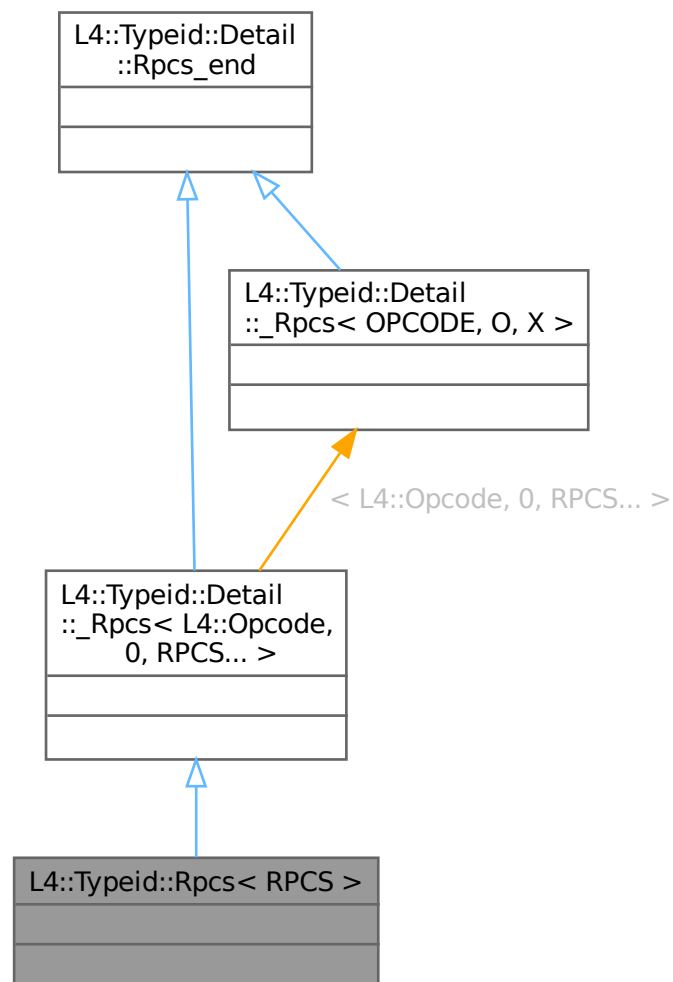
Standard list of RPCs of an interface.

```
#include <l4/sys/capability>
```

Inheritance diagram for L4::Typeid::Rpc< RPCS >:



Collaboration diagram for L4::Typeid::Rpc< RPCS >:



16.223.1 Detailed Description

```
template<typename ... RPCS>
struct L4::Typeid::Rpc< RPCS >
```

Standard list of RPCs of an interface.

Template Parameters

<i>RPCS</i>	list of RPC types as defined by L4_RPC etc.
-------------	---

This is the default list for RPC functions of an interface, it uses [L4::Opcode](#) as opcode type and uses opcodes starting from 0.

Examples

[examples/clntsrv/src/shared.h](#).

Definition at line [428](#) of file [__typeinfo.h](#).

The documentation for this struct was generated from the following file:

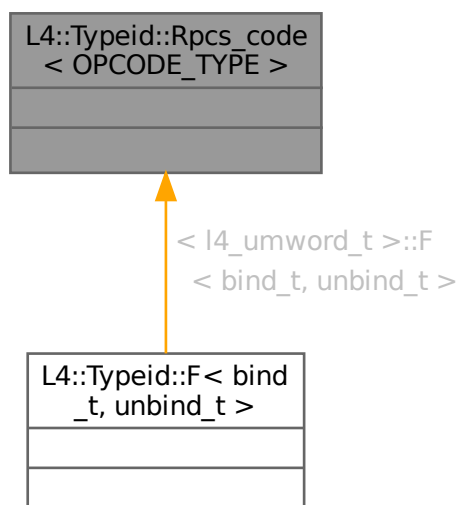
- [l4/sys/__typeinfo.h](#)

16.224 L4::Typeid::Rpc_code< OPCODE_TYPE > Struct Template Reference

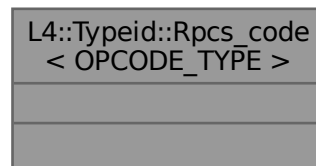
List of RPCs of an interface using a special opcode type.

```
#include <l4/sys/capability>
```

Inheritance diagram for L4::Typeid::Rpc_code< OPCODE_TYPE >:



Collaboration diagram for L4::Typeid::Rpc_code< OPCODE_TYPE >:



Data Structures

- struct [F](#)

16.224.1 Detailed Description

```
template<typename OPCODE_TYPE>
struct L4::Typeid::Rpc_code< OPCODE_TYPE >
```

List of RPCs of an interface using a special opcode type.

Template Parameters

<i>OPCODE_TYPE</i>	The data type of the opcode.
--------------------	------------------------------

List for RPC functions of an interface, using OPCODE_TYPE as data type for the opcode, opcodes starting from 0.

Definition at line [439](#) of file [__typeinfo.h](#).

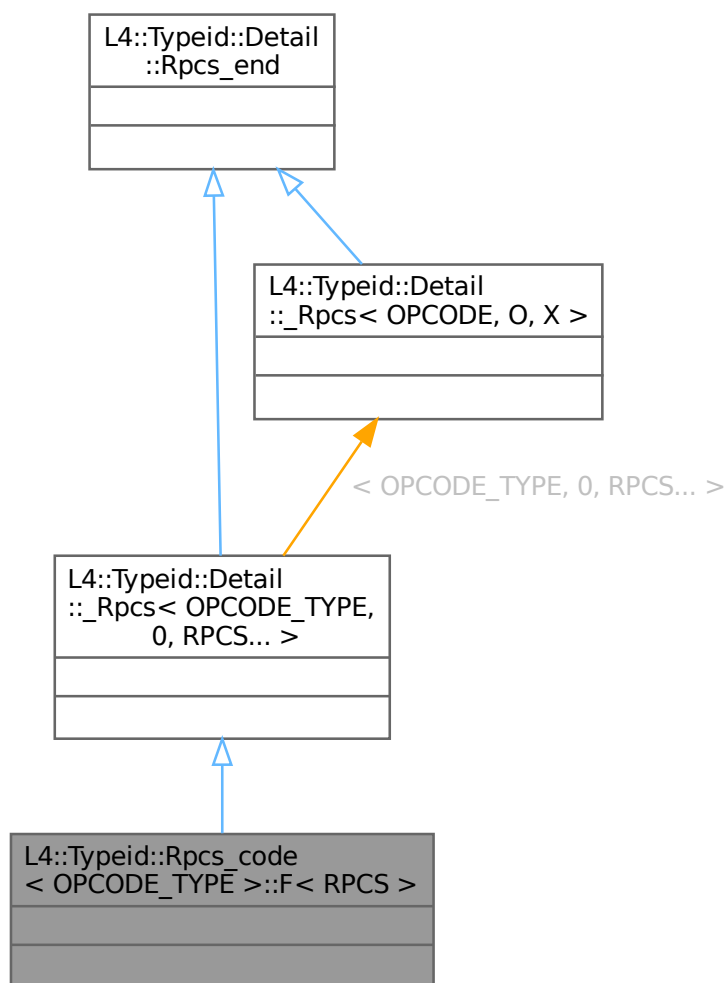
The documentation for this struct was generated from the following file:

- [l4/sys/__typeinfo.h](#)

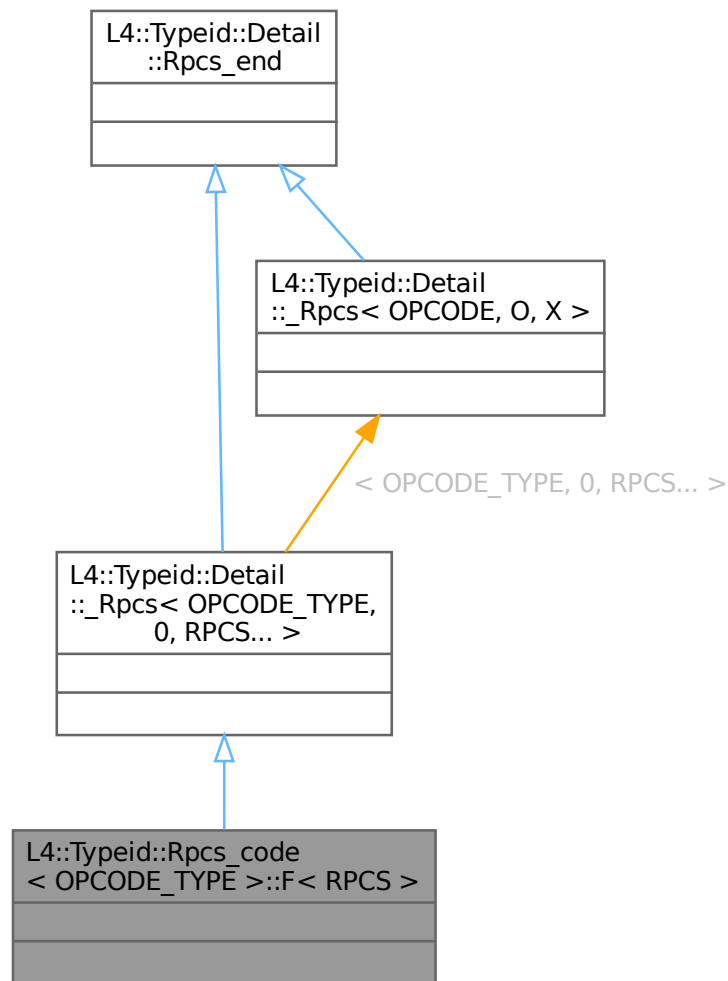
16.225 L4::Typeid::Rpc_code< OPCODE_TYPE >::F< RPCS > Struct Template Reference

```
#include <__typeinfo.h>
```

Inheritance diagram for L4::Typeid::Rpc_code< OPCODE_TYPE >::F< RPCS >:



Collaboration diagram for L4::Typeid::Rpc_code< OPCODE_TYPE >::F< RPCS >:



16.225.1 Detailed Description

```

template<typename OPCODE_TYPE>
template<typename ... RPCS>
struct L4::Typeid::Rpc_code< OPCODE_TYPE >::F< RPCS >

```

Template Parameters

<i>RPCS</i>	list of RPC types as defined by L4_RPC etc.
-------------	---

Definition at line 445 of file [__typeinfo.h](#).

The documentation for this struct was generated from the following file:

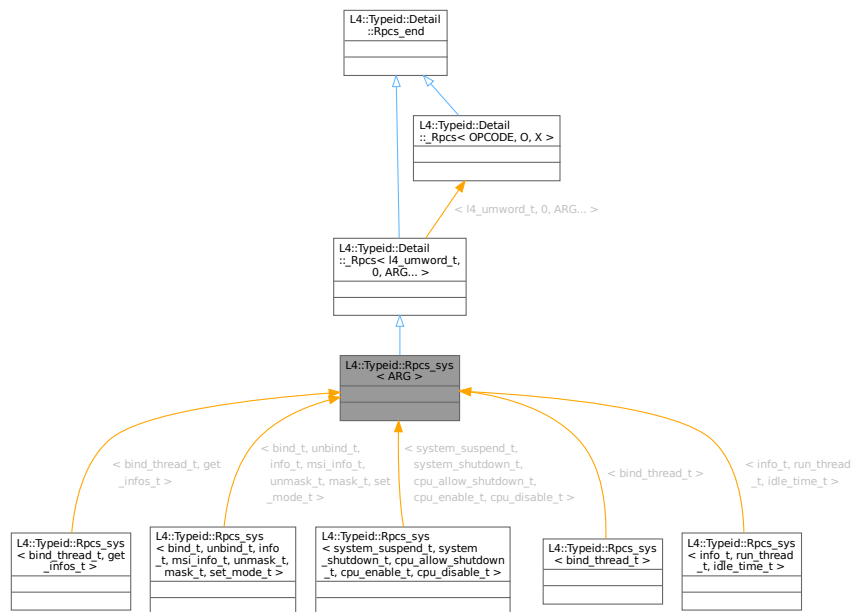
- [l4/sys/__typeinfo.h](#)

16.226 L4::Typeid::Rpcsys< ARG > Struct Template Reference

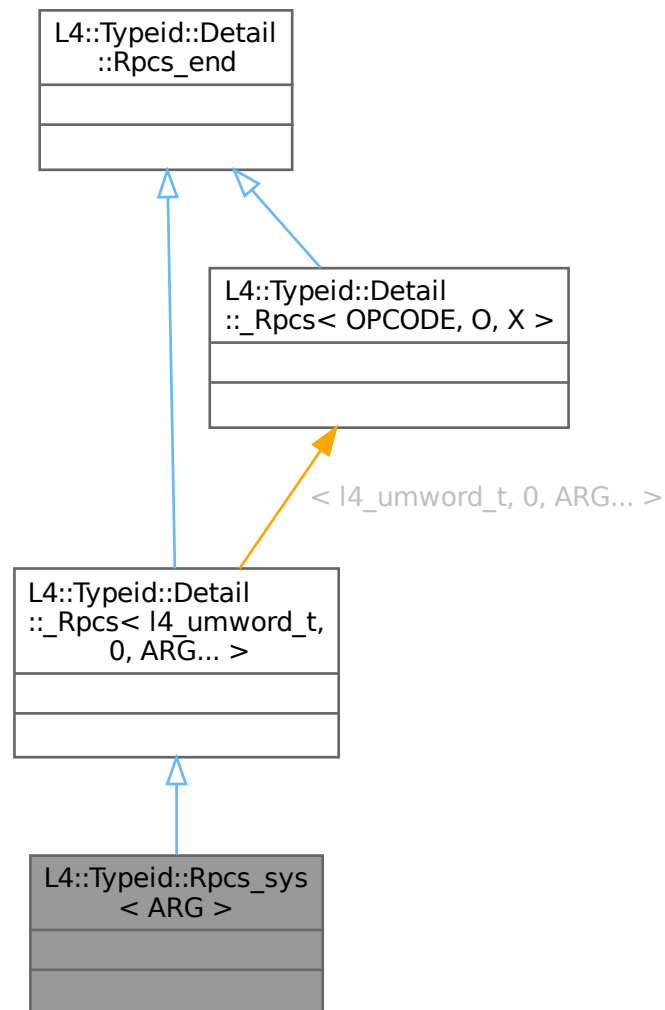
List of RPCs typically used for kernel interfaces.

```
#include <l4/sys/capability>
```

Inheritance diagram for L4::Typeid::Rpcsys< ARG >:



Collaboration diagram for L4::Typeid::Rpcsys< ARG >:



16.226.1 Detailed Description

```
template<typename ... ARG>
struct L4::Typeid::Rpcsys< ARG >
```

List of RPCs typically used for kernel interfaces.

Template Parameters

<i>RPCS</i>	list of RPC types as defined by L4_RPC etc.
-------------	---

This list of RPC functions uses [l4_umword_t](#) as type for the opcode as most kernel protocol do.

Definition at line 465 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

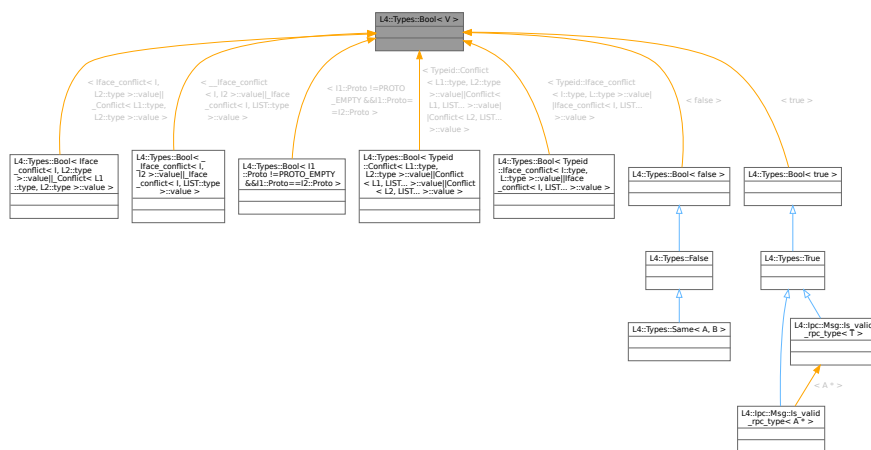
- `l4/sys/__typeinfo.h`

16.227 L4::Types::Bool< V > Struct Template Reference

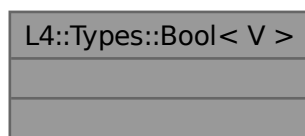
Boolean meta type.

```
#include <types>
```

Inheritance diagram for L4::Types::Bool< V >:



Collaboration diagram for L4::Types::Bool< V >:



Public Types

- `typedef Bool < V > type`
The meta type itself.

16.227.1 Detailed Description

```
template<bool V>  
struct L4::Types::Bool< V >
```

Boolean meta type.

Template Parameters

V	The boolean value
---	-------------------

Definition at line 288 of file [types](#).

The documentation for this struct was generated from the following file:

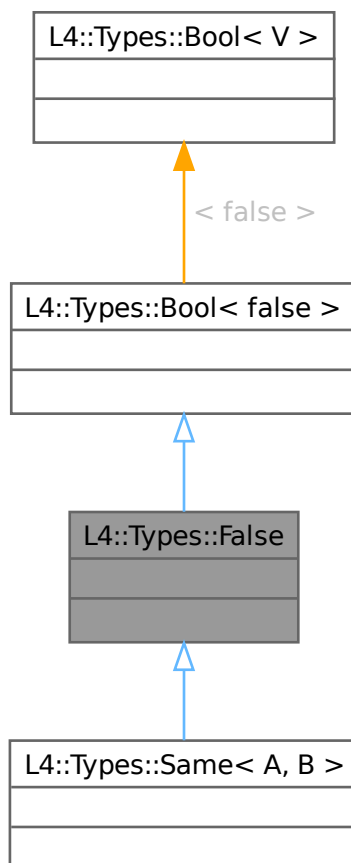
- [l4/sys/cxx/types](#)

16.228 L4::Types::False Struct Reference

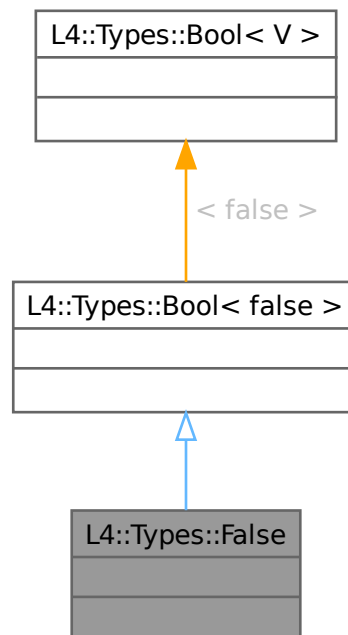
[False](#) meta value.

```
#include <types>
```

Inheritance diagram for L4::Types::False:



Collaboration diagram for L4::Types::False:



Additional Inherited Members

Public Types inherited from `L4::Types::Bool< false >`

- typedef `Bool< V >` **type**
The meta type itself.

16.228.1 Detailed Description

`False` meta value.

Definition at line 296 of file `types`.

The documentation for this struct was generated from the following file:

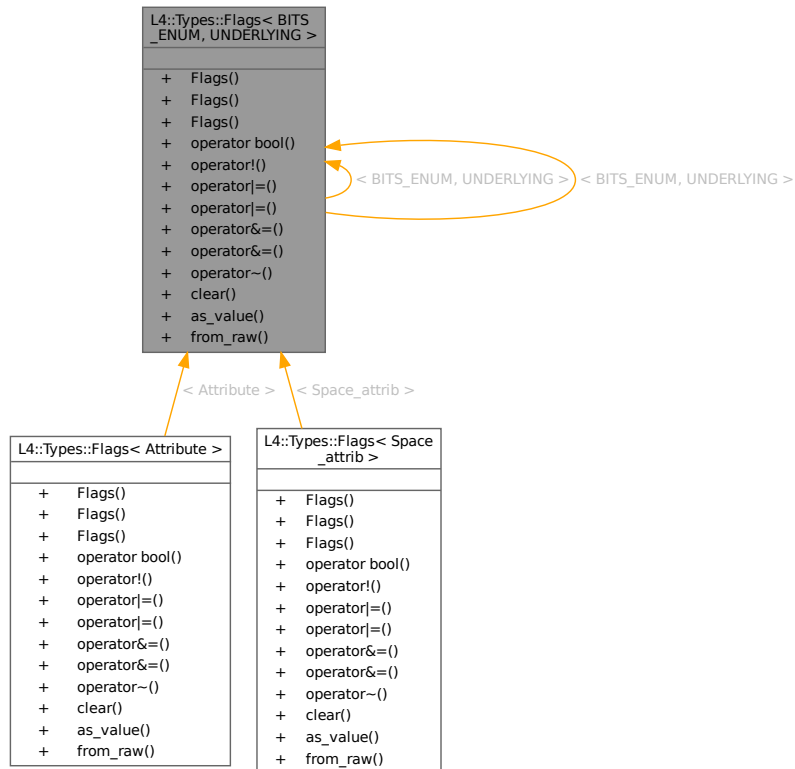
- `l4/sys/cxx/types`

16.229 L4::Types::Flags< BITS_ENUM, UNDERLYING > Class Template Reference

Template for defining typical [Flags](#) bitmaps.

```
#include <types>
```

Inheritance diagram for L4::Types::Flags< BITS_ENUM, UNDERLYING >:



Collaboration diagram for L4::Types::Flags< BITS_ENUM, UNDERLYING >:

L4::Types::Flags< BITS_ENUM, UNDERLYING >
<ul style="list-style-type: none"> + Flags() + Flags() + Flags() + operator bool() + operator!() + operator =() + operator =() + operator&=() + operator&=() + operator~() + clear() + as_value() + from_raw()

Public Types

- enum [None_type](#) { [None](#) }
The none type to get an empty bitmap.
- typedef UNDERLYING **value_type**
type of the underlying value
- typedef BITS_ENUM **bits_enum_type**
enum type defining a name for each bit
- typedef [Flags](#)< BITS_ENUM, UNDERLYING > **type**
the Flags<> type itself

Public Member Functions

- [Flags](#) ([None_type](#))
Make an empty bitmap.
- **Flags** ()
Make default [Flags](#).
- [Flags](#) (BITS_ENUM e)
Make flags from bit name.
- **operator bool** () const
Support for `if (flags)` syntax (test for non-empty flags).
- **bool operator!** () const
Support for `if (!flags)` syntax (test for empty flags).

- **type & operator|** (type rhs)
Support | of two compatible [Flags](#) types.
- **type & operator|** (bits_enum_type rhs)
Support | of [Flags](#) type and bit name.
- **type & operator&=** (type rhs)
Support &= of two compatible [Flags](#) types.
- **type & operator&=** (bits_enum_type rhs)
Support &= of [Flags](#) type and bit name.
- **type operator~** () const
Support ~ for [Flags](#) types.
- **type & clear** (bits_enum_type flag)
Clear the given flag.
- **value_type as_value** () const
Get the underlying value.

Static Public Member Functions

- static **type from_raw** (value_type v)
Make flags from a raw value of [value_type](#).

Friends

- **type operator|** (type lhs, type rhs)
Support | of two compatible [Flags](#) types.
- **type operator|** (type lhs, bits_enum_type rhs)
Support | of [Flags](#) type and bit name.
- **type operator&** (type lhs, type rhs)
Support & of two compatible [Flags](#) types.
- **type operator&** (type lhs, bits_enum_type rhs)
Support & of [Flags](#) type and bit name.

16.229.1 Detailed Description

```
template<typename BITS_ENUM, typename UNDERLYING = unsigned long>
class L4::Types::Flags< BITS_ENUM, UNDERLYING >
```

Template for defining typical [Flags](#) bitmaps.

Template Parameters

<i>BITS_ENUM</i>	enum type that defines a name for each bit in the bitmap. The values of the enum members must be the number of the bit (<i>not</i> a mask).
<i>UNDERLYING</i>	The underlying data type used to represent the bitmap.

The resulting data type provides a type-safe version that allows bitwise `and` and `or` operations with the `BITS_ENUM` members. As well as, test for `0` or `!0`.

Example:

```
enum Test_flag
{
    Do_weak_tests,
    Do_strong_tests
};

typedef L4::Types::Flags<Test_flag> Test_flags;

Test_flags x = Do_weak_tests;

if (x & Do_strong_tests) { ... }
x |= Do_strong_tests;
if (x & Do_strong_tests) { ... }
```

Definition at line 52 of file [types](#).

16.229.2 Member Enumeration Documentation

16.229.2.1 None_type

```
template<typename BITS_ENUM, typename UNDERLYING = unsigned long>
enum L4::Types::Flags::None_type
```

The none type to get an empty bitmap.

Enumerator

None	Use this to get an empty bitmap.
------	----------------------------------

Definition at line 68 of file [types](#).

16.229.3 Constructor & Destructor Documentation

16.229.3.1 Flags() [1/2]

```
template<typename BITS_ENUM, typename UNDERLYING = unsigned long>
L4::Types::Flags< BITS_ENUM, UNDERLYING >::Flags (
    None_type ) [inline]
```

Make an empty bitmap.

Usually used for implicit conversion from [Flags::None](#).

```
Flags x = Flags::None;
```

Definition at line 78 of file [types](#).

16.229.3.2 Flags() [2/2]

```
template<typename BITS_ENUM, typename UNDERLYING = unsigned long>
L4::Types::Flags< BITS_ENUM, UNDERLYING >::Flags (
    BITS_ENUM e) [inline]
```

Make flags from bit name.

Usually used for implicit conversion for a bit name.

```
Test_flags f = Do_strong_tests;
```

Definition at line 91 of file [types](#).

16.229.4 Member Function Documentation

16.229.4.1 clear()

```
template<typename BITS_ENUM, typename UNDERLYING = unsigned long>
type & L4::Types::Flags< BITS_ENUM, UNDERLYING >::clear (
    bits_enum_type flag) [inline]
```

Clear the given flag.

Parameters

<i>flag</i>	The flag that shall be cleared.
-------------	---------------------------------

`flags.clear(The_flag)` is a shortcut for `flags &= ~Flags(The_flag)`.

Definition at line 142 of file [types](#).

16.229.4.2 from_raw()

```
template<typename BITS_ENUM, typename UNDERLYING = unsigned long>
type L4::Types::Flags< BITS_ENUM, UNDERLYING >::from_raw (
    value_type v) [inline], [static]
```

Make flags from a raw value of [value_type](#).

This function may be used for example in C wrapper code.

Definition at line 98 of file [types](#).

The documentation for this class was generated from the following file:

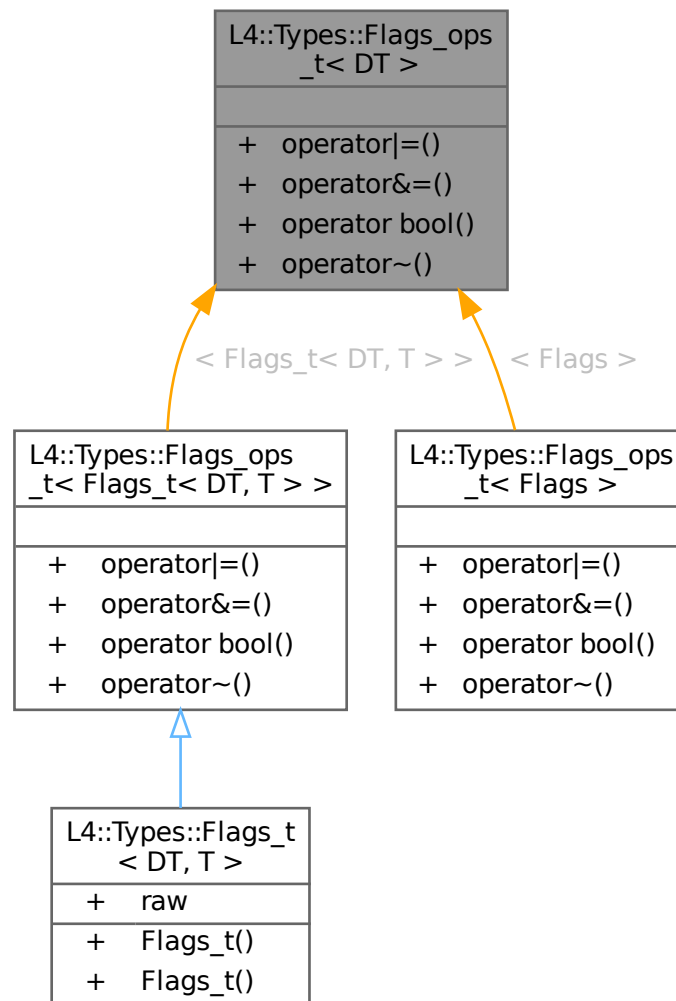
- [l4/sys/cxx/types](#)

16.230 L4::Types::Flags_ops_t< DT > Struct Template Reference

Mixin class to define a set of friend bitwise operators on `DT`.

```
#include <types>
```

Inheritance diagram for L4::Types::Flags_ops_t< DT >:



Collaboration diagram for L4::Types::Flags_ops_t< DT >:

L4::Types::Flags_ops_t< DT >
<ul style="list-style-type: none"> + operator =() + operator&=() + operator bool() + operator~()

Public Member Functions

- DT **operator|=** (DT r)
bitwise or assignment for DT
- DT **operator&=** (DT r)
bitwise and assignment for DT
- constexpr **operator bool** () const
explicit conversion to bool for tests
- constexpr DT **operator~** () const
bitwise negation for DT

Friends

- constexpr DT **operator|** (DT l, DT r)
bitwise or for DT
- constexpr DT **operator&** (DT l, DT r)
bitwise and for DT
- constexpr bool **operator==** (DT l, DT r)
equality for DT
- constexpr bool **operator!=** (DT l, DT r)
inequality for DT

16.230.1 Detailed Description

```
template<typename DT>
struct L4::Types::Flags_ops_t< DT >
```

Mixin class to define a set of friend bitwise operators on DT.

Template Parameters

<i>DT</i>	The type usually inheriting from Flags_ops_t with a member <i>raw</i> of enum or integral type.
-----------	---

Definition at line 220 of file [types](#).

The documentation for this struct was generated from the following file:

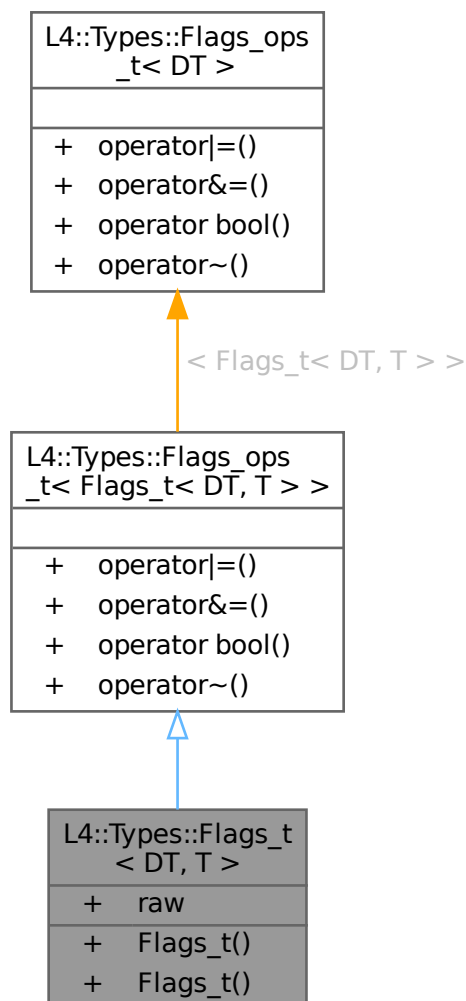
- [l4/sys/cxx/types](#)

16.231 L4::Types::Flags_t< DT, T > Struct Template Reference

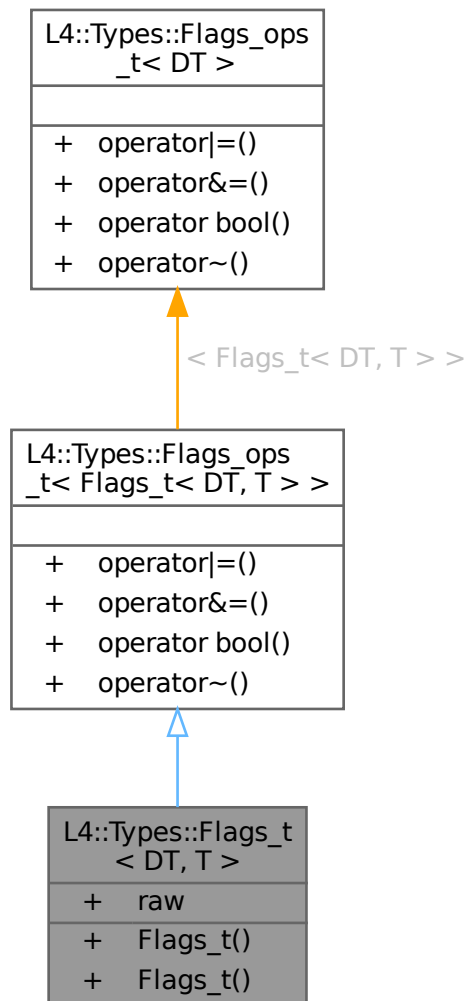
Template type to define a flags type with bitwise operations.

```
#include <types>
```

Inheritance diagram for L4::Types::Flags_t< DT, T >:



Collaboration diagram for L4::Types::Flags_t< DT, T >:



Public Member Functions

- **Flags_t**()=default
Default (uninitializing) constructor.
- constexpr **Flags_t**(T f)
Explicit initialization from the underlying type.

Public Member Functions inherited from L4::Types::Flags_ops_t< Flags_t< DT, T > >

- DT **operator|=**(DT r)
bitwise or assignment for DT
- DT **operator&=**(DT r)
bitwise and assignment for DT

- constexpr **operator bool** () const
explicit conversion to bool for tests
- constexpr DT **operator~** () const
bitwise negation for DT

Data Fields

- **T raw**
Raw integral value.

16.231.1 Detailed Description

```
template<typename DT, typename T>
struct L4::Types::Flags_t< DT, T >
```

Template type to define a flags type with bitwise operations.

Template Parameters

<i>DT</i>	determinator type to make the resulting type unique (unused).
<i>T</i>	underlying type used to store the bits, usually an integral type.

Definition at line 272 of file [types](#).

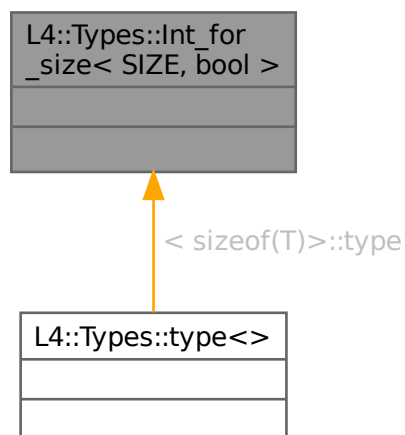
The documentation for this struct was generated from the following file:

- [l4/sys/cxx/types](#)

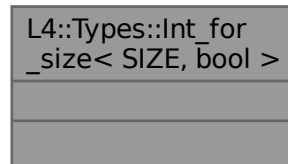
16.232 L4::Types::Int_for_size< SIZE, bool > Struct Template Reference

Metafunction to get an unsigned integral type for the given size.

Inheritance diagram for L4::Types::Int_for_size< SIZE, bool >:



Collaboration diagram for L4::Types::Int_for_size< SIZE, bool >:



16.232.1 Detailed Description

```
template<unsigned SIZE, bool = true>
struct L4::Types::Int_for_size< SIZE, bool >
```

Metafunction to get an unsigned integral type for the given size.

Template Parameters

<i>SIZE</i>	The size of the integer in bytes.
-------------	-----------------------------------

Definition at line 153 of file [types](#).

The documentation for this struct was generated from the following file:

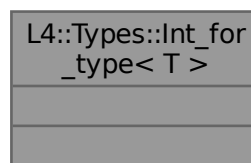
- [l4/sys/cxx/types](#)

16.233 L4::Types::Int_for_type< T > Struct Template Reference

Metafunction to get an integral type of the same size as T.

```
#include <types>
```

Collaboration diagram for L4::Types::Int_for_type< T >:



Public Types

- typedef [Int_for_size](#)< sizeof(T)>::type **type**
The resulting unsigned integer type with the size like T.

16.233.1 Detailed Description

```
template<typename T>
struct L4::Types::Int_for_type< T >
```

Metafunction to get an integral type of the same size as T.

Template Parameters

<i>T</i>	The type for which an unsigned integral type with the same size is needed.
----------	--

Definition at line [180](#) of file [types](#).

The documentation for this struct was generated from the following file:

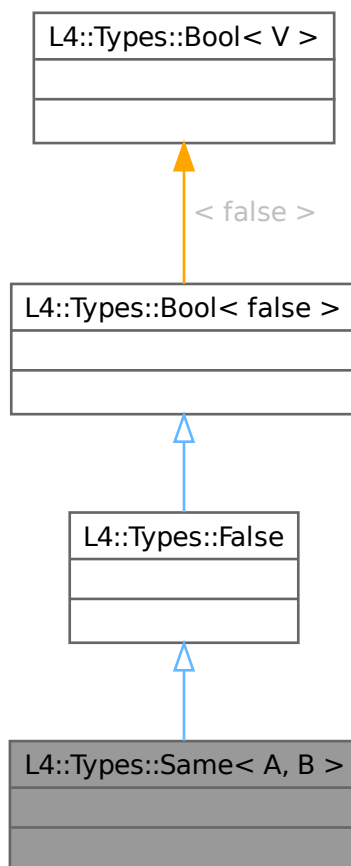
- [l4/sys/cxx/types](#)

16.234 L4::Types::Same< A, B > Struct Template Reference

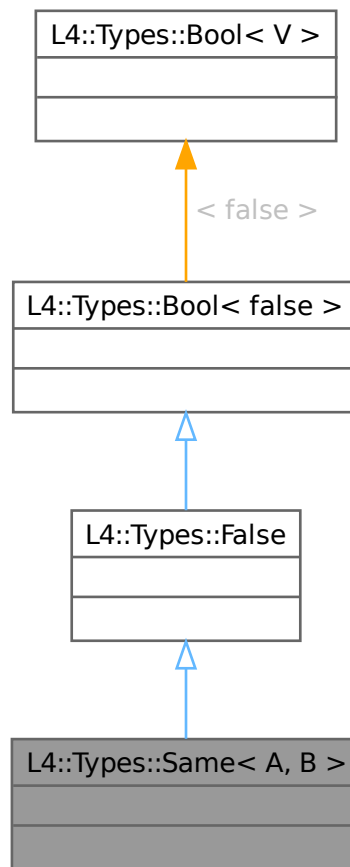
Compare two data types for equality.

```
#include <types>
```

Inheritance diagram for L4::Types::Same< A, B >:



Collaboration diagram for `L4::Types::Same< A, B >`:



Additional Inherited Members

Public Types inherited from `L4::Types::Bool< false >`

- typedef `Bool< V > type`
The meta type itself.

16.234.1 Detailed Description

```
template<typename A, typename B>
struct L4::Types::Same< A, B >
```

Compare two data types for equality.

Template Parameters

<i>A</i>	The first data type
<i>B</i>	The second data type

The result is the boolean [True](#) if A and B are the same types.

Definition at line [312](#) of file [types](#).

The documentation for this struct was generated from the following file:

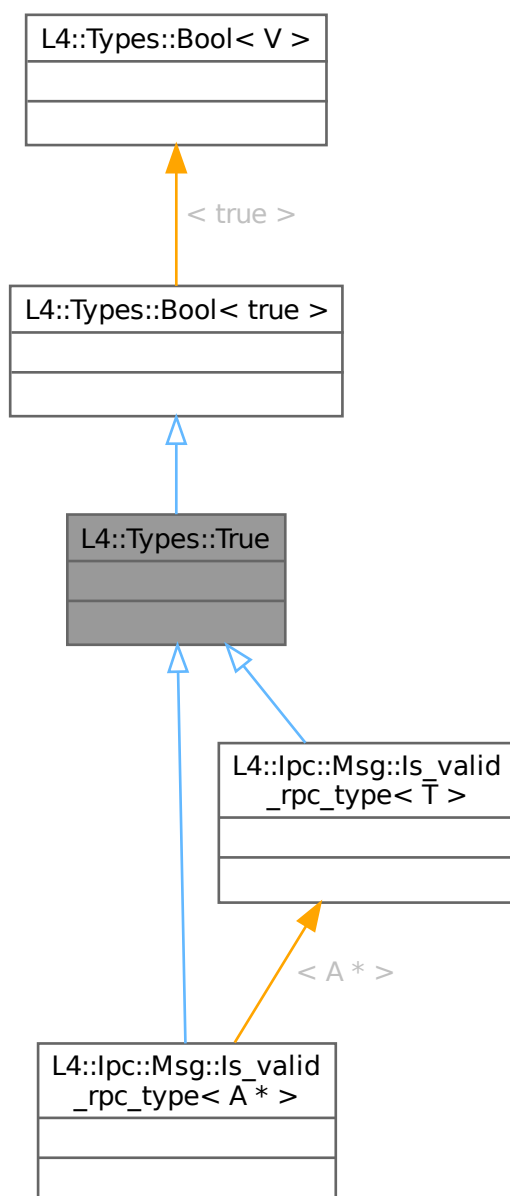
- [l4/sys/cxx/types](#)

16.235 L4::Types::True Struct Reference

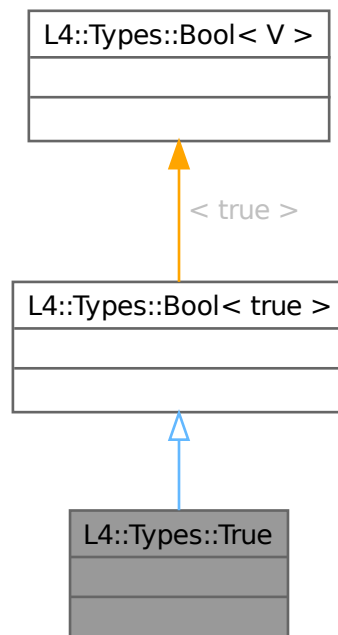
[True](#) meta value.

```
#include <types>
```

Inheritance diagram for L4::Types::True:



Collaboration diagram for L4::Types::True:



Additional Inherited Members

Public Types inherited from `L4::Types::Bool< true >`

- typedef `Bool< V >` **type**
The meta type itself.

16.235.1 Detailed Description

`True` meta value.

Definition at line 300 of file `types`.

The documentation for this struct was generated from the following file:

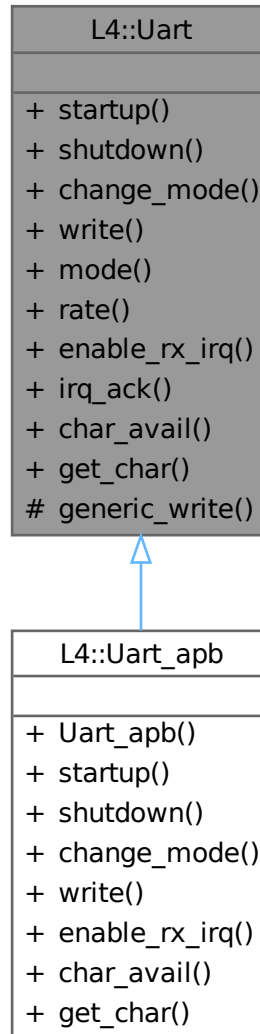
- `l4/sys/cxx/types`

16.236 L4::Uart Class Reference

[Uart](#) driver abstraction.

```
#include <uart_base.h>
```

Inheritance diagram for L4::Uart:



Collaboration diagram for L4::Uart:

L4::Uart
<ul style="list-style-type: none"> + startup() + shutdown() + change_mode() + write() + mode() + rate() + enable_rx_irq() + irq_ack() + char_avail() + get_char() # generic_write()

Public Member Functions

- virtual bool [startup](#) (Io_register_block const *regs)=0
Start the UART driver.
- virtual void [shutdown](#) ()=0
Terminate the UART driver.
- virtual bool [change_mode](#) (Transfer_mode m, Baud_rate r)=0
Set certain parameters of the UART.
- virtual int [write](#) (char const *s, unsigned long count, bool blocking=true) const =0
Transmit a number of characters.
- Transfer_mode [mode](#) () const
Return the transfer mode.
- Baud_rate [rate](#) () const
Return the baud rate.
- virtual bool [enable_rx_irq](#) (bool=true)
Enable the receive IRQ.
- virtual void [irq_ack](#) ()
Acknowledge a received interrupt.
- virtual int [char_avail](#) () const =0
Check if there is at least one character available for reading from the UART.
- virtual int [get_char](#) (bool blocking=true) const =0
Read a character from the UART.

Protected Member Functions

- template<typename Uart_driver, bool Timeout_guard = true>
int [generic_write](#) (char const *s, unsigned long count, bool blocking=true) const
Internal function transmitting each character one-after-another and finally waiting that the transmission did actually finish.

16.236.1 Detailed Description

[Uart](#) driver abstraction.

Definition at line 20 of file [uart_base.h](#).

16.236.2 Member Function Documentation

16.236.2.1 `change_mode()`

```
virtual bool L4::Uart::change_mode (  
    Transfer_mode m,  
    Baud_rate r) [pure virtual]
```

Set certain parameters of the UART.

Parameters

<i>m</i>	UART mode. Depends on the hardware.
<i>r</i>	Baud rate.

Return values

<i>true</i>	Mode setting succeeded (or was not performed at all).
<i>false</i>	Mode setting failed for some reason.

Note

Some drivers don't perform any mode setting at all and just return true.

Implemented in [L4::Uart_apb](#).

16.236.2.2 `char_avail()`

```
virtual int L4::Uart::char_avail () const [pure virtual]
```

Check if there is at least one character available for reading from the UART.

Returns

0 if there is no character available for reading, !=0 otherwise.

Implemented in [L4::Uart_apb](#).

16.236.2.3 `enable_rx_irq()`

```
virtual bool L4::Uart::enable_rx_irq (  
    bool = true) [inline], [virtual]
```

Enable the receive IRQ.

Return values

<i>true</i>	The RX IRQ was successfully enabled / disabled.
<i>false</i>	The RX IRQ couldn't be enabled / disabled. The driver does not support this operation.

Reimplemented in [L4::Uart_apb](#).

Definition at line 103 of file [uart_base.h](#).

16.236.2.4 generic_write()

```
template<typename Uart_driver, bool Timeout_guard = true>
int L4::Uart::generic_write (
    char const * s,
    unsigned long count,
    bool blocking = true) const [inline], [protected]
```

Internal function transmitting each character one-after-another and finally waiting that the transmission did actually finish.

Parameters

<i>s</i>	Buffer containing the characters.
<i>count</i>	The number of characters to transmit.
<i>blocking</i>	If true, wait until there is space in the transmit buffer and also wait until every character was successful transmitted. Otherwise do not wait.

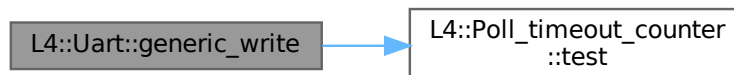
Returns

The number of successful written characters.

Definition at line 141 of file [uart_base.h](#).

References [L4::Poll_timeout_counter::test\(\)](#).

Here is the call graph for this function:



16.236.2.5 get_char()

```
virtual int L4::Uart::get_char (
    bool blocking = true) const [pure virtual]
```

Read a character from the UART.

Parameters

<i>blocking</i>	If true, wait until a character is available for reading. Otherwise do not wait and just return -1 if no character is available.
-----------------	--

Returns

The actual character read from the UART.

Implemented in [L4::Uart_apb](#).

16.236.2.6 mode()

```
Transfer_mode L4::Uart::mode () const [inline]
```

Return the transfer mode.

Returns

The transfer mode.

Definition at line 86 of file [uart_base.h](#).

16.236.2.7 rate()

```
Baud_rate L4::Uart::rate () const [inline]
```

Return the baud rate.

Returns

The baud rate.

Definition at line 93 of file [uart_base.h](#).

16.236.2.8 shutdown()

```
virtual void L4::Uart::shutdown () [pure virtual]
```

Terminate the UART driver.

This includes disabling of interrupts.

Implemented in [L4::Uart_apb](#).

16.236.2.9 startup()

```
virtual bool L4::Uart::startup (
    Io_register_block const * regs) [pure virtual]
```

Start the UART driver.

Parameters

<i>regs</i>	IO register block of the UART.
-------------	--------------------------------

Return values

<i>true</i>	Startup succeeded.
<i>false</i>	Startup failed.

Implemented in [L4::Uart_apb](#).

16.236.2.10 write()

```
virtual int L4::Uart::write (
    char const * s,
    unsigned long count,
    bool blocking = true) const [pure virtual]
```

Transmit a number of characters.

Parameters

<i>s</i>	Buffer containing the characters.
<i>count</i>	Number of characters to transmit.
<i>blocking</i>	If true, wait until there is space in the transmit buffer and also wait until every character was successful transmitted. Otherwise do not wait.

Returns

The number of successfully written characters.

Implemented in [L4::Uart_apb](#).

The documentation for this class was generated from the following file:

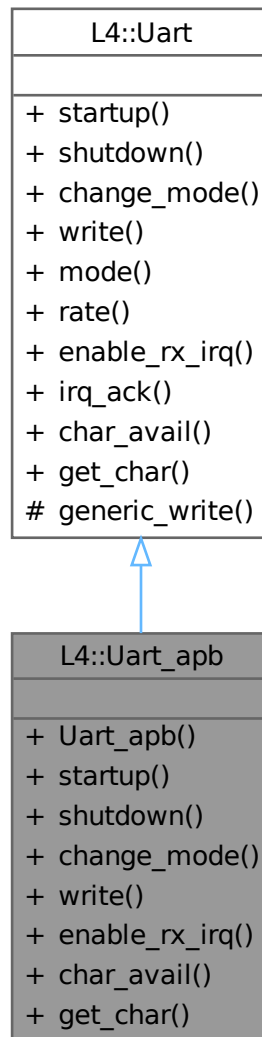
- pkg/drivers-frst/uart/include/uart_base.h

16.237 L4::Uart_apb Class Reference

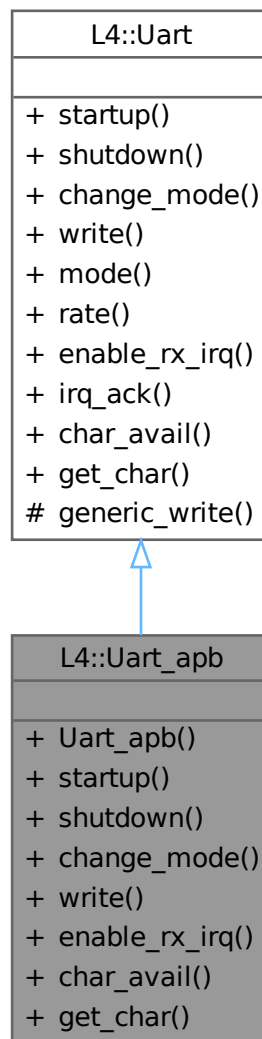
Driver for the Advanced Peripheral Bus (APB) UART from the Cortex-M System Design Kit (CMSDK).

```
#include <uart_apb.h>
```

Inheritance diagram for L4::Uart_apb:



Collaboration diagram for L4::Uart_apb:



Public Member Functions

- **Uart_apb** (unsigned freq)
freq == 0 means unknown and don't change baud rate
- bool **startup** (lo_register_block const *) override
Start the UART driver.
- void **shutdown** () override
Terminate the UART driver.
- bool **change_mode** (Transfer_mode m, Baud_rate r) override
Set certain parameters of the UART.
- int **write** (char const *s, unsigned long count, bool blocking=true) const override
Transmit a number of characters.

- bool `enable_rx_irq` (bool enable) override
Enable the receive IRQ.
- int `char_avail` () const override
Check if there is at least one character available for reading from the UART.
- int `get_char` (bool blocking=true) const override
Read a character from the UART.

Public Member Functions inherited from `L4::Uart`

- Transfer_mode `mode` () const
Return the transfer mode.
- Baud_rate `rate` () const
Return the baud rate.
- virtual void `irq_ack` ()
Acknowledge a received interrupt.

Additional Inherited Members

Protected Member Functions inherited from `L4::Uart`

- template<typename Uart_driver, bool Timeout_guard = true>
int `generic_write` (char const *s, unsigned long count, bool blocking=true) const
Internal function transmitting each character one-after-another and finally waiting that the transmission did actually finish.

16.237.1 Detailed Description

Driver for the Advanced Peripheral Bus (APB) UART from the Cortex-M System Design Kit (CMSDK).

Definition at line 17 of file `uart_apb.h`.

16.237.2 Member Function Documentation

16.237.2.1 `change_mode()`

```
bool L4::Uart_apb::change_mode (
    Transfer_mode m,
    Baud_rate r) [override], [virtual]
```

Set certain parameters of the UART.

Parameters

<i>m</i>	UART mode. Depends on the hardware.
<i>r</i>	Baud rate.

Return values

<i>true</i>	Mode setting succeeded (or was not performed at all).
<i>false</i>	Mode setting failed for some reason.

Note

Some drivers don't perform any mode setting at all and just return true.

Implements [L4::Uart](#).

16.237.2.2 char_avail()

```
int L4::Uart_apb::char_avail () const [override], [virtual]
```

Check if there is at least one character available for reading from the UART.

Returns

0 if there is no character available for reading, !=0 otherwise.

Implements [L4::Uart](#).

16.237.2.3 enable_rx_irq()

```
bool L4::Uart_apb::enable_rx_irq (
    bool ) [override], [virtual]
```

Enable the receive IRQ.

Return values

<i>true</i>	The RX IRQ was successfully enabled / disabled.
<i>false</i>	The RX IRQ couldn't be enabled / disabled. The driver does not support this operation.

Reimplemented from [L4::Uart](#).

16.237.2.4 get_char()

```
int L4::Uart_apb::get_char (
    bool blocking = true) const [override], [virtual]
```

Read a character from the UART.

Parameters

<i>blocking</i>	If true, wait until a character is available for reading. Otherwise do not wait and just return -1 if no character is available.
-----------------	--

Returns

The actual character read from the UART.

Implements [L4::Uart](#).

16.237.2.5 shutdown()

```
void L4::Uart_apb::shutdown () [override], [virtual]
```

Terminate the UART driver.

This includes disabling of interrupts.

Implements [L4::Uart](#).

16.237.2.6 startup()

```
bool L4::Uart_apb::startup (
    Io_register_block const * regs) [override], [virtual]
```

Start the UART driver.

Parameters

<i>regs</i>	IO register block of the UART.
-------------	--------------------------------

Return values

<i>true</i>	Startup succeeded.
<i>false</i>	Startup failed.

Implements [L4::Uart](#).

16.237.2.7 write()

```
int L4::Uart_apb::write (
    char const * s,
    unsigned long count,
    bool blocking = true) const [override], [virtual]
```

Transmit a number of characters.

Parameters

<i>s</i>	Buffer containing the characters.
<i>count</i>	Number of characters to transmit.
<i>blocking</i>	If true, wait until there is space in the transmit buffer and also wait until every character was successful transmitted. Otherwise do not wait.

Returns

The number of successfully written characters.

Implements [L4::Uart](#).

The documentation for this class was generated from the following file:

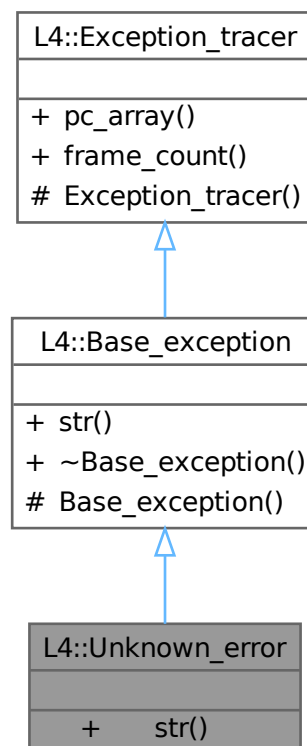
- pkg/drivers-frst/uart/include/uart_apb.h

16.238 L4::Unknown_error Class Reference

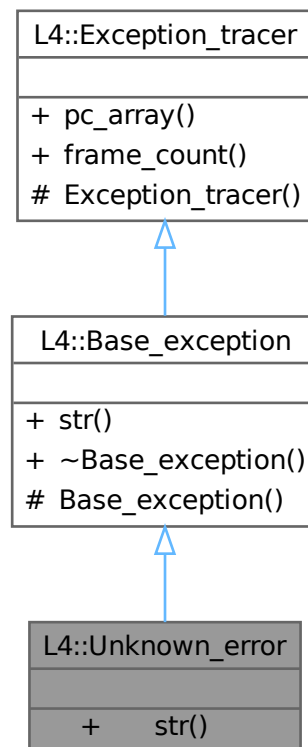
[Exception](#) for an unknown condition.

```
#include <l4/cxx/exceptions>
```

Inheritance diagram for L4::Unknown_error:



Collaboration diagram for L4::Unknown_error:



Public Member Functions

- `char const * str ()` `const noexcept` override
Return a human readable string for the exception.

Public Member Functions inherited from [L4::Base_exception](#)

- `virtual ~Base_exception ()` `noexcept`
Destruction.

Public Member Functions inherited from [L4::Exception_tracer](#)

- `void const *const * pc_array ()` `const noexcept`
Get the array containing the call trace.
- `int frame_count ()` `const noexcept`
Get the number of entries that are valid in the call trace.

Additional Inherited Members

Protected Member Functions inherited from [L4::Base_exception](#)

- **Base_exception** () noexcept
Create a base exception.

Protected Member Functions inherited from [L4::Exception_tracer](#)

- **Exception_tracer** () noexcept
Create a back trace.

16.238.1 Detailed Description

[Exception](#) for an unknown condition.

This error is usually used when a server returns an unknown return state to the client, this may indicate incompatible messages used by the client and the server.

Definition at line 208 of file [exceptions](#).

The documentation for this class was generated from the following file:

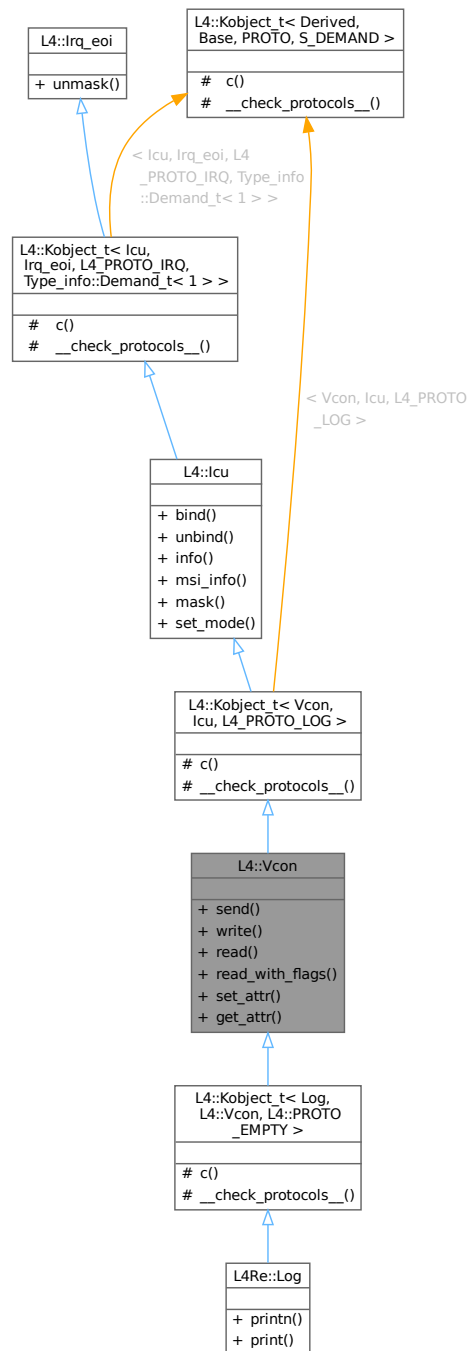
- [l4/cxx/exceptions](#)

16.239 L4::Vcon Class Reference

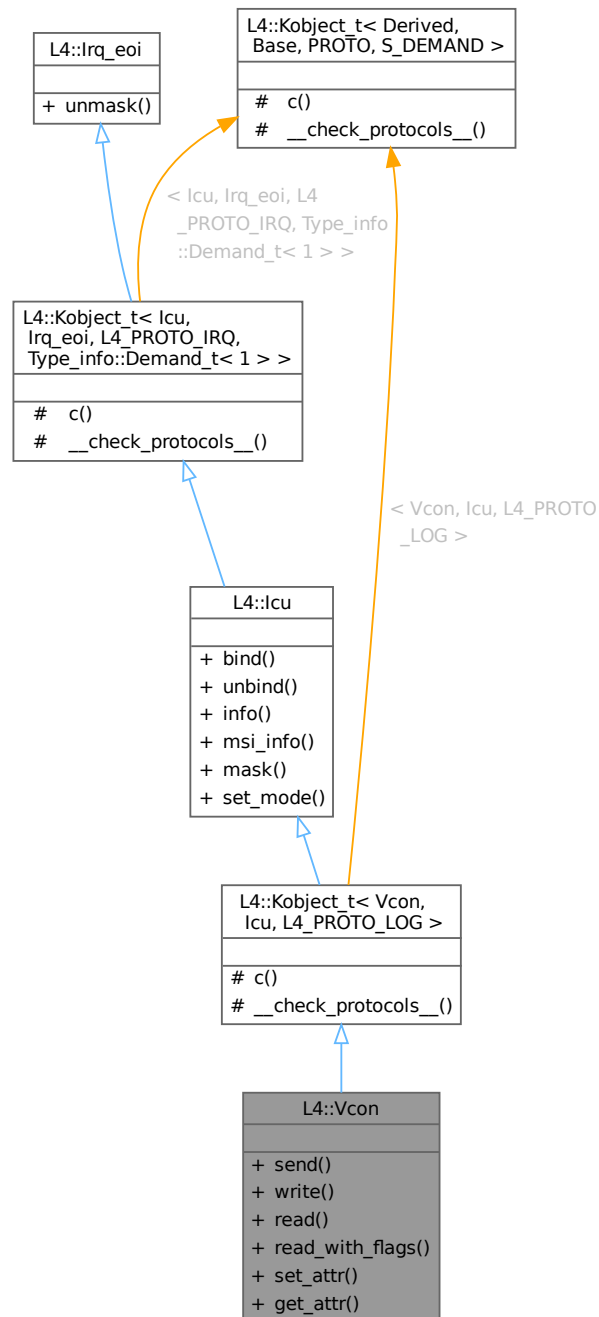
C++ [L4 Vcon](#) interface, see [Virtual Console](#) for the C interface.

```
#include <vcon>
```

Inheritance diagram for L4::Vcon:



Collaboration diagram for L4::Vcon:



Public Member Functions

- `l4_msgtag_t send` (char const *buf, unsigned size, `l4_utcb_t *utcb=l4_utcb()`) const noexcept
Send data to *this* virtual console.
- `long write` (char const *buf, unsigned size, `l4_utcb_t *utcb=l4_utcb()`) const noexcept
Write data to *this* virtual console.
- `int read` (char *buf, unsigned size, `l4_utcb_t *utcb=l4_utcb()`) const noexcept

- *Read data from `this` virtual console.*
- `int read_with_flags (char *buf, unsigned size, l4_utcb_t *utcb=l4_utcb\(\)) const noexcept`
Read data from `this` virtual console which also returns flags.
- `l4_msgtag_t set_attr (l4_vcon_attr_t const *attr, l4_utcb_t *utcb=l4_utcb\(\)) const noexcept`
Set the attributes of `this` virtual console.
- `l4_msgtag_t get_attr (l4_vcon_attr_t *attr, l4_utcb_t *utcb=l4_utcb\(\)) const noexcept`
Get attributes of `this` virtual console.

Public Member Functions inherited from [L4::Icu](#)

- `l4_msgtag_t bind (unsigned irqnum, L4::Cap< Triggerable > irq, l4_utcb_t *utcb=l4_utcb\(\)) noexcept`
Bind an interrupt line of an interrupt controller to an interrupt object.
- `l4_msgtag_t unbind (unsigned irqnum, L4::Cap< Triggerable > irq, l4_utcb_t *utcb=l4_utcb\(\)) noexcept`
Remove binding of an interrupt line from the interrupt controller object.
- `l4_msgtag_t info (l4_icu_info_t *info, l4_utcb_t *utcb=l4_utcb\(\)) noexcept`
Get information about the ICU features.
- `l4_msgtag_t msi_info (l4_umword_t irqnum, l4_uint64_t source, l4_icu_msi_info_t *msi_info)`
Get MSI info about IRQ.
- `l4_msgtag_t mask (unsigned irqnum, l4_umword_t *label=0, l4_timeout_t to=L4_IPC_NEVER, l4_utcb_t *utcb=l4_utcb\(\)) noexcept`
Mask an IRQ line.
- `l4_msgtag_t set_mode (unsigned irqnum, l4_umword_t mode, l4_utcb_t *utcb=l4_utcb\(\)) noexcept`
Set interrupt mode.

Public Member Functions inherited from [L4::Irq_eoi](#)

- `l4_msgtag_t unmask (unsigned irqnum, l4_umword_t *label=0, l4_timeout_t to=L4_IPC_NEVER, l4_utcb_t *utcb=l4_utcb\(\)) noexcept`
Unmask the given interrupt line.

Additional Inherited Members

Protected Types inherited from [L4::Kobject_t](#)< [Vcon](#), [Icu](#), [L4_PROTO_LOG](#) >

- `typedef Vcon Class`
The target interface type (inheriting from [Kobject_t](#)).
- `typedef Typeid::Iface< PROTO, Vcon > __Iface`
The interface description for the derived class.
- `typedef Typeid::Merge_list< Typeid::Iface_list< __Iface >, typename Icu::__Iface_list > __Iface_list`
The list of all RPC interfaces provided directly or through inheritance.

Protected Types inherited from

[L4::Kobject_t](#)< [Icu](#), [Irq_eoi](#), [L4_PROTO_IRQ](#), [Type_info::Demand_t](#)< 1 > >

- `typedef Icu Class`
The target interface type (inheriting from [Kobject_t](#)).
- `typedef Typeid::Iface< PROTO, Icu > __Iface`
The interface description for the derived class.
- `typedef Typeid::Merge_list< Typeid::Iface_list< __Iface >, typename Irq_eoi::__Iface_list > __Iface_list`
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from [L4::Kobject_t< Vcon, Icu, L4_PROTO_LOG >](#)

- [L4::Cap< Class > c \(\)](#) const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from [L4::Kobject_t< Icu, Irq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > >](#)

- [L4::Cap< Class > c \(\)](#) const noexcept
Get the capability to ourselves.

Static Protected Member Functions inherited from [L4::Kobject_t< Vcon, Icu, L4_PROTO_LOG >](#)

- static void [__check_protocols__ \(\)](#) noexcept
Helper to check for protocol conflicts.

Static Protected Member Functions inherited from [L4::Kobject_t< Icu, Irq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > >](#)

- static void [__check_protocols__ \(\)](#) noexcept
Helper to check for protocol conflicts.

16.239.1 Detailed Description

C++ [L4 Vcon](#) interface, see [Virtual Console](#) for the C interface.

[L4::Vcon](#) is a virtual console for simple character-based input and output. The interrupt for read events is provided by the virtual key interrupt.

The [Vcon](#) interface inherits from [L4::Icu](#) and [L4::Irq_eoi](#) for managing the virtual key interrupt which, in contrast to hardware IRQs, implements a limited functionality:

- Only IRQ line 0 is supported, no MSI vectors.
- The IRQ is edge-triggered and the IRQ mode cannot be changed.
- As the IRQ is edge-triggered, it does not have to be explicitly unmasked.

A server implementing the virtual console protocol has a queue for input events. When the first input event is added to the empty queue, the virtual key interrupt is triggered. Further events are added to the queue without generating further interrupts. The queue is emptied when a client reads all queued input events.

Include File

```
#include <l4/sys/vcon>
```

See the [Virtual Console](#) for the C interface.

Definition at line 45 of file [vcon](#).

16.239.2 Member Function Documentation

16.239.2.1 get_attr()

```
l4_msgtag_t L4::Vcon::get_attr (
    l4_vcon_attr_t * attr,
    l4_utcb_t * utcb = l4_utcb()) const [inline], [noexcept]
```

Get attributes of this virtual console.

Parameters

out	<i>attr</i>	Attribute structure. Contains the attributes after a successful call of this function.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

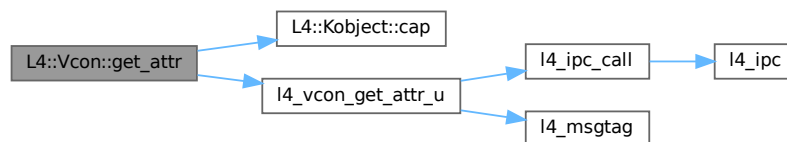
Returns

Syscall return tag.

Definition at line 151 of file [vcon](#).

References [L4::Kobject::cap\(\)](#), and [l4_vcon_get_attr_u\(\)](#).

Here is the call graph for this function:

**16.239.2.2 read()**

```

int L4::Vcon::read (
    char * buf,
    unsigned size,
    l4_utcb_t * utcb = l4_utcb()) const [inline], [noexcept]

```

Read data from this virtual console.

Parameters

out	<i>buf</i>	Pointer to data buffer.
	<i>size</i>	Size of the data buffer in bytes.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Return values

<code>-L4_EPERM</code>	Insufficient permissions; see precondition.
<code>>size</code>	More bytes to read, <code>size</code> bytes are in the buffer <code>buf</code> .
<code><=size</code>	Number of bytes read.

Precondition

The invoked [Vcon](#) capability must have the permission [L4_CAP_FPAGE_W](#).

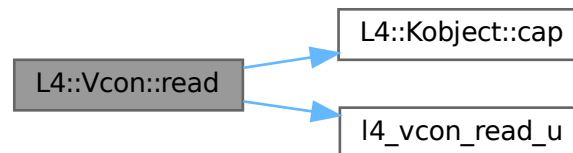
Note

Size must not exceed [L4_VCON_READ_SIZE](#).

Definition at line 98 of file [vcon](#).

References [L4::Kobject::cap\(\)](#), and [l4_vcon_read_u\(\)](#).

Here is the call graph for this function:

**16.239.2.3 read_with_flags()**

```

int L4::Vcon::read_with_flags (
    char * buf,
    unsigned size,
    l4_utcb_t * utcb = l4_utcb()) const [inline], [noexcept]
  
```

Read data from this virtual console which also returns flags.

Parameters

out	<i>buf</i>	Pointer to data buffer.
	<i>size</i>	Size of the data buffer in bytes.
	<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

Return values

<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i>>size</i>	More bytes to read, <i>size</i> bytes are in the buffer <i>buf</i> .
<i><=size</i>	Number of bytes read.

Precondition

The invoked [Vcon](#) capability must have the permission [L4_CAP_FPAGE_W](#).

If this function returns a positive value the caller can check the [L4_VCON_READ_STAT_BREAK](#) flag bit for a break condition. The bytes read can be obtained by masking the return value with [L4_VCON_READ_SIZE_MASK](#).

If a break condition is signaled, it is always the first event in the transmitted content, i.e. all characters supplied by this read call follow the break condition.

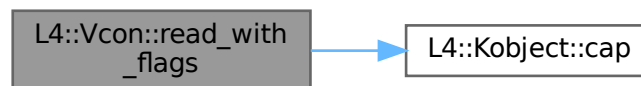
Note

Size must not exceed [L4_VCON_READ_SIZE](#).

Definition at line 125 of file [vcon](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:

**16.239.2.4 send()**

```

14_msgtag_t L4::Vcon::send (
    char const * buf,
    unsigned size,
    14_utcb_t * utcb = 14_utcb()) const [inline], [noexcept]
  
```

Send data to `this` virtual console.

Parameters

<i>buf</i>	Pointer to the data buffer.
<i>size</i>	Size of the data buffer in bytes.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to 14_utcb .

Returns

Syscall return tag

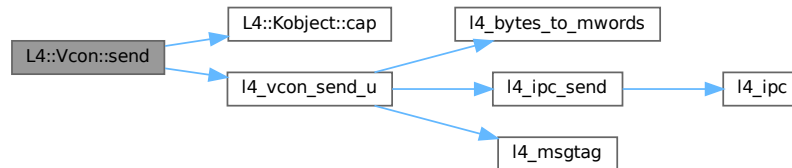
Note

Size must not exceed [L4_VCON_WRITE_SIZE](#), a proper value of the `size` parameter is NOT checked. Also, this function is a send only operation, this means there is no return value except for a failed send operation. Use [l4_ipc_error\(\)](#) to check for send errors, do not use [l4_error\(\)](#), as [l4_error\(\)](#) will always return an error.

Definition at line 65 of file [vcon](#).

References [L4::Kobject::cap\(\)](#), and [l4_vcon_send_u\(\)](#).

Here is the call graph for this function:

**16.239.2.5 set_attr()**

```

l4_msgtag_t L4::Vcon::set_attr (
    l4_vcon_attr_t const * attr,
    l4_utcb_t * utcb = l4_utcb()) const [inline], [noexcept]
  
```

Set the attributes of this virtual console.

Parameters

<i>attr</i>	Attribute structure with the attributes for the virtual console.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

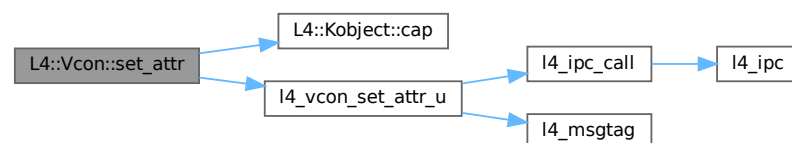
Returns

Syscall return tag.

Definition at line 138 of file [vcon](#).

References [L4::Kobject::cap\(\)](#), and [l4_vcon_set_attr_u\(\)](#).

Here is the call graph for this function:



16.239.2.6 write()

```
long L4::Vcon::write (
    char const * buf,
    unsigned size,
    l4_utcb_t * utcb = l4_utcb()) const [inline], [noexcept]
```

Write data to this virtual console.

Parameters

<i>buf</i>	Pointer to the data buffer.
<i>size</i>	Size of the data buffer in bytes.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. Defaults to l4_utcb .

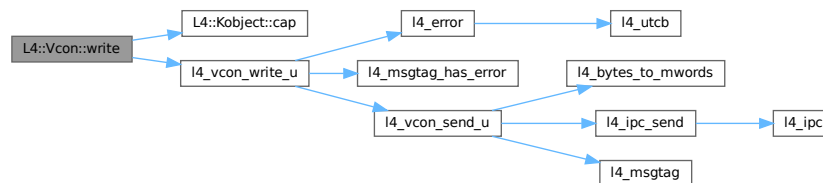
Return values

< 0	Error.
>= 0	Number of bytes written to the virtual console.

Definition at line 79 of file [vcon](#).

References [L4::Kobject::cap\(\)](#), and [l4_vcon_write_u\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

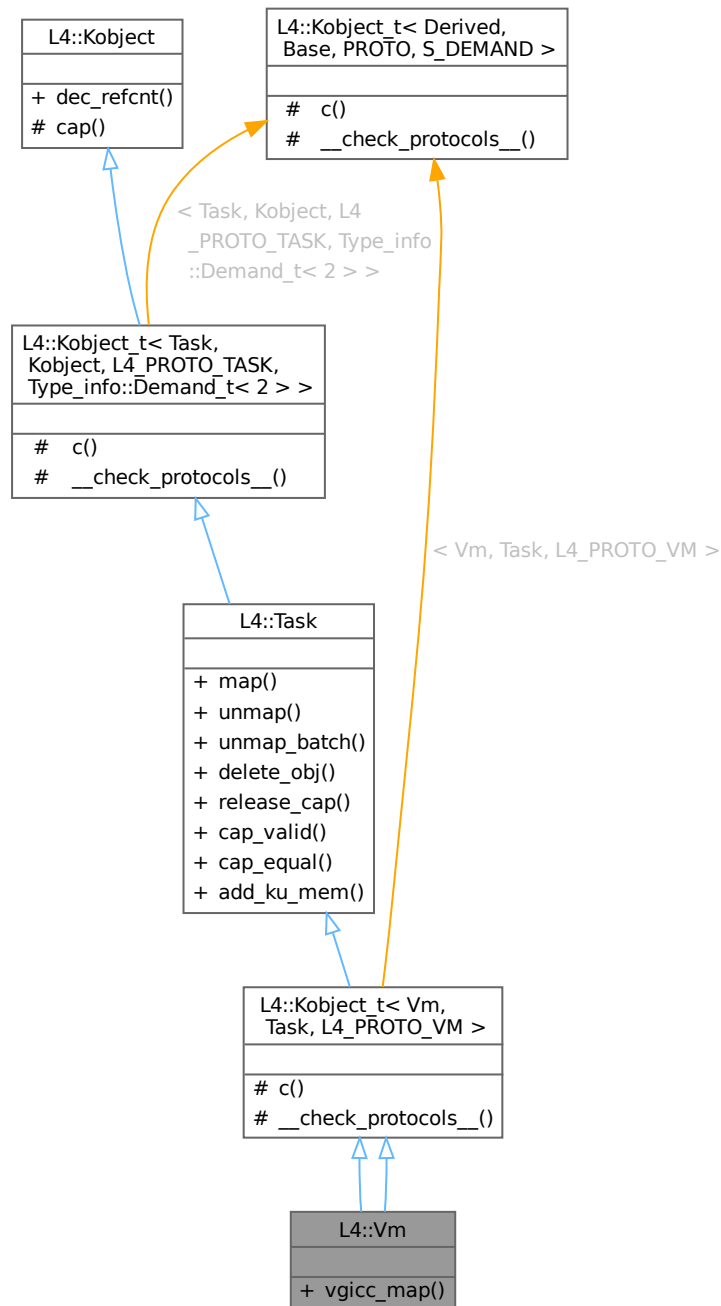
- [l4/sys/vcon](#)

16.240 L4::Vm Class Reference

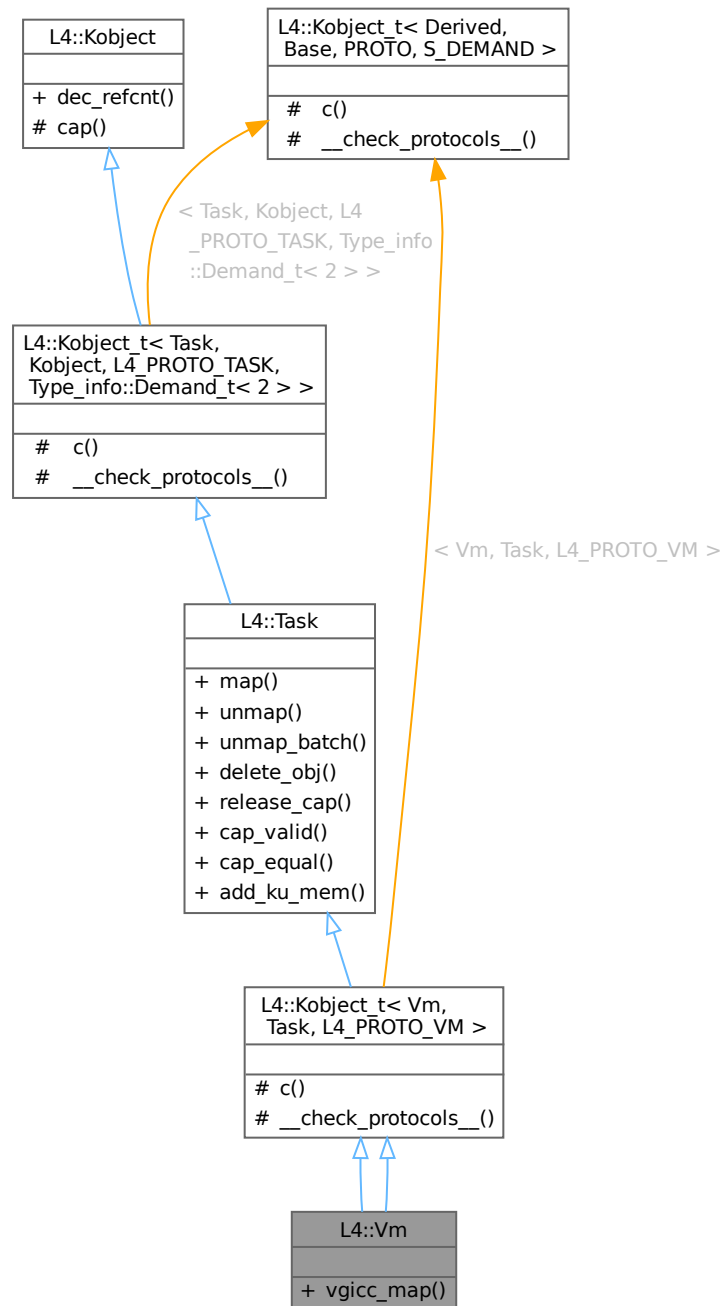
Virtual machine host address space.

```
#include <vm>
```

Inheritance diagram for L4::Vm:



Collaboration diagram for L4::Vm:



Public Member Functions

- `l4_msgtag_t vgicc_map(l4_fpage_t const vgicc_fpage, l4_utcb_t *utcb=l4_utcb()) noexcept`
Map the GIC virtual CPU interface page to the task in case Fiasco detected a GIC version 2.

Public Member Functions inherited from [L4::Task](#)

- [l4_msgtag_t map](#) ([Cap](#)< [Task](#) > const &src_task, [l4_fpage_t](#) const &snd_fpage, [l4_umword_t](#) snd_base, [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept
Map resources available in the source task to a destination task.
- [l4_msgtag_t unmap](#) ([l4_fpage_t](#) const &fpage, [l4_umword_t](#) map_mask, [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept
Revoke rights from the task.
- [l4_msgtag_t unmap_batch](#) ([l4_fpage_t](#) const *fpages, unsigned num_fpages, [l4_umword_t](#) map_mask, [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept
Revoke rights from a task.
- [l4_msgtag_t delete_obj](#) ([L4::Cap](#)< void > obj, [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept
Release capability and delete object.
- [l4_msgtag_t release_cap](#) ([L4::Cap](#)< void > cap, [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept
Release object capability.
- [l4_msgtag_t cap_valid](#) ([Cap](#)< void > const &cap, [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept
Check whether a capability is present (refers to an object).
- [l4_msgtag_t cap_equal](#) ([Cap](#)< void > const &cap_a, [Cap](#)< void > const &cap_b, [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept
Test whether two capabilities point to the same object with the same permissions (only considering selected permissions).
- [l4_msgtag_t add_ku_mem](#) ([l4_fpage_t](#) *fpage, [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept
Add kernel-user memory.

Public Member Functions inherited from [L4::Kobject](#)

- [l4_msgtag_t dec_refcnt](#) ([l4_mword_t](#) diff, [l4_utcb_t](#) *utcb=[l4_utcb](#)())
Decrement the in kernel reference counter for the object.

Additional Inherited Members

Protected Types inherited from [L4::Kobject_t](#)< [Vm](#), [Task](#), [L4_PROTO_VM](#) >

- typedef [Vm](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< [PROTO](#), [Vm](#) > **__iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__iface** >, typename [Task](#)::__Iface_list > **__iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Types inherited from

[L4::Kobject_t](#)< [Task](#), [Kobject](#), [L4_PROTO_TASK](#), [Type_info::Demand_t](#)< 2 > >

- typedef [Task](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< [PROTO](#), [Task](#) > **__iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__iface** >, typename [Kobject](#)::__Iface_list > **__iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from [L4::Kobject_t< Vm, Task, L4_PROTO_VM >](#)

- [L4::Cap< Class > c\(\)](#) const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from [L4::Kobject_t< Task, Kobject, L4_PROTO_TASK, Type_info::Demand_t< 2 > >](#)

- [L4::Cap< Class > c\(\)](#) const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from [L4::Kobject](#)

- [l4_cap_idx_t cap\(\)](#) const noexcept
Return capability selector.

Static Protected Member Functions inherited from [L4::Kobject_t< Vm, Task, L4_PROTO_VM >](#)

- static void [__check_protocols__\(\)](#) noexcept
Helper to check for protocol conflicts.

Static Protected Member Functions inherited from [L4::Kobject_t< Task, Kobject, L4_PROTO_TASK, Type_info::Demand_t< 2 > >](#)

- static void [__check_protocols__\(\)](#) noexcept
Helper to check for protocol conflicts.

16.240.1 Detailed Description

Virtual machine host address space.

[L4::Vm](#) is a specialisation of [L4::Task](#), used for virtual machines. The microkernel employs an appropriate page-table format for hosting VMs, such as ePT on VT-x. On Arm, it offers a call to make the virtual GICC area available to the VM.

Definition at line 17 of file [__vm-arm.h](#).

16.240.2 Member Function Documentation

16.240.2.1 [vgicc_map\(\)](#)

```
l4_msgtag_t L4::Vm::vgicc_map (
    l4_fpage_t const vgicc_fpage,
    l4_utcb_t * utcb = l4_utcb()) [inline], [noexcept]
```

Map the GIC virtual CPU interface page to the task in case Fiasco detected a GIC version 2.

Parameters

<i>vgicc_fpage</i>	Flexpage that describes an area in the address space of the destination task to map the vGICC page to.
<i>utcb</i>	UTCB to be used for this operation, shall be the UTCB of the calling thread. See l4_utcb .

Returns

Syscall return tag.

Definition at line 30 of file [__vm-arm.h](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following files:

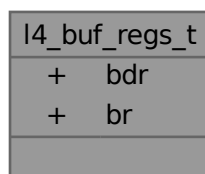
- [l4/sys/__vm-arm.h](#)
- [l4/sys/vm](#)

16.241 l4_buf_regs_t Struct Reference

Encapsulation of the buffer-registers block in the UTCB.

```
#include <utcb.h>
```

Collaboration diagram for `l4_buf_regs_t`:



Data Fields

- [l4_umword_t](#) **bdr**
Buffer descriptor.
- [l4_umword_t](#) **br** [L4_UTCB_GENERIC_BUFFERS_SIZE]
Buffer registers.

16.241.1 Detailed Description

Encapsulation of the buffer-registers block in the UTCB.

Definition at line 82 of file [utcb.h](#).

The documentation for this struct was generated from the following file:

- l4/sys/[utcb.h](#)

16.242 l4_exc_regs_t Struct Reference

UTCB structure for exceptions.

```
#include <utcb.h>
```

Collaboration diagram for l4_exc_regs_t:

l4_exc_regs_t
+ pfa
+ err
+ r
+ sp
+ ulr
+ _dummy1
+ pc
+ cpsr
+ tpidrro
+ tpidrurw
and 31 more...

Data Fields

- [l4_umword_t pfa](#)
page fault address
- [l4_umword_t err](#)
error code
- [l4_umword_t r \[13\]](#)
registers
- [l4_umword_t sp](#)
stack pointer
- [l4_umword_t ulr](#)
ulr
- [l4_umword_t _dummy1](#)
dummy
- [l4_umword_t pc](#)
pc
- [l4_umword_t cpsr](#)
cpsr
- [l4_umword_t tpidruro](#)
Thread-ID register.
- [l4_umword_t tpidrurw](#)
Thread-ID register.
- [l4_umword_t r15](#)
r15
- [l4_umword_t r14](#)
r14
- [l4_umword_t r13](#)
r13
- [l4_umword_t r12](#)
r12
- [l4_umword_t r11](#)
r11
- [l4_umword_t r10](#)
r10
- [l4_umword_t r9](#)
r9
- [l4_umword_t r8](#)
r8
- [l4_umword_t rdi](#)
rdi
- [l4_umword_t rsi](#)
rsi
- [l4_umword_t rbp](#)
rbp
- [l4_umword_t rbx](#)
rbx
- [l4_umword_t rdx](#)
rdx
- [l4_umword_t rcx](#)
rcx
- [l4_umword_t rax](#)

- rax*
- [l4_umword_t trapno](#)
trap number
- [l4_umword_t dummy1](#)
dummy
- [l4_umword_t ss](#)
stack segment register
- [l4_umword_t es](#)
es register
- [l4_umword_t ds](#)
ds register
- [l4_umword_t gs](#)
gs register
- [l4_umword_t fs](#)
fs register
- [l4_umword_t edi](#)
edi register
- [l4_umword_t esi](#)
esi register
- [l4_umword_t ebp](#)
ebp register
- [l4_umword_t ebx](#)
ebx register
- [l4_umword_t edx](#)
edx register
- [l4_umword_t ecx](#)
ecx register
- [l4_umword_t eax](#)
eax register

16.242.1 Detailed Description

UTCB structure for exceptions.

Examples

[examples/sys/aliens/main.c](#), [examples/sys/singlestep/main.c](#), and [examples/sys/start-with-exc/main.c](#).

Definition at line 27 of file [utcb.h](#).

16.242.2 Field Documentation

16.242.2.1 flags

```
l4\_umword\_t l4_exc_regs_t::flags
```

rflags

eflags

Definition at line 37 of file [utcb.h](#).

16.242.2.2 ss

`l4_umword_t l4_exc_regs_t::ss`

stack segment register

ss register

Definition at line 71 of file [utcb.h](#).

The documentation for this struct was generated from the following files:

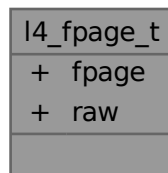
- [arm/l4/sys/utcb.h](#)
- [arm64/l4/sys/utcb.h](#)
- [amd64/l4/sys/utcb.h](#)
- [x86/l4/sys/utcb.h](#)

16.243 l4_fpage_t Union Reference

[L4](#) flexpage type.

```
#include <__l4_fpage.h>
```

Collaboration diagram for `l4_fpage_t`:



Data Fields

- [l4_umword_t](#) **fpage**
Raw value.
- [l4_umword_t](#) **raw**
Raw value.

16.243.1 Detailed Description

[L4](#) flexpage type.

Definition at line 76 of file [__l4_fpage.h](#).

The documentation for this union was generated from the following file:

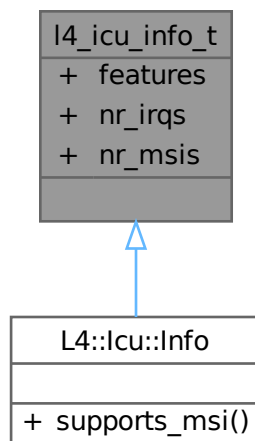
- [l4/sys/__l4_fpage.h](#)

16.244 l4_icu_info_t Struct Reference

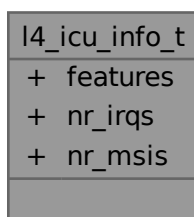
Info structure for an ICU.

```
#include <icu.h>
```

Inheritance diagram for l4_icu_info_t:



Collaboration diagram for l4_icu_info_t:



Data Fields

- unsigned `features`
Feature flags.
- unsigned `nr_irqs`
The number of IRQ lines supported by the ICU,.
- unsigned `nr_msis`
The number of MSI vectors supported by the ICU,.

16.244.1 Detailed Description

Info structure for an ICU.

This structure contains information about the features of an ICU.

See also

[l4_icu_info\(\)](#).

Definition at line 163 of file [icu.h](#).

16.244.2 Field Documentation

16.244.2.1 features

```
unsigned l4_icu_info_t::features
```

Feature flags.

If [L4_ICU_FLAG_MSI](#) is set the ICU supports MSIs.

Definition at line 170 of file [icu.h](#).

Referenced by [L4::Icu::Info::supports_msi\(\)](#).

The documentation for this struct was generated from the following file:

- [l4/sys/icu.h](#)

16.245 l4_icu_msi_info_t Struct Reference

Info to use for a specific MSI.

```
#include <icu.h>
```

Collaboration diagram for [l4_icu_msi_info_t](#):

<code>l4_icu_msi_info_t</code>
<ul style="list-style-type: none">+ <code>msi_addr</code>+ <code>msi_data</code>

Data Fields

- [l4_uint64_t msi_addr](#)
Value to use as address when sending this MSI.
- [l4_uint32_t msi_data](#)
Value to use as data written to msi_addr, when sending this MSI.

16.245.1 Detailed Description

Info to use for a specific MSI.

Definition at line 184 of file [icu.h](#).

The documentation for this struct was generated from the following file:

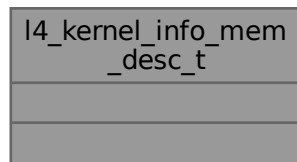
- [l4/sys/icu.h](#)

16.246 l4_kernel_info_mem_desc_t Struct Reference

Memory descriptor data structure.

```
#include <memdesc.h>
```

Collaboration diagram for l4_kernel_info_mem_desc_t:



16.246.1 Detailed Description

Memory descriptor data structure.

Note

This data type is opaque, and must be accessed by the accessor functions defined in this module.

Definition at line 73 of file [memdesc.h](#).

The documentation for this struct was generated from the following file:

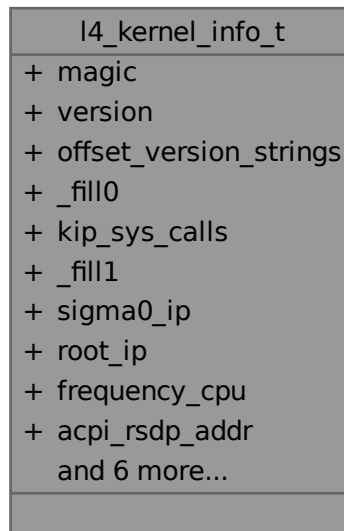
- [l4/sys/memdesc.h](#)

16.247 l4_kernel_info_t Struct Reference

[L4 Kernel Interface Page](#).

```
#include <kip.h>
```

Collaboration diagram for l4_kernel_info_t:



Data Fields

- [l4_uint32_t](#) **magic**
Kernel Info Page identifier ("L4μK").
- [l4_uint32_t](#) **version**
Kernel version.
- [l4_uint8_t](#) **offset_version_strings**
offset to version string
- [l4_uint8_t](#) **_fill0** [3]
reserved
- [l4_uint8_t](#) **kip_sys_calls**
pointer to system calls
- [l4_uint8_t](#) **_fill1** [2]
reserved
- [l4_uint64_t](#) **sigma0_ip**
Sigma0 instruction pointer.
- [l4_uint64_t](#) **root_ip**
Root task instruction pointer.
- [l4_uint64_t](#) **frequency_cpu**
CPU frequency in kHz.

- [l4_uint64_t acpi_rsdp_addr](#)
ACPI RSDP/XSDP.
- [l4_uint64_t dt_addr](#)
Device Tree.
- [l4_uint64_t user_ptr](#)
user_ptr
- [l4_uint32_t scheduler_granularity](#)
for rounding time slices
- [l4_uint32_t mem_descs](#)
memory descriptors relative to Kip
- [l4_uint32_t mem_descs_num](#)
number of memory descriptors
- [l4_uint64_t _res2 \[2\]](#)
internal - spare space

16.247.1 Detailed Description

[L4 Kernel Interface Page.](#)

32-bit architecture may assume that the upper 32 bits of addresses is 0

Definition at line 36 of file [kip.h](#).

The documentation for this struct was generated from the following file:

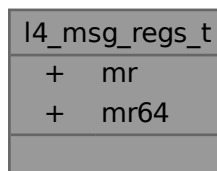
- [l4/sys/kip.h](#)

16.248 l4_msg_regs_t Union Reference

Encapsulation of the message-register block in the UTCB.

```
#include <utcb.h>
```

Collaboration diagram for l4_msg_regs_t:



Data Fields

- [l4_umword_t](#) **mr** [L4_UTCB_GENERIC_DATA_SIZE]
Message registers.
- [l4_uint64_t](#) **mr64** [L4_UTCB_GENERIC_DATA_SIZE/(sizeof([l4_uint64_t](#))/sizeof([l4_umword_t](#)))]
Message registers 64bit alias.

16.248.1 Detailed Description

Encapsulation of the message-register block in the UTCB.

Examples

[examples/sys/utcb-ipc/main.c](#).

Definition at line 67 of file [utcb.h](#).

The documentation for this union was generated from the following file:

- [l4/sys/utcb.h](#)

16.249 l4_msgtag_t Struct Reference

Message tag data structure.

```
#include <types.h>
```

Collaboration diagram for [l4_msgtag_t](#):

l4_msgtag_t
+ raw
+ label()
+ label()
+ words()
+ items()
+ flags()
+ is_page_fault()
+ is_exception()
+ is_sigma0()
+ is_io_page_fault()
+ has_error()

Public Member Functions

- long **label** () const [L4_NOTHROW](#)
Get the protocol value.
- void **label** (long v) [L4_NOTHROW](#)
Set the protocol value.
- unsigned **words** () const [L4_NOTHROW](#)
Get the number of untyped words.
- unsigned **items** () const [L4_NOTHROW](#)
Get the number of typed items.
- unsigned **flags** () const [L4_NOTHROW](#)
Get the flags value.
- bool **is_page_fault** () const [L4_NOTHROW](#)
Test if protocol indicates page-fault protocol.
- bool **is_exception** () const [L4_NOTHROW](#)
Test if protocol indicates exception protocol.
- bool **is_sigma0** () const [L4_NOTHROW](#)
Test if protocol indicates sigma0 protocol.
- bool **is_io_page_fault** () const [L4_NOTHROW](#)
Test if protocol indicates IO-page-fault protocol.
- bool **has_error** () const [L4_NOTHROW](#)
Test if flags indicate an error.

Data Fields

- [l4_mword_t](#) raw
raw value

16.249.1 Detailed Description

Message tag data structure.

Include File

```
#include <l4/sys/types.h>
```

Describes the details of an IPC operation, in particular which parts of the UTCB have to be transmitted, and also flags to enable real-time and FPU extensions.

The message tag also contains a user-defined label that could be used to specify a protocol ID. Some negative values are reserved for kernel protocols such as page faults and exceptions.

The type must be treated completely opaque.

Examples

[examples/libs/l4re/c++/shared_ds/ds_srv.cc](#), [examples/libs/l4re/streammap/server.cc](#), [examples/sys/aliens/main.c](#), [examples/sys/ipc/ipc_example.c](#), [examples/sys/singlestep/main.c](#), [examples/sys/start-with-exc/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

Definition at line 153 of file [types.h](#).

16.249.2 Member Function Documentation

16.249.2.1 flags()

```
unsigned l4_msgtag_t::flags () const [inline]
```

Get the flags value.

The flags are a combination of the flags defined by [L4_msgtag_flags](#).

Definition at line 178 of file [types.h](#).

References [L4_NOTHROW](#), and [raw](#).

16.249.2.2 has_error()

```
bool l4_msgtag_t::has_error () const [inline]
```

Test if flags indicate an error.

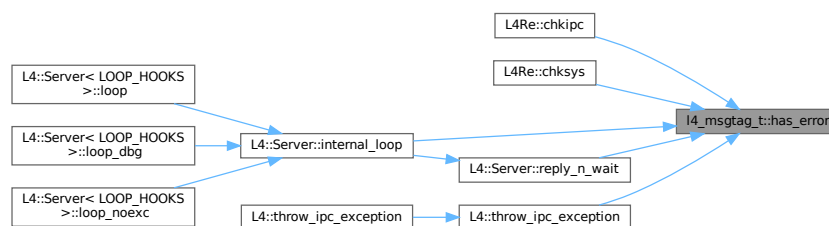
If true, the error code is stored in the UTCB, see [l4_utcb_tcr\(\)->error](#).

Definition at line 191 of file [types.h](#).

References [L4_MSGTAG_ERROR](#), [L4_NOTHROW](#), and [raw](#).

Referenced by [L4Re::chkipc\(\)](#), [L4Re::chksys\(\)](#), [L4::Server< LOOP_HOOKS >::internal_loop\(\)](#), [L4::Server< LOOP_HOOKS >::reply](#) and [L4::throw_ipc_exception\(\)](#).

Here is the caller graph for this function:



The documentation for this struct was generated from the following file:

- [l4/sys/types.h](#)

16.250 l4_sched_cpu_set_t Struct Reference

CPU sets.

```
#include <scheduler.h>
```

Collaboration diagram for l4_sched_cpu_set_t:

l4_sched_cpu_set_t
+ gran_offset
+ map
+ granularity()
+ offset()
+ set()

Public Member Functions

- unsigned char [granularity](#) () const
- unsigned [offset](#) () const
- void [set](#) (unsigned char [granularity](#), unsigned [offset](#))

Set offset and granularity.

Data Fields

- [l4_umword_t](#) [gran_offset](#)
Combination of granularity and offset.
- [l4_umword_t](#) [map](#)
Bitmap of CPUs.

16.250.1 Detailed Description

CPU sets.

Examples

[examples/sys/migrate/thread_migrate.cc](#).

Definition at line [58](#) of file [scheduler.h](#).

16.250.2 Member Function Documentation

16.250.2.1 granularity()

```
unsigned char l4_sched_cpu_set_t::granularity () const [inline]
```

Returns

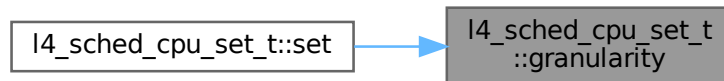
Get granularity value

Definition at line 81 of file [scheduler.h](#).

References [gran_offset](#).

Referenced by [set\(\)](#).

Here is the caller graph for this function:



16.250.2.2 offset()

```
unsigned l4_sched_cpu_set_t::offset () const [inline]
```

Returns

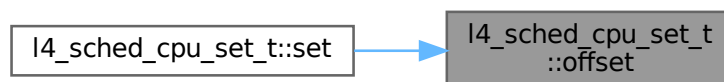
Get offset value

Definition at line 83 of file [scheduler.h](#).

References [gran_offset](#).

Referenced by [set\(\)](#).

Here is the caller graph for this function:



16.250.2.3 set()

```
void l4_sched_cpu_set_t::set (  
    unsigned char granularity,  
    unsigned offset) [inline]
```

Set offset and granularity.

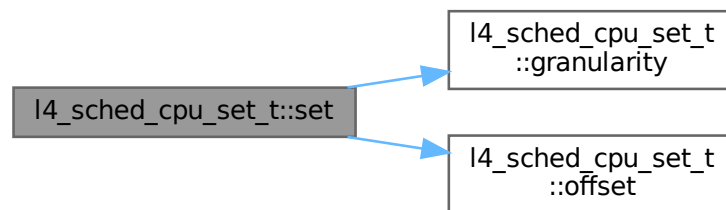
Parameters

<i>granularity</i>	Granularity in log2 notation.
<i>offset</i>	Offset. Must be a multiple of $2^{\text{granularity}}$.

Definition at line 90 of file [scheduler.h](#).

References [gran_offset](#), [granularity\(\)](#), and [offset\(\)](#).

Here is the call graph for this function:



16.250.3 Field Documentation

16.250.3.1 gran_offset

`l4_umword_t l4_sched_cpu_set_t::gran_offset`

Combination of granularity and offset.

The granularity defines how many CPUs each bit in map describes. And the offset is the number of the first CPU described by the first bit in the bitmap.

Precondition

offset must be a multiple of $2^{\text{granularity}}$.

MSB	LSB
8bit granularity	24bit offset ..

Definition at line 72 of file [scheduler.h](#).

Referenced by [granularity\(\)](#), [L4::Scheduler::info\(\)](#), [l4_sched_cpu_set\(\)](#), [offset\(\)](#), and [set\(\)](#).

The documentation for this struct was generated from the following file:

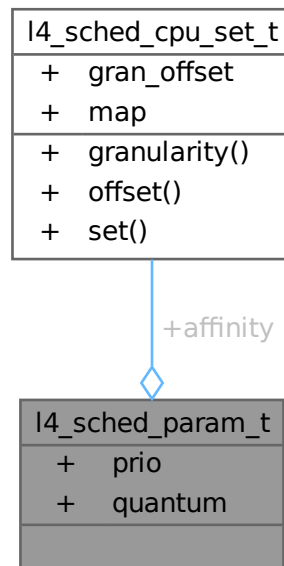
- [l4/sys/scheduler.h](#)

16.251 l4_sched_param_t Struct Reference

Scheduler parameter set.

```
#include <scheduler.h>
```

Collaboration diagram for l4_sched_param_t:



Data Fields

- [l4_sched_cpu_set_t affinity](#)
CPU affinity.
- [l4_umword_t prio](#)
Priority for scheduling.
- [l4_umword_t quantum](#)
Timeslice in micro seconds.

16.251.1 Detailed Description

Scheduler parameter set.

Examples

[examples/sys/aliens/main.c](#), [examples/sys/migrate/thread_migrate.cc](#), [examples/sys/singlestep/main.c](#), [examples/sys/start-with-exc/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

Definition at line 173 of file [scheduler.h](#).

16.251.2 Field Documentation

16.251.2.1 prio

`l4_umword_t l4_sched_param_t::prio`

Priority for scheduling.

The kernel supports priorities for userland threads in the range of 1..255. Priority 0 is reserved for the kernel.

Definition at line 182 of file [scheduler.h](#).

Referenced by [l4_sched_param\(\)](#).

The documentation for this struct was generated from the following file:

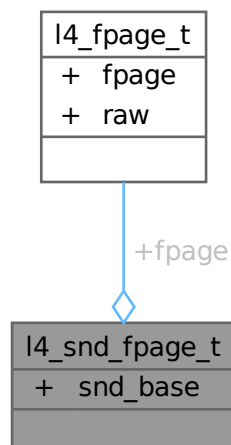
- [l4/sys/scheduler.h](#)

16.252 l4_snd_fpage_t Struct Reference

Send-flexpage types.

```
#include <__l4_fpage.h>
```

Collaboration diagram for `l4_snd_fpage_t`:



Data Fields

- `l4_umword_t snd_base`
Offset in receive window (send base).
- `l4_fpage_t fpage`
Source flexpage descriptor.

16.252.1 Detailed Description

Send-flexpage types.

Definition at line 99 of file [__l4_fpage.h](#).

The documentation for this struct was generated from the following file:

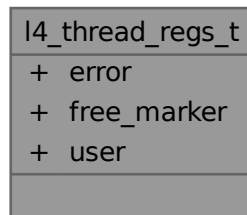
- l4/sys/__l4_fpage.h

16.253 l4_thread_regs_t Struct Reference

Encapsulation of the thread-control-register block of the UTCB.

```
#include <utcb.h>
```

Collaboration diagram for l4_thread_regs_t:



Data Fields

- [l4_umword_t error](#)
System call error code (see [l4_ipc_tcr_error_t](#)).
- [l4_umword_t free_marker](#)
Kernel free marker.
- [l4_umword_t user](#) [3]
User values (ignored and preserved by the kernel).

16.253.1 Detailed Description

Encapsulation of the thread-control-register block of the UTCB.

Definition at line 99 of file [utcb.h](#).

16.253.2 Field Documentation

16.253.2.1 error

```
l4_umword_t l4_thread_regs_t::error
```

System call error code (see [l4_ipc_tcr_error_t](#)).

If the kernel indicates an error in the message tag (see [l4_msgtag_has_error\(\)](#) and [l4_msgtag_t::has_error\(\)](#)), the kernel writes the error code to this field.

Definition at line 106 of file [utcb.h](#).

16.253.2.2 free_marker

```
l4_umword_t l4_thread_regs_t::free_marker
```

Kernel free marker.

The kernel sets this field to zero as soon as it is guaranteed that the kernel does not use the UTCB anymore for the bound thread. This usually happens while a thread is deleted. However, it is not defined when exactly the kernel sets the field. In particular, the point in time is not necessarily related to any IPC.

Userland may use this field for determining if a UTCB can be re-used for another thread. Note that, in order to make use of that feature, userland has to set this field to a non-zero value when a thread is bound with this UTCB.

Definition at line 120 of file [utcb.h](#).

The documentation for this struct was generated from the following file:

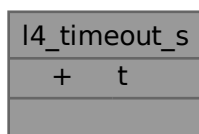
- [l4/sys/utcb.h](#)

16.254 l4_timeout_s Struct Reference

Basic timeout specification.

```
#include <__timeout.h>
```

Collaboration diagram for `l4_timeout_s`:



Data Fields

- [l4_uint16_t t](#)
timeout value

16.254.1 Detailed Description

Basic timeout specification.

If bit 15 == 0, basically a floating point number with 10 bits mantissa and 5 bits exponent ($t = m \cdot 2^e$).

If the mantissa is zero, the exponent encodes special values, see [L4_IPC_TIMEOUT_0](#) and [L4_IPC_TIMEOUT_NEVER](#).

If bit 15 == 1 the timeout is absolute and the lower 6 bits encode the index of the UTCB buffer register(s) holding the absolute 64-bit timeout value. On 32-bit systems, two consecutive UTCB buffer registers are used.

Definition at line 40 of file [__timeout.h](#).

The documentation for this struct was generated from the following file:

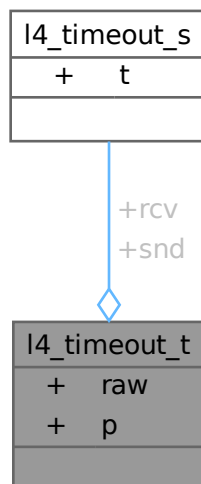
- [l4/sys/__timeout.h](#)

16.255 l4_timeout_t Union Reference

Timeout pair.

```
#include <__timeout.h>
```

Collaboration diagram for l4_timeout_t:



Data Fields

- `l4_uint32_t raw`
raw value
- struct {
 `l4_timeout_s rcv`
 receive timeout
 `l4_timeout_s snd`
 send timeout
} `p`

combined timeout

16.255.1 Detailed Description

Timeout pair.

For IPC there are usually a send and a receive timeout. So this structure contains a pair of timeouts.

Definition at line 52 of file `__timeout.h`.

The documentation for this union was generated from the following file:

- `l4/sys/__timeout.h`

16.256 l4_vcon_attr_t Struct Reference

Vcon attribute structure.

```
#include <vcon.h>
```

Collaboration diagram for `l4_vcon_attr_t`:

<code>l4_vcon_attr_t</code>
+ <code>i_flags</code>
+ <code>o_flags</code>
+ <code>l_flags</code>
+ <code>set_raw()</code>

Public Member Functions

- void `set_raw()`
Set terminal attributes to disable all special processing.

Data Fields

- [l4_umword_t i_flags](#)
input flags
- [l4_umword_t o_flags](#)
output flags
- [l4_umword_t l_flags](#)
local flags

16.256.1 Detailed Description

Vcon attribute structure.

The flags members can be a combination of their respective enums.

See also

[L4_vcon_i_flags](#)
[L4_vcon_o_flags](#)
[L4_vcon_l_flags](#)

Examples

[examples/sys/isr/main.c](#).

Definition at line 187 of file [vcon.h](#).

16.256.2 Member Function Documentation

16.256.2.1 set_raw()

```
void l4_vcon_attr_t::set_raw () [inline]
```

Set terminal attributes to disable all special processing.

Removes all flags that would mangle the read or written characters. Also disables echoing and any special processing of characters.

Definition at line 450 of file [vcon.h](#).

References [l4_vcon_set_attr_raw\(\)](#).

Here is the call graph for this function:



The documentation for this struct was generated from the following file:

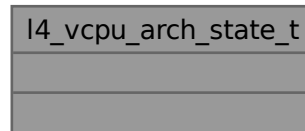
- [l4/sys/vcon.h](#)

16.257 l4_vcpu_arch_state_t Struct Reference

Architecture-specific vCPU state.

```
#include <__vcpu-arch.h>
```

Collaboration diagram for l4_vcpu_arch_state_t:



16.257.1 Detailed Description

Architecture-specific vCPU state.

Definition at line 74 of file [__vcpu-arch.h](#).

The documentation for this struct was generated from the following files:

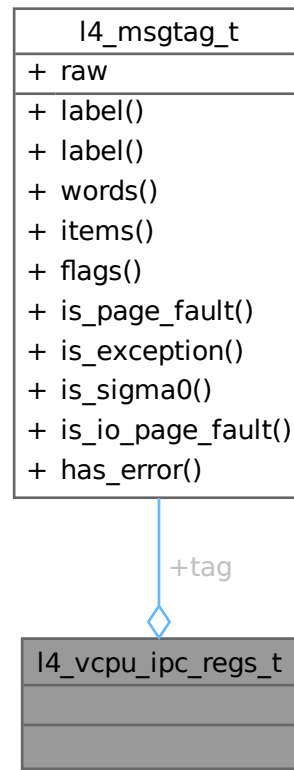
- [arm/l4/sys/__vcpu-arch.h](#)
- [arm64/l4/sys/__vcpu-arch.h](#)
- [amd64/l4/sys/__vcpu-arch.h](#)

16.258 l4_vcpu_ipc_regs_t Struct Reference

vCPU message registers.

```
#include <__vcpu-arch.h>
```


Collaboration diagram for l4_vcpu_ipc_regs_t:



16.258.1 Detailed Description

vCPU message registers.

Definition at line 83 of file [__vcpu-arch.h](#).

The documentation for this struct was generated from the following files:

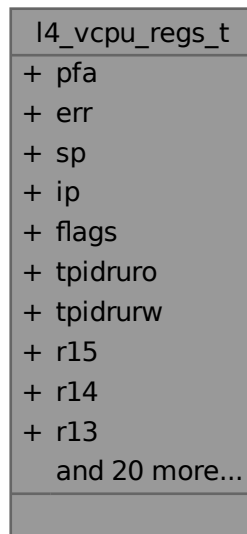
- [arm/l4/sys/__vcpu-arch.h](#)
- [arm64/l4/sys/__vcpu-arch.h](#)
- [amd64/l4/sys/__vcpu-arch.h](#)
- [x86/l4/sys/__vcpu-arch.h](#)

16.259 l4_vcpu_regs_t Struct Reference

vCPU registers.

```
#include <__vcpu-arch.h>
```

Collaboration diagram for `l4_vcpu_regs_t`:



Data Fields

- `l4_umword_t pfa`
page fault address
- `l4_umword_t err`
error code
- `l4_umword_t sp`
stack pointer
- `l4_umword_t ip`
instruction pointer
- `l4_umword_t flags`
eflags
- `l4_umword_t tpidruro`
Thread-ID register.
- `l4_umword_t tpidrurw`
Thread-ID register.
- `l4_umword_t r15`
r15 register
- `l4_umword_t r14`
r14 register
- `l4_umword_t r13`
r13 register
- `l4_umword_t r12`
r12 register
- `l4_umword_t r11`
r11 register

- [l4_umword_t r10](#)
r10 register
- [l4_umword_t r9](#)
r9 register
- [l4_umword_t r8](#)
r8 register
- [l4_umword_t di](#)
rdi register
- [l4_umword_t si](#)
rsi register
- [l4_umword_t bp](#)
rbp register
- [l4_umword_t bx](#)
rbx register
- [l4_umword_t dx](#)
rdx register
- [l4_umword_t cx](#)
rcx register
- [l4_umword_t ax](#)
rax register
- [l4_umword_t trapno](#)
trap number
- [l4_umword_t cs](#)
dummy
- [l4_umword_t ss](#)
ss register
- [l4_umword_t es](#)
es register
- [l4_umword_t ds](#)
ds register
- [l4_umword_t gs](#)
gs register
- [l4_umword_t fs](#)
fs register
- [l4_umword_t dummy1](#)
dummy

16.259.1 Detailed Description

vCPU registers.

Definition at line 55 of file [__vcpu-arch.h](#).

16.259.2 Field Documentation

16.259.2.1 `ax`

[l4_umword_t](#) `l4_vcpu_regs_t::ax`

rax register

eax register

Definition at line 77 of file [__vcpu-arch.h](#).

16.259.2.2 bp

`l4_umword_t l4_vcpu_regs_t::bp`

rbp register

ebp register

Definition at line 72 of file [__vcpu-arch.h](#).

16.259.2.3 bx

`l4_umword_t l4_vcpu_regs_t::bx`

rbx register

ebx register

Definition at line 74 of file [__vcpu-arch.h](#).

16.259.2.4 cx

`l4_umword_t l4_vcpu_regs_t::cx`

rcx register

ecx register

Definition at line 76 of file [__vcpu-arch.h](#).

16.259.2.5 di

`l4_umword_t l4_vcpu_regs_t::di`

rdi register

edi register

Definition at line 70 of file [__vcpu-arch.h](#).

16.259.2.6 dx

`l4_umword_t l4_vcpu_regs_t::dx`

rdx register

edx register

Definition at line 75 of file [__vcpu-arch.h](#).

16.259.2.7 si

`l4_umword_t l4_vcpu_regs_t::si`

rsi register

esi register

Definition at line 71 of file `__vcpu-arch.h`.

The documentation for this struct was generated from the following files:

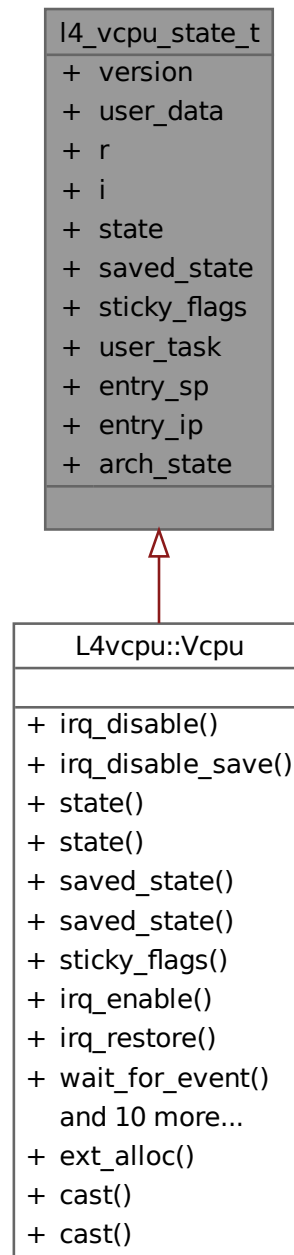
- `arm/l4/sys/__vcpu-arch.h`
- `amd64/l4/sys/__vcpu-arch.h`
- `x86/l4/sys/__vcpu-arch.h`

16.260 l4_vcpu_state_t Struct Reference

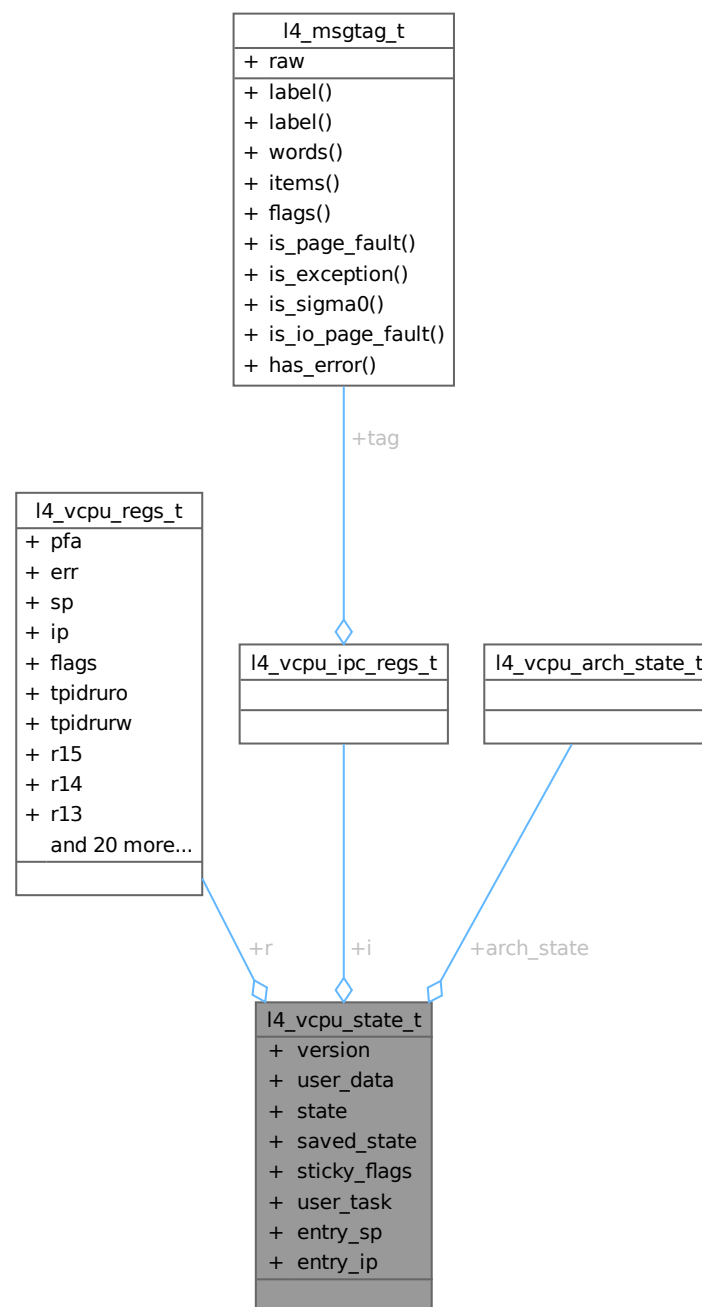
State of a vCPU.

```
#include <vcpu.h>
```

Inheritance diagram for l4_vcpu_state_t:



Collaboration diagram for l4_vcpu_state_t:



Data Fields

- [l4_umword_t version](#)
vCPU ABI version.
- [l4_umword_t user_data](#) [7]
User-specific data.
- [l4_vcpu_regs_t r](#)

- Register state.*
 - [l4_vcpu_ipc_regs_t](#) **i**
- IPC state.*
 - [l4_uint16_t](#) **state**
- Current vCPU state. See [L4_vcpu_state_flags](#).*
 - [l4_uint16_t](#) **saved_state**
- Saved vCPU state. See [L4_vcpu_state_flags](#).*
 - [l4_uint16_t](#) **sticky_flags**
- Pending flags. See [L4_vcpu_sticky_flags](#).*
 - [l4_cap_idx_t](#) **user_task**
- User task to use.*
 - [l4_umword_t](#) **entry_sp**
- Stack pointer for entry (when coming from user task).*
 - [l4_umword_t](#) **entry_ip**
- IP for entry.*
 - [l4_vcpu_arch_state_t](#) **arch_state**
- Architecture-specific state.*

16.260.1 Detailed Description

State of a vCPU.

Definition at line 75 of file [vcpu.h](#).

16.260.2 Field Documentation

16.260.2.1 version

```
l4\_umword\_t l4_vcpu_state_t::version
```

vCPU ABI version.

Set by the kernel and must be checked by the user for equality with [L4_VCPU_STATE_VERSION](#).

Definition at line 77 of file [vcpu.h](#).

The documentation for this struct was generated from the following file:

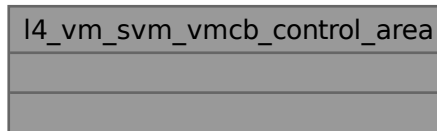
- [l4/sys/vcpu.h](#)

16.261 l4_vm_svm_vmcb_control_area Struct Reference

VMCB structure for SVM VMs.

```
#include <__vm-svm.h>
```

Collaboration diagram for l4_vm_svm_vmcb_control_area:



16.261.1 Detailed Description

VMCB structure for SVM VMs.

Definition at line 28 of file [__vm-svm.h](#).

The documentation for this struct was generated from the following file:

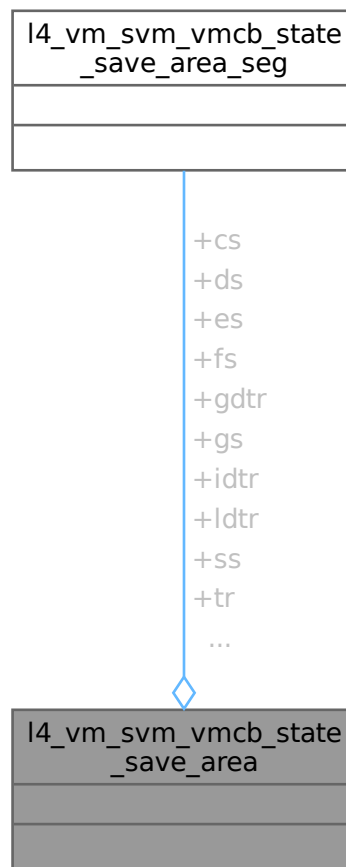
- l4/sys/__vm-svm.h

16.262 l4_vm_svm_vmcb_state_save_area Struct Reference

State save area structure for SVM VMs.

```
#include <__vm-svm.h>
```

Collaboration diagram for `l4_vm_svm_vmcb_state_save_area`:



16.262.1 Detailed Description

State save area structure for SVM VMs.

Definition at line 85 of file [__vm-svm.h](#).

The documentation for this struct was generated from the following file:

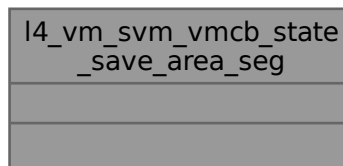
- `l4/sys/__vm-svm.h`

16.263 `l4_vm_svm_vmcb_state_save_area_seg` Struct Reference

State save area segment selector struct.

```
#include <__vm-svm.h>
```

Collaboration diagram for l4_vm_svm_vmcb_state_save_area_seg:



16.263.1 Detailed Description

State save area segment selector struct.

Definition at line 73 of file [__vm-svm.h](#).

The documentation for this struct was generated from the following file:

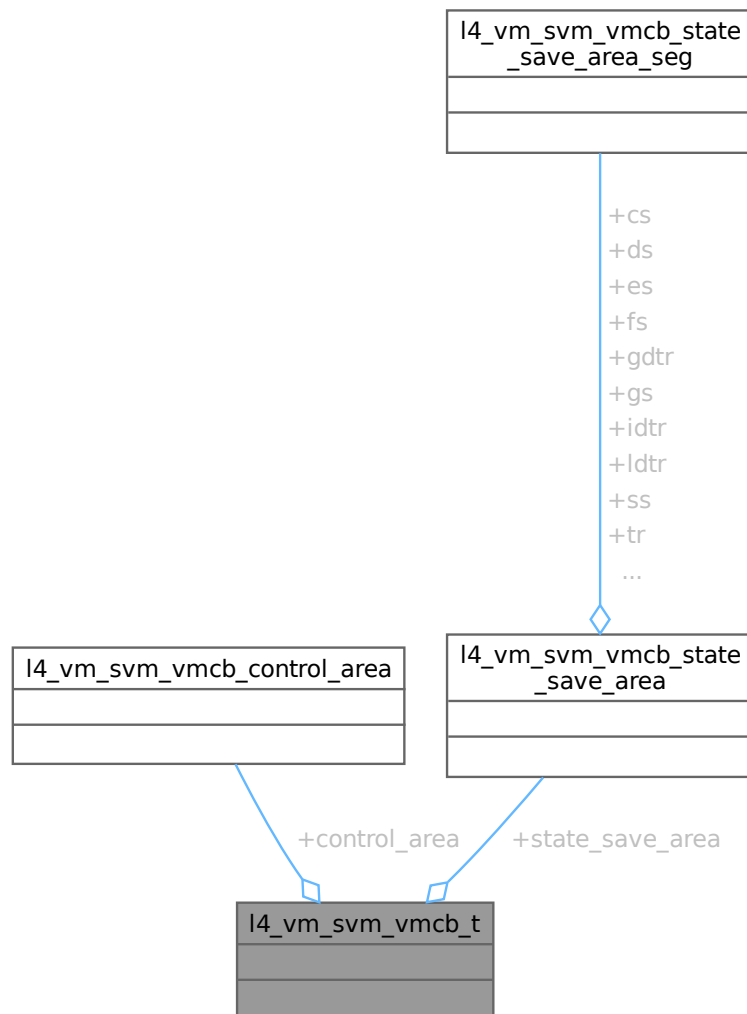
- l4/sys/__vm-svm.h

16.264 l4_vm_svm_vmcb_t Struct Reference

Control structure for SVM VMs.

```
#include <__vm-svm.h>
```

Collaboration diagram for `l4_vm_svm_vmcb_t`:



16.264.1 Detailed Description

Control structure for SVM VMs.

Definition at line 154 of file [__vm-svm.h](#).

The documentation for this struct was generated from the following file:

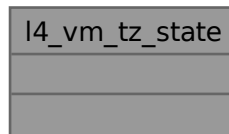
- `l4/sys/__vm-svm.h`

16.265 l4_vm_tz_state Struct Reference

state structure for TrustZone VMs

```
#include <vm.h>
```

Collaboration diagram for l4_vm_tz_state:



16.265.1 Detailed Description

state structure for TrustZone VMs

Definition at line 41 of file [vm.h](#).

The documentation for this struct was generated from the following file:

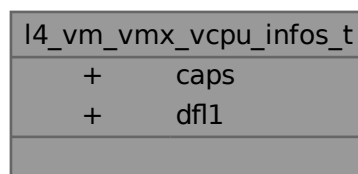
- [arm/l4/sys/vm.h](#)

16.266 l4_vm_vmx_vcpu_infos_t Struct Reference

VMX information members.

```
#include <__vm-vmx.h>
```

Collaboration diagram for l4_vm_vmx_vcpu_infos_t:



Data Fields

- [l4_uint64_t caps](#) [[L4_VM_VMX_NUM_CAPS_REGS](#)]
Exported VMX capability registers. See [L4_vm_vmx_caps_regs](#).
- [l4_uint32_t dfl1](#) [[L4_VM_VMX_NUM_DFL1_REGS](#)]
Exported VMX capability registers (default to 1 bits).

16.266.1 Detailed Description

VMX information members.

Definition at line [239](#) of file [__vm-vmx.h](#).

16.266.2 Field Documentation

16.266.2.1 dfl1

```
l4\_uint32\_t l4_vm_vmx_vcpu_infos_t::dfl1 [L4\_VM\_VMX\_NUM\_DFL1\_REGS]
```

Exported VMX capability registers (default to 1 bits).

See [L4_vm_vmx_dfl1_regs](#).

Definition at line [246](#) of file [__vm-vmx.h](#).

The documentation for this struct was generated from the following file:

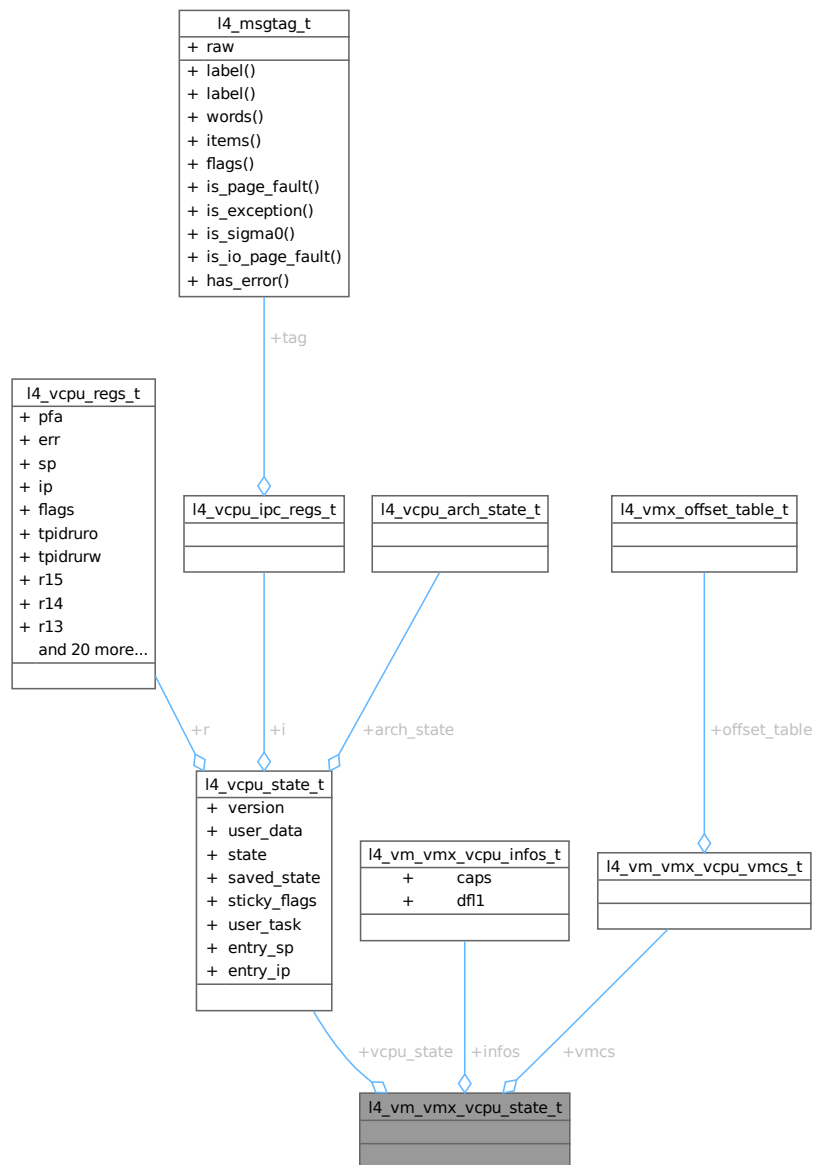
- [l4/sys/__vm-vmx.h](#)

16.267 l4_vm_vmx_vcpu_state_t Struct Reference

VMX vCPU state.

```
#include <\_\_vm-vmx.h>
```

Collaboration diagram for l4_vm_vmx_vcpu_state_t:



16.267.1 Detailed Description

VMX vCPU state.

This is a specialization of the generic vCPU state for VMX. This data structure represents the following memory layout:

- 0x000 - 0x1ff: Standard vCPU state (with padding). See [l4_vcpu_state_t](#).
- 0x200 - 0x3ff: VMX information members (with padding). See [l4_vm_vmx_vcpu_infos_t](#).
- 0x400 - 0xffff: VMX software VMCS. See [l4_vm_vmx_vcpu_vmcs_t](#).

Note

The memory layout is documented here for reference purposes. However, the users are strongly discouraged from accessing the data structure directly. The API functions defined in this file are the preferred way of achieving the functionality.

Definition at line 267 of file [__vm-vmx.h](#).

The documentation for this struct was generated from the following file:

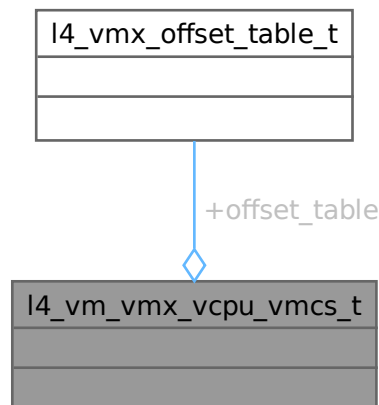
- l4/sys/__vm-vmx.h

16.268 l4_vm_vmx_vcpu_vmcs_t Struct Reference

VMX software VMCS.

```
#include <__vm-vmx.h>
```

Collaboration diagram for l4_vm_vmx_vcpu_vmcs_t:



16.268.1 Detailed Description

VMX software VMCS.

This data structure represents the following memory layout:

- 0x000 - 0x007: Reserved (ignored by the kernel). In the hardware VMCS, the revision identifier and the abort indicator are stored in this area. Hereby we simply ignore these two entries.
- 0x008 - 0x00f: User space data (ignored by the kernel). This currently stores the pointer to a different software VMCS whose content has been loaded to this software VMCS.

- 0x010 - 0x013: VMCS field index of the software-defined CR2 field in the software VMCS.
- 0x014 - 0x017: Reserved.
- 0x018 - 0x01f: Capability of the vCPU context, i.e. the hardware VMCS object (with padding).
- 0x020 - 0x047: Software VMCS field offset table. See [l4_vmx_offset_table_t](#).
- 0x048 - 0x0bf: Reserved.
- 0x0c0 - 0xabf: Software VMCS fields (with padding).
- 0xac0 - 0xbff: Software VMCS fields dirty bitmap (with padding).

Note

The memory layout is documented here for reference purposes. However, the users are strongly discouraged from accessing the data structure directly. The API functions defined in this file are the preferred way of achieving the functionality.

Definition at line 205 of file [__vm-vmx.h](#).

The documentation for this struct was generated from the following file:

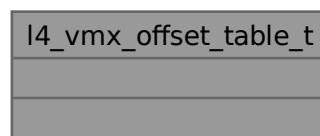
- l4/sys/__vm-vmx.h

16.269 l4_vmx_offset_table_t Struct Reference

Software VMCS field offset table.

```
#include <__vm-vmx.h>
```

Collaboration diagram for l4_vmx_offset_table_t:



16.269.1 Detailed Description

Software VMCS field offset table.

This data structure represents the following memory layout:

- 0x00 - 0x02: 3 offsets for 16-bit fields.
- 0x03: Reserved.
- 0x04 - 0x06: 3 offsets for 64-bit fields.
- 0x07: Reserved.
- 0x08 - 0x0a: 3 offsets for 32-bit fields.
- 0x0b: Reserved.
- 0x0c - 0x0e: 3 offsets for natural-width fields.
- 0x0f: Reserved.
- 0x10 - 0x12: 3 limits for 16-bit fields.
- 0x13: Reserved.
- 0x14 - 0x16: 3 limits for 64-bit fields.
- 0x17: Reserved.
- 0x18 - 0x1a: 3 limits for 32-bit fields.
- 0x1b: Reserved.
- 0x1c - 0x1e: 3 limits for natural-width fields.
- 0x1f: Reserved.
- 0x20 - 0x23: 4 index shifts.
- 0x24: Offset of the first software VMCS field.
- 0x25: Size of the software VMCS fields.
- 0x26 - 0x27: Reserved.

The offsets/limits in each size category are in the following order:

- Control fields.
- Read-only fields.
- Guest fields.

The index shifts are in the following order:

- 16-bit.
- 64-bit.
- 32-bit.
- Natural-width.

All offsets/limits/sizes are represented in a 64-byte granule.

The offsets (after being multiplied by 64) are indexes in the values array in [l4_vm_vmx_vcpu_vmcs_t](#) and bit indexes in the dirty_bitmap array in [l4_vm_vmx_vcpu_vmcs_t](#).

The limits (after being multiplied by 64) represent the range of the available indexes.

Note

The memory layout is documented here for reference purposes. However, the users are strongly discouraged from accessing the data structure directly. The API functions defined in this file are the preferred way of achieving the functionality.

Definition at line 155 of file [__vm-vmx.h](#).

The documentation for this struct was generated from the following file:

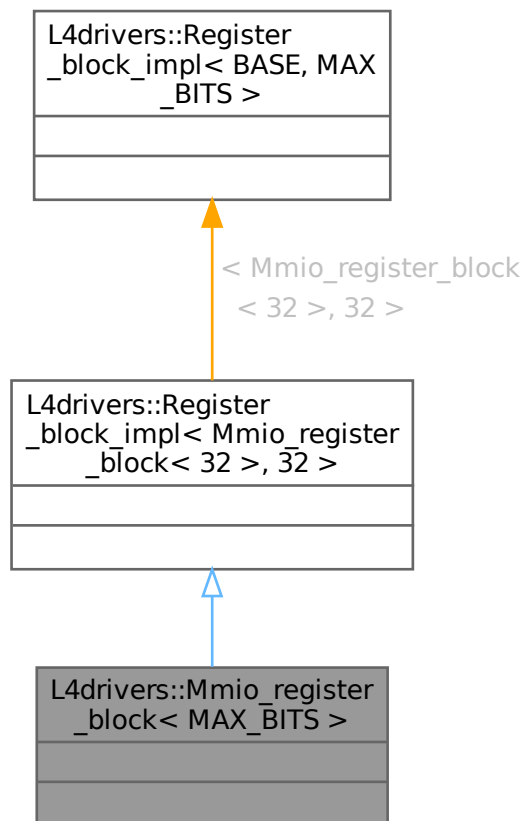
- l4/sys/__vm-vmx.h

16.270 L4drivers::Mmio_register_block< MAX_BITS > Struct Template Reference

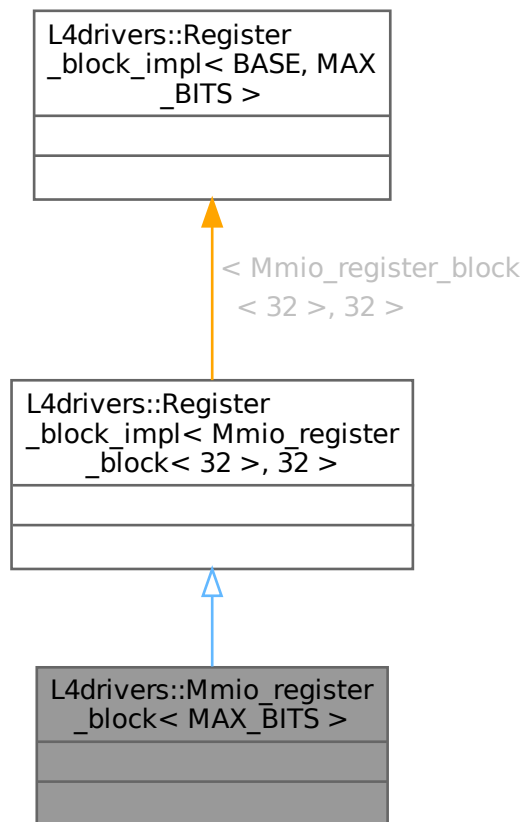
An MMIO block with up to 64-bit register access (32-bit default) and little endian byte order.

```
#include <hw_mmio_register_block>
```

Inheritance diagram for L4drivers::Mmio_register_block< MAX_BITS >:



Collaboration diagram for L4drivers::Mmio_register_block< MAX_BITS >:



16.270.1 Detailed Description

```
template<unsigned MAX_BITS = 32>
struct L4drivers::Mmio_register_block< MAX_BITS >
```

An MMIO block with up to 64-bit register access (32-bit default) and little endian byte order.

Definition at line 43 of file [hw_mmio_register_block](#).

The documentation for this struct was generated from the following file:

- `pkg/drivers-frst/include/hw_mmio_register_block`

16.271 L4drivers::Register_block< MAX_BITS, BLOCK > Class Template Reference

Handles a reference to a register block of the given maximum access width.

```
#include <hw_register_block>
```

Collaboration diagram for L4drivers::Register_block< MAX_BITS, BLOCK >:

L4drivers::Register_block< MAX_BITS, BLOCK >	
+	r()
+	operator[]()
+	r()
+	operator[]()

Public Member Functions

- template<unsigned BITS>
[Ro_register_tmpl](#)< BITS, Block > [r](#) (unsigned offset) const
Read only access to register at offset offset.
- [Ro_register_operator](#)[] (unsigned offset) const
Read only access to register at offset offset.
- template<unsigned BITS>
[Register_tmpl](#)< BITS, Block > [r](#) (unsigned offset)
Read/write access to register at offset offset.
- [Register_operator](#)[] (unsigned offset)
Read/write access to register at offset offset.

16.271.1 Detailed Description

```
template<unsigned MAX_BITS, typename BLOCK = Register_block_tmpl< Register_block_base<MAX_BITS> >>
class L4drivers::Register_block< MAX_BITS, BLOCK >
```

Handles a reference to a register block of the given maximum access width.

Register block.

Template Parameters

MAX_BITS	Maximum access width for the registers in this block.
BLOCK	Type implementing the register accesses (<code>read<>()</code> , <code>write<>()</code> , <code>modify<>()</code> , <code>set<>()</code> , and <code>clear<>()</code>).

Provides access to registers in this block via `r<WIDTH>()` and `operator[]()`.

Example usage:

```
void test()
{
    // create a register block reference for max. 16bit accesses, using a
    // MMIO register block implementation (at address 0x1000).
    Hw::Register_block<16> regs = new Hw::Mmio_register_block<16>(0x1000);

    // Alternatively it is allowed to use an implementation that allows
    // wider access than actually needed.
    Hw::Register_block<16> regs = new Hw::Mmio_register_block<32>(0x1000);

    // read a 16bit register at offset 8byte
    unsigned short x = regs.r<16>(8);
    unsigned short x1 = regs[8]; // alternative

    // read an 8bit register at offset 0byte
    unsigned v = regs.r<8>(0);

    // do a 16bit write to register at offset 2byte (four variants)
    regs[2] = 22;
    regs.r<16>(2) = 22;
    regs[2].write(22);
    regs.r<16>().write(22);

    // do an 8bit write (two variants)
    regs.r<8>(0) = 9;
    regs.r<8>(0).write(9);

    // do 16bit read-modify-write (two variants)
    regs[4].modify(0xf, 3); // clear 4 lowest bits and set them to 3
    regs.r<16>(4).modify(0xf, 3);

    // do 8bit read-modify-write
    regs.r<8>(0).modify(0xf, 3);

    // fails to compile, because of too wide access
    // (32 bit access but regs is Hw::Register_block<16>)
    unsigned long v = regs.r<32>(4);
}
```

Definition at line 330 of file `hw_register_block`.

16.271.2 Member Function Documentation

16.271.2.1 `operator[]()` [1/2]

```
template<unsigned MAX_BITS, typename BLOCK = Register_block_tmpl< Register_block_base<MAX_↵
BITS> >>
Register L4drivers::Register_block< MAX_BITS, BLOCK >::operator[] (
    unsigned offset) [inline]
```

Read/write access to register at offset *offset*.

Parameters

<i>offset</i>	The offset of the register within the register file.
---------------	--

Returns

register object allowing read and write access with width *MAX_BITS*.

Definition at line 385 of file [hw_register_block](#).

References [r\(\)](#).

Here is the call graph for this function:

**16.271.2.2 operator[]() [2/2]**

```

template<unsigned MAX_BITS, typename BLOCK = Register_block_tmpl< Register_block_base<MAX_BITS> >>
Ro_register L4drivers::Register_block< MAX_BITS, BLOCK >::operator[] (
    unsigned offset) const [inline]
  
```

Read only access to register at offset *offset*.

Parameters

<i>offset</i>	The offset of the register within the register file.
---------------	--

Returns

register object allowing read only access with width *MAX_BITS*.

Definition at line 365 of file [hw_register_block](#).

References [r\(\)](#).

Here is the call graph for this function:



16.271.2.3 `r()` [1/2]

```
template<unsigned MAX_BITS, typename BLOCK = Register_block_tmpl< Register_block_base<MAX_BITS> >>
template<unsigned BITS>
Register_tmpl< BITS, Block > L4drivers::Register_block< MAX_BITS, BLOCK >::r (
    unsigned offset) [inline]
```

Read/write access to register at offset *offset*.

Template Parameters

<i>BITS</i>	the access width in bits for the register.
-------------	--

Parameters

<i>offset</i>	The offset of the register within the register file.
---------------	--

Returns

register object allowing read and write access with width *BITS*.

Definition at line 376 of file [hw_register_block](#).

16.271.2.4 `r()` [2/2]

```
template<unsigned MAX_BITS, typename BLOCK = Register_block_tmpl< Register_block_base<MAX_BITS> >>
template<unsigned BITS>
Ro_register_tmpl< BITS, Block > L4drivers::Register_block< MAX_BITS, BLOCK >::r (
    unsigned offset) const [inline]
```

Read only access to register at offset *offset*.

Template Parameters

<i>BITS</i>	the access width in bits for the register.
-------------	--

Parameters

<i>offset</i>	The offset of the register within the register file.
---------------	--

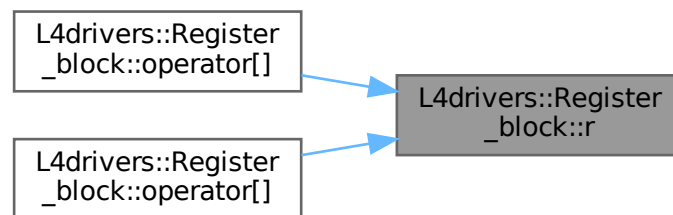
Returns

register object allowing read only access with width *BITS*.

Definition at line 357 of file [hw_register_block](#).

Referenced by [operator\[\]\(\)](#), and [operator\[\]\(\)](#).

Here is the caller graph for this function:



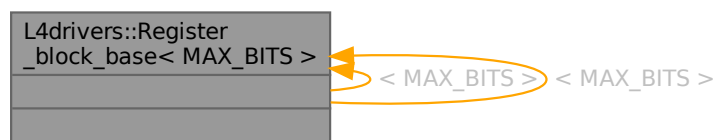
The documentation for this class was generated from the following file:

- `pkg/drivers-frst/include/hw_register_block`

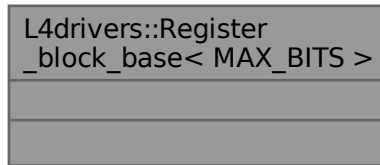
16.272 L4drivers::Register_block_base< MAX_BITS > Struct Template Reference

Abstract register block interface.

Inheritance diagram for `L4drivers::Register_block_base< MAX_BITS >`:



Collaboration diagram for L4drivers::Register_block_base< MAX_BITS >:



16.272.1 Detailed Description

```
template<unsigned MAX_BITS = 32>
struct L4drivers::Register_block_base< MAX_BITS >
```

Abstract register block interface.

Template Parameters

<i>MAX_BITS</i>	The maximum access width for the registers.
-----------------	---

This interfaces is based on virtual `do_read_<xx>` and `do_write_<xx>` methods that have to be implemented up to the maximum access width.

Definition at line 72 of file [hw_register_block](#).

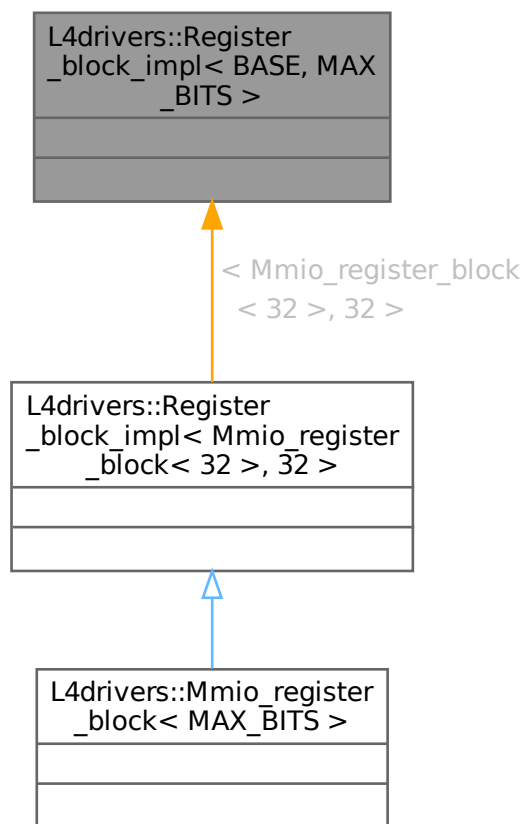
The documentation for this struct was generated from the following file:

- `pkg/drivers-frst/include/hw_register_block`

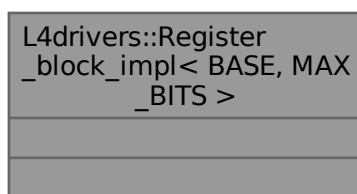
16.273 L4drivers::Register_block_impl< BASE, MAX_BITS > Struct Template Reference

Implementation helper for register blocks.

Inheritance diagram for L4drivers::Register_block_impl< BASE, MAX_BITS >:



Collaboration diagram for L4drivers::Register_block_impl< BASE, MAX_BITS >:



16.273.1 Detailed Description

```
template<typename BASE, unsigned MAX_BITS = 32>
struct L4drivers::Register_block_impl< BASE, MAX_BITS >
```

Implementation helper for register blocks.

Parameters

<i>BASE</i>	The class implementing read<> and write<> template functions for accessing the registers. This class must inherit from Register_block_impl .
<i>MAX_BITS</i>	The maximum access width for the register file. Supported values are 8, 16, 32, or 64.

This template allows easy implementation of register files by providing read<> and write<> template functions, see [Mmio_register_block](#) as an example.

Definition at line 455 of file [hw_register_block](#).

The documentation for this struct was generated from the following file:

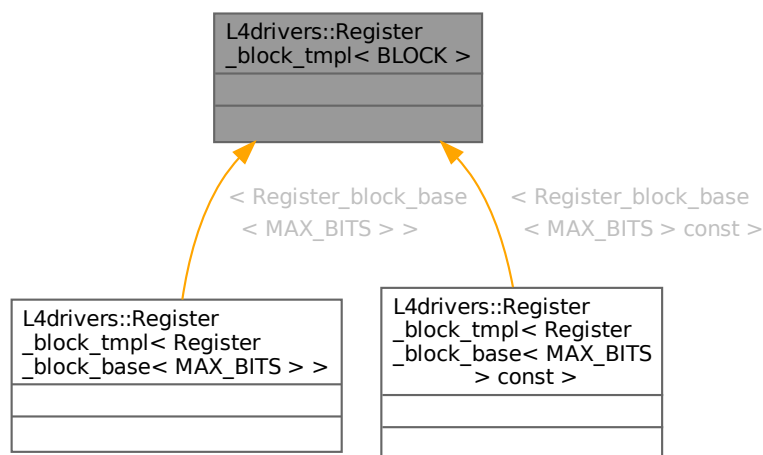
- [pkg/drivers-frst/include/hw_register_block](#)

16.274 L4drivers::Register_block_tmpl< BLOCK > Class Template Reference

Helper template that translates to the [Register_block_base](#) interface.

```
#include <hw_register_block>
```

Inheritance diagram for L4drivers::Register_block_tmpl< BLOCK >:



Collaboration diagram for L4drivers::Register_block_tmpl< BLOCK >:



16.274.1 Detailed Description

```
template<typename BLOCK>
class L4drivers::Register_block_tmpl< BLOCK >
```

Helper template that translates to the [Register_block_base](#) interface.

Template Parameters

<i>BLOCK</i>	The type of the Register_block_base interface to use.
--------------	---

This helper translates `read<T>()`, `write<T>()`, `set<T>()`, `clear<T>()`, and `modify<T>()` calls to `BLOCK::do_read_<xx>` and `BLOCK::do_write_<xx>`.

Definition at line 156 of file [hw_register_block](#).

The documentation for this class was generated from the following file:

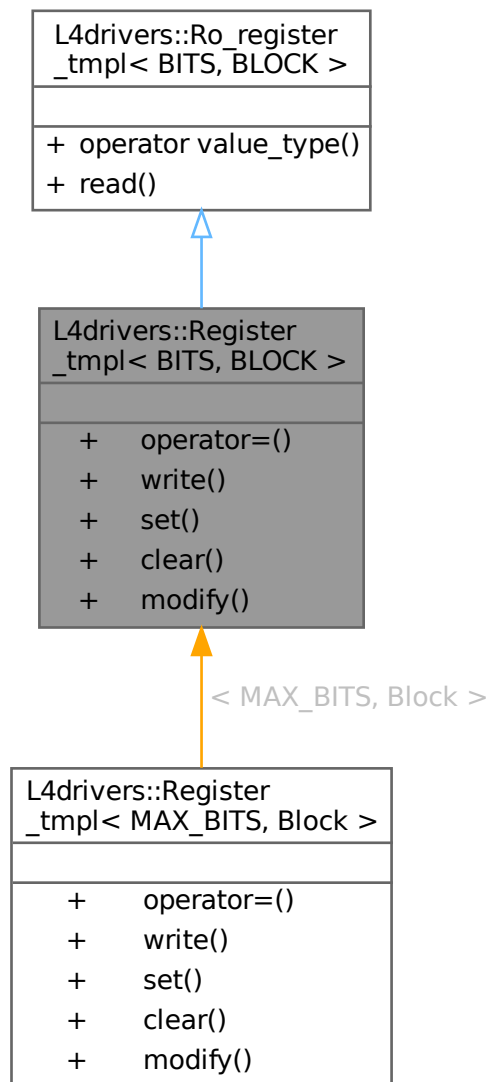
- `pkg/drivers-frst/include/hw_register_block`

16.275 L4drivers::Register_tmpl< BITS, BLOCK > Class Template Reference

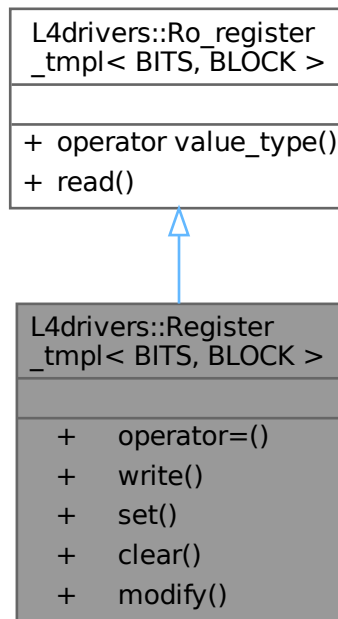
Single hardware register inside a [Register_block_base](#) interface.

```
#include <hw_register_block>
```

Inheritance diagram for L4drivers::Register_tmpl< BITS, BLOCK >:



Collaboration diagram for L4drivers::Register_tmpl< BITS, BLOCK >:



Public Member Functions

- `Register_tmpl & operator= (value_type val)`
write val into the hardware register.
- `void write (value_type val)`
write val into the hardware register.
- `value_type set (value_type set_bits)`
set bits in set_bits in the hardware register.
- `value_type clear (value_type clear_bits)`
clears bits in clear_bits in the hardware register.
- `value_type modify (value_type clear_bits, value_type set_bits)`
clears bits in clear_bits and sets bits in set_bits in the hardware register.

Public Member Functions inherited from `L4drivers::Ro_register_tmpl< BITS, BLOCK >`

- `operator value_type () const`
read the value from the hardware register.
- `value_type read () const`
read the value from the hardware register.

16.275.1 Detailed Description

```
template<unsigned BITS, typename BLOCK>  
class L4drivers::Register_tmpl< BITS, BLOCK >
```

Single hardware register inside a [Register_block_base](#) interface.

Template Parameters

<i>BITS</i>	The access width for the register in bits.
<i>BLOCK</i>	the type of the Register_block_base interface.

Note

Objects of this type must be used only in temporary contexts not in global, class, or object scope.

Definition at line 237 of file [hw_register_block](#).

16.275.2 Member Function Documentation**16.275.2.1 clear()**

```
template<unsigned BITS, typename BLOCK>
value_type L4drivers::Register_tmpl< BITS, BLOCK >::clear (
    value_type clear_bits) [inline]
```

clears bits in *clear_bits* in the hardware register.

Parameters

<i>clear_bits</i>	bits to be cleared within the hardware register.
-------------------	--

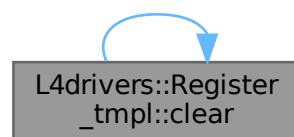
This is a read-modify-write function that does a logical and of the old value from the register with the negated value of *clear_bits*.

```
unsigned old_value = read();
write(old_value & ~clear_bits);
```

Definition at line 290 of file [hw_register_block](#).

Referenced by [L4drivers::Register_tmpl< MAX_BITS, Block >::clear\(\)](#).

Here is the caller graph for this function:

**16.275.2.2 modify()**

```
template<unsigned BITS, typename BLOCK>
value_type L4drivers::Register_tmpl< BITS, BLOCK >::modify (
    value_type clear_bits,
    value_type set_bits) [inline]
```

clears bits in *clear_bits* and sets bits in *set_bits* in the hardware register.

Parameters

<i>clear_bits</i>	bits to be cleared within the hardware register.
<i>set_bits</i>	bits to set in the hardware register.

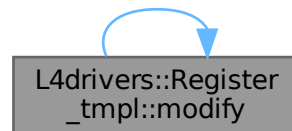
This is a read-modify-write function that first does a logical and of the old value from the register with the negated value of *clear_bits* and then does a logical or with *set_bits*.

```
unsigned old_value = read();
write((old_value & ~clear_bits) | set_bits);
```

Definition at line 308 of file [hw_register_block](#).

Referenced by [L4drivers::Register_tmpl< MAX_BITS, Block >::modify\(\)](#).

Here is the caller graph for this function:



16.275.2.3 operator=()

```
template<unsigned BITS, typename BLOCK>
Register_tmpl & L4drivers::Register_tmpl< BITS, BLOCK >::operator= (
    value_type val) [inline]
```

write *val* into the hardware register.

Parameters

<i>val</i>	the value to write into the hardware register.
------------	--

Definition at line 252 of file [hw_register_block](#).

16.275.2.4 set()

```
template<unsigned BITS, typename BLOCK>
value_type L4drivers::Register_tmpl< BITS, BLOCK >::set (
    value_type set_bits) [inline]
```

set bits in *set_bits* in the hardware register.

Parameters

<i>set_bits</i>	bits to be set within the hardware register.
-----------------	--

This is a read-modify-write function that does a logical or of the old value from the register with *set_bits*.

```
unsigned old_value = read();
write(old_value | set_bits);
```

Definition at line 274 of file [hw_register_block](#).

Referenced by [L4drivers::Register_tmpl< MAX_BITS, Block >::set\(\)](#).

Here is the caller graph for this function:



16.275.2.5 write()

```
template<unsigned BITS, typename BLOCK>
void L4drivers::Register_tmpl< BITS, BLOCK >::write (
    value_type val) [inline]
```

write *val* into the hardware register.

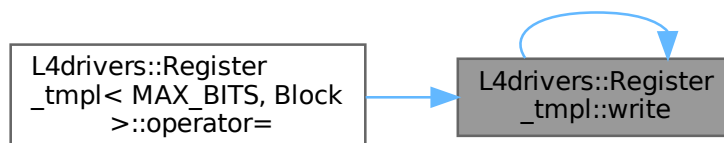
Parameters

<i>val</i>	the value to write into the hardware register.
------------	--

Definition at line 259 of file [hw_register_block](#).

Referenced by [L4drivers::Register_tmpl< MAX_BITS, Block >::operator=\(\)](#), and [L4drivers::Register_tmpl< MAX_BITS, Block >::write\(\)](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

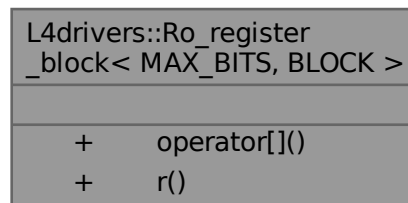
- [pkg/drivers-frst/include/hw_register_block](#)

16.276 L4drivers::Ro_register_block< MAX_BITS, BLOCK > Class Template Reference

Handles a reference to a read only register block of the given maximum access width.

```
#include <hw_register_block>
```

Collaboration diagram for L4drivers::Ro_register_block< MAX_BITS, BLOCK >:



Public Member Functions

- [Ro_register operator\[\]](#) (unsigned offset) const
Read only access to register at offset offset.
- [template<unsigned BITS>](#)
[Ro_register_tmpl](#)< BITS, Block > [r](#) (unsigned offset) const
Read only access to register at offset offset.

16.276.1 Detailed Description

```
template<unsigned MAX_BITS, typename BLOCK = Register_block_tmpl< Register_block_base<MAX_BITS> const >>
class L4drivers::Ro_register_block< MAX_BITS, BLOCK >
```

Handles a reference to a read only register block of the given maximum access width.

Template Parameters

<i>MAX_BITS</i>	Maximum access width for the registers in this block.
<i>BLOCK</i>	Type implementing the register accesses (read<>()),

Provides read only access to registers in this block via [r<WIDTH>\(\)](#) and [operator\[\]\(\)](#).

Definition at line 404 of file [hw_register_block](#).

16.276.2 Member Function Documentation

16.276.2.1 operator[]()

```
template<unsigned MAX_BITS, typename BLOCK = Register_block_tmpl< Register_block_base<MAX_BITS> const >>
Ro_register L4drivers::Ro_register_block< MAX_BITS, BLOCK >::operator[] (
    unsigned offset) const [inline]
```

Read only access to register at offset *offset*.

Parameters

<i>offset</i>	The offset of the register within the register file.
---------------	--

Returns

register object allowing read only access with width *MAX_BITS*.

Definition at line 426 of file [hw_register_block](#).

16.276.2.2 r()

```
template<unsigned MAX_BITS, typename BLOCK = Register_block_tmpl< Register_block_base<MAX_BITS> const >>
template<unsigned BITS>
Ro_register_tmpl< BITS, Block > L4drivers::Ro_register_block< MAX_BITS, BLOCK >::r (
    unsigned offset) const [inline]
```

Read only access to register at offset *offset*.

Template Parameters

<i>BITS</i>	the access width in bits for the register.
-------------	--

Parameters

<i>offset</i>	The offset of the register within the register file.
---------------	--

Returns

register object allowing read only access with width *BITS*.

Definition at line 436 of file [hw_register_block](#).

The documentation for this class was generated from the following file:

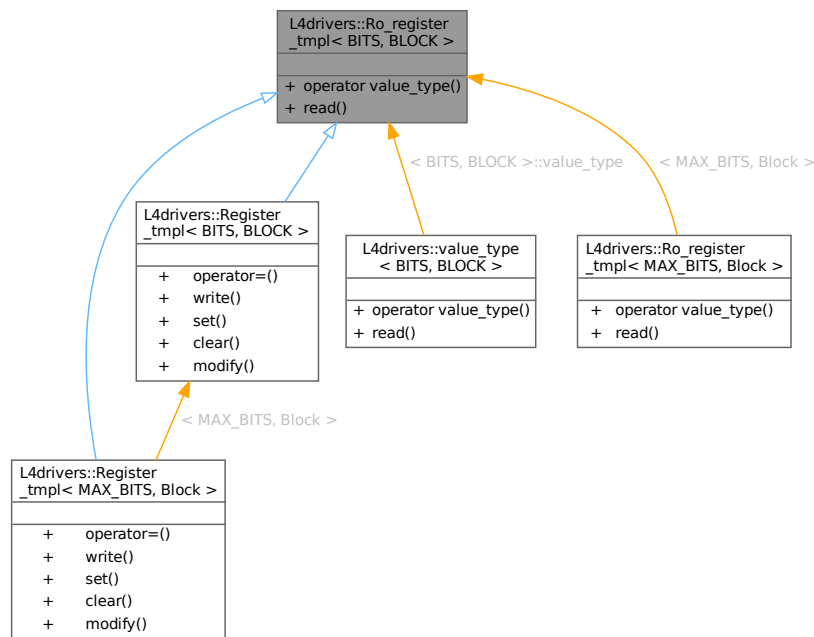
- pkg/drivers-frst/include/hw_register_block

16.277 L4drivers::Ro_register_tmpl< BITS, BLOCK > Class Template Reference

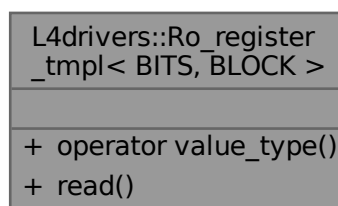
Single read only register inside a [Register_block_base](#) interface.

```
#include <hw_register_block>
```

Inheritance diagram for L4drivers::Ro_register_tmpl< BITS, BLOCK >:



Collaboration diagram for L4drivers::Ro_register_tmpl< BITS, BLOCK >:



Public Member Functions

- `operator value_type()` const
read the value from the hardware register.
- `value_type read()` const
read the value from the hardware register.

16.277.1 Detailed Description

```
template<unsigned BITS, typename BLOCK>
class L4drivers::Ro_register_tmpl< BITS, BLOCK >
```

Single read only register inside a [Register_block_base](#) interface.

Template Parameters

<i>BITS</i>	The access with of the register in bits.
<i>BLOCK</i>	The type for the Register_block_base interface.

Note

Objects of this type must be used only in temporary contexts not in global, class, or object scope.

Allows simple read only access to a hardware register.

Definition at line 201 of file [hw_register_block](#).

16.277.2 Member Function Documentation

16.277.2.1 operator value_type()

```
template<unsigned BITS, typename BLOCK>
L4drivers::Ro_register_tmpl< BITS, BLOCK >::operator value_type () const [inline]
```

read the value from the hardware register.

Returns

value read from the hardware register.

Definition at line 217 of file [hw_register_block](#).

16.277.2.2 read()

```
template<unsigned BITS, typename BLOCK>
value_type L4drivers::Ro_register_tmpl< BITS, BLOCK >::read () const [inline]
```

read the value from the hardware register.

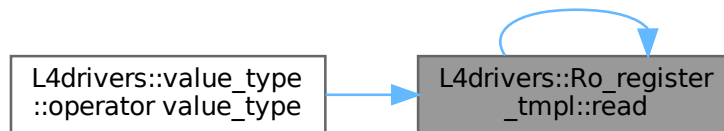
Returns

value from the hardware register.

Definition at line 224 of file [hw_register_block](#).

Referenced by [L4drivers::value_type< BITS, BLOCK >::operator value_type\(\)](#), and [L4drivers::value_type< BITS, BLOCK >::read\(\)](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

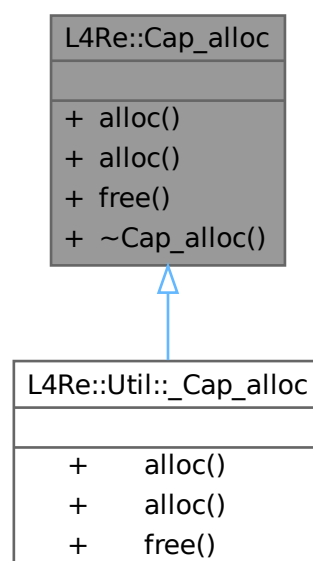
- `pkg/drivers-frst/include/hw_register_block`

16.278 L4Re::Cap_alloc Class Reference

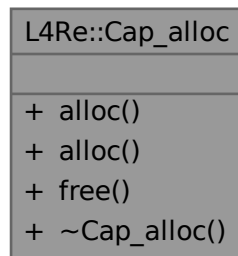
Capability allocator interface.

```
#include <cap_alloc>
```

Inheritance diagram for L4Re::Cap_alloc:



Collaboration diagram for L4Re::Cap_alloc:



Public Member Functions

- virtual [L4::Cap](#)< void > [alloc](#) () noexcept=0
Allocate a capability.
- template<typename T>
[L4::Cap](#)< T > [alloc](#) () noexcept
Allocate a capability.
- virtual void [free](#) ([L4::Cap](#)< void > cap, [l4_cap_idx_t](#) task=[L4_INVALID_CAP](#), unsigned unmap_↔
flags=[L4_FP_ALL_SPACES](#)) noexcept=0
Free a capability.
- virtual ~[Cap_alloc](#) ()=0
Destructor.

16.278.1 Detailed Description

Capability allocator interface.

Definition at line 30 of file [cap_alloc](#).

16.278.2 Member Function Documentation

16.278.2.1 [alloc\(\)](#) [1/2]

```
template<typename T>
L4::Cap< T > L4Re::Cap_alloc::alloc () [inline], [noexcept]
```

Allocate a capability.

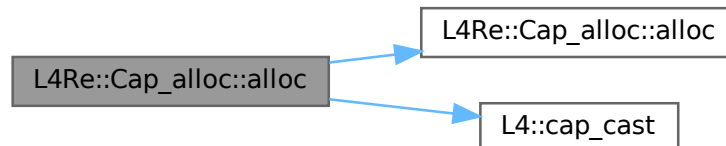
Returns

Capability of type T

Definition at line 53 of file [cap_alloc](#).

References [alloc\(\)](#), and [L4::cap_cast\(\)](#).

Here is the call graph for this function:

**16.278.2.2 alloc() [2/2]**

```
virtual L4::Cap< void > L4Re::Cap_alloc::alloc () [pure virtual], [noexcept]
```

Allocate a capability.

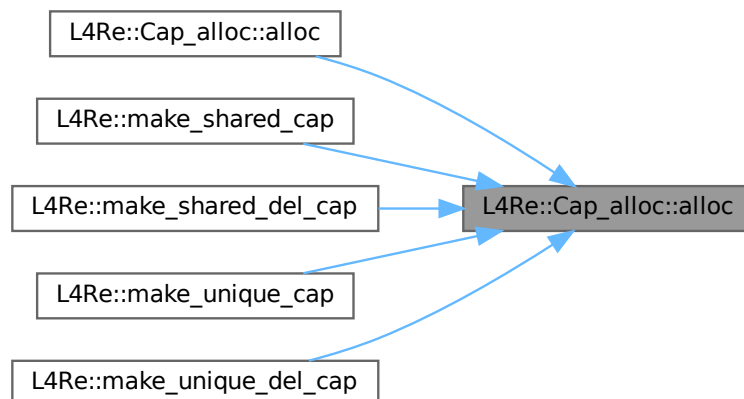
Returns

Capability of type void

Implemented in [L4Re::Util::_Cap_alloc](#), and [L4Re::Util::_Cap_alloc](#).

Referenced by [alloc\(\)](#), [L4Re::make_shared_cap\(\)](#), [L4Re::make_shared_del_cap\(\)](#), [L4Re::make_unique_cap\(\)](#), and [L4Re::make_unique_del_cap\(\)](#).

Here is the caller graph for this function:



16.278.2.3 free()

```
virtual void L4Re::Cap_alloc::free (  
    L4::Cap< void > cap,  
    l4_cap_idx_t task = L4_INVALID_CAP,  
    unsigned unmap_flags = L4_FP_ALL_SPACES) [pure virtual], [noexcept]
```

Free a capability.

Parameters

<i>cap</i>	Capability to free.
<i>task</i>	If set, task to unmap the capability from.
<i>unmap_flags</i>	Flags for unmap, see l4_unmap_flags_t .

Implemented in [L4Re::Util::_Cap_alloc](#).

References [L4_FP_ALL_SPACES](#), and [L4_INVALID_CAP](#).

The documentation for this class was generated from the following file:

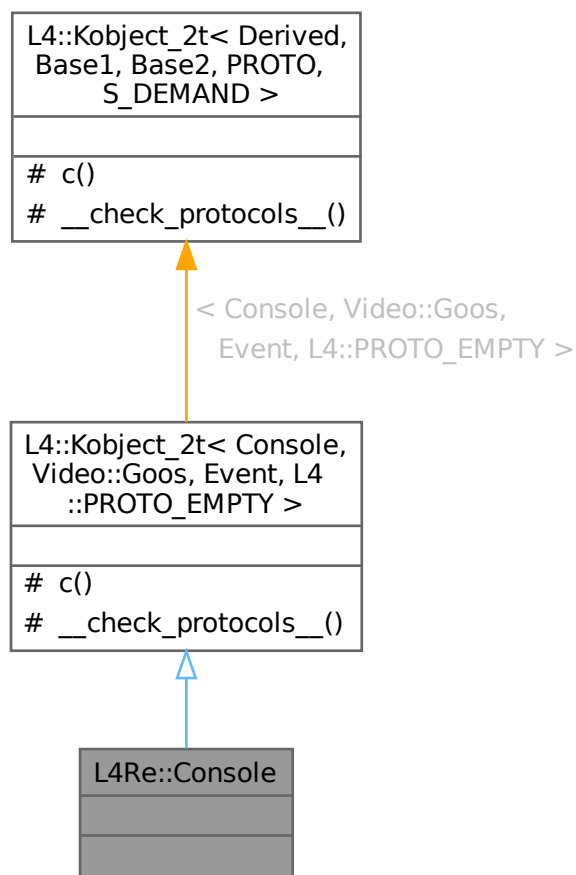
- [l4/re/cap_alloc](#)

16.279 L4Re::Console Class Reference

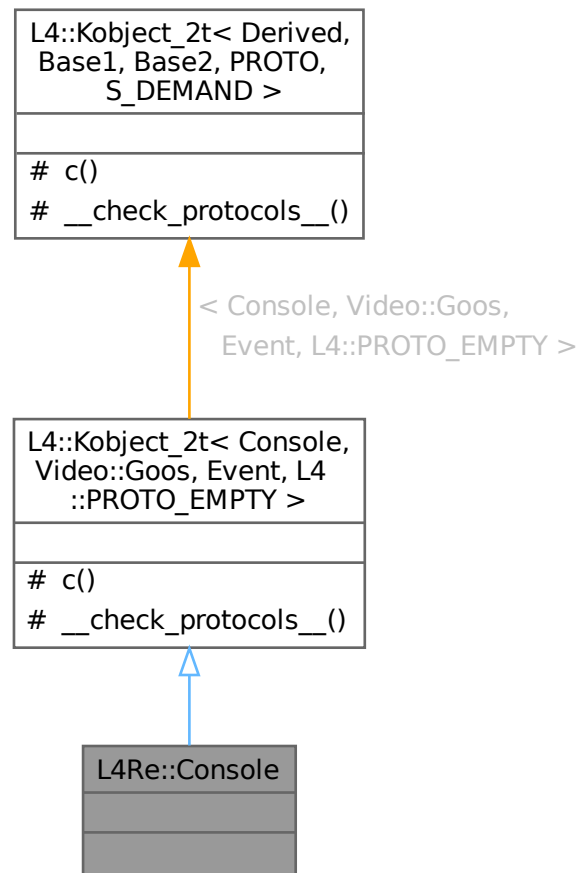
[Console](#) class.

```
#include <console>
```

Inheritance diagram for L4Re::Console:



Collaboration diagram for L4Re::Console:



Additional Inherited Members

Protected Types inherited from

L4::Kobject_2t< Console, Video::Goos, Event, L4::PROTO_EMPTY >

- typedef Console [Class](#)
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< PROTO, Console > [__iface](#)
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< [__iface](#) >, Typeid::Merge_list< typename Video::Goos::_↔
_iface_list, typename Event::_iface_list > > [__iface_list](#)
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from

L4::Kobject_2t< Console, Video::Goos, Event, L4::PROTO_EMPTY >

- [L4::Cap< Class > c \(\)](#) const noexcept
Get the capability to ourselves.

Static Protected Member Functions inherited from L4::Kobject_2t< Console, Video::Goos, Event, L4::PROTO_EMPTY >

- static void `__check_protocols__()` noexcept
Helper to check for protocol conflicts.

16.279.1 Detailed Description

`Console` class.

Definition at line 28 of file `console`.

The documentation for this class was generated from the following file:

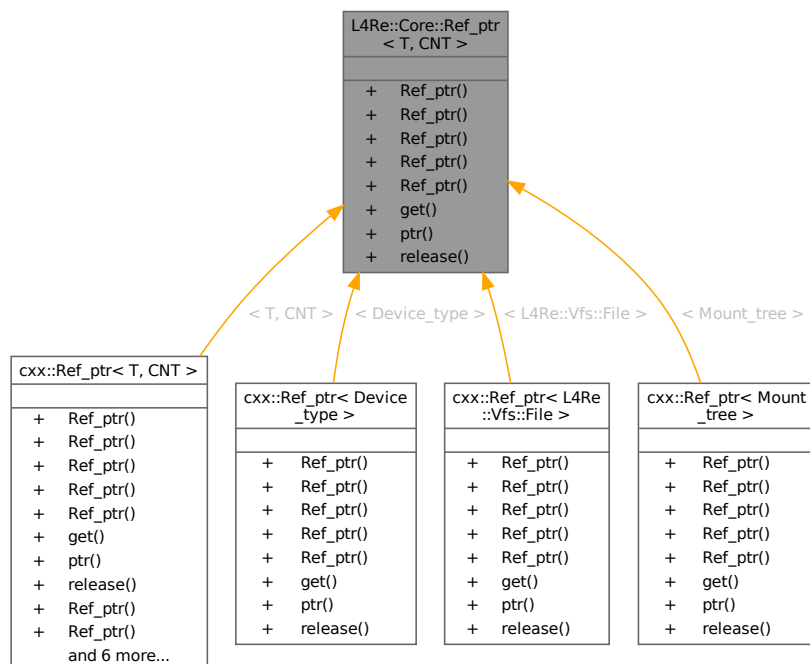
- `l4/re/console`

16.280 L4Re::Core::Ref_ptr< T, CNT > Class Template Reference

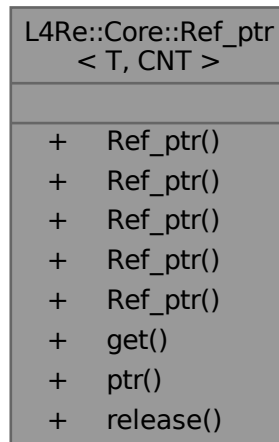
A reference-counting pointer with automatic cleanup.

```
#include <ref_ptr>
```

Inheritance diagram for L4Re::Core::Ref_ptr< T, CNT >:



Collaboration diagram for L4Re::Core::Ref_ptr< T, CNT >:



Public Member Functions

- **Ref_ptr** () noexcept
Default constructor creates a pointer with no managed object.
- **Ref_ptr** (Wp const &o) noexcept
Create a shared pointer from a weak pointer.
- **Ref_ptr** (decltype(nullptr) n) noexcept
allow creation from `nullptr`
- template<typename X>
Ref_ptr (X *o) noexcept
Create a shared pointer from a raw pointer.
- **Ref_ptr** (T *o, bool d) noexcept
Create a shared pointer from a raw pointer without creating a new reference.
- T * **get** () const noexcept
Return a raw pointer to the object this shared pointer points to.
- T * **ptr** () const noexcept
Return a raw pointer to the object this shared pointer points to.
- T * **release** () noexcept
Release the shared pointer without removing the reference.

16.280.1 Detailed Description

```
template<typename T = void, template< typename X > class CNT = Default_ref_counter>
class L4Re::Core::Ref_ptr< T, CNT >
```

A reference-counting pointer with automatic cleanup.

Template Parameters

<i>T</i>	Type of object the pointer points to.
<i>CNT</i>	Type of management class that manages the life time of the object.

This pointer is similar to the standard C++-11 `shared_ptr` but it does the reference counting directly in the object being pointed to, so that no additional management structures need to be allocated from the heap.

Classes that use this pointer type must implement two functions:

```
int remove_ref()
```

is called when a reference is removed and must return 0 when there are no further references to the object.

```
void add_ref()
```

is called when another `ref_ptr` to the object is created.

`Ref_obj` provides a simple implementation of this interface from which classes may inherit.

Definition at line 70 of file [ref_ptr](#).

16.280.2 Constructor & Destructor Documentation

16.280.2.1 `Ref_ptr()` [1/3]

```
template<typename T = void, template< typename X > class CNT = Default_ref_counter>
cxx::Ref_ptr< T, CNT >::Ref_ptr (
    Wp const & o) [inline], [noexcept]
```

Create a shared pointer from a weak pointer.

Increases references.

Definition at line 88 of file [ref_ptr](#).

16.280.2.2 `Ref_ptr()` [2/3]

```
template<typename T = void, template< typename X > class CNT = Default_ref_counter>
template<typename X>
cxx::Ref_ptr< T, CNT >::Ref_ptr (
    X * o) [inline], [explicit], [noexcept]
```

Create a shared pointer from a raw pointer.

In contrast to C++11 `shared_ptr` it is safe to use this constructor multiple times and have the same reference counter.

Definition at line 101 of file [ref_ptr](#).

16.280.2.3 `Ref_ptr()` [3/3]

```
template<typename T = void, template< typename X > class CNT = Default_ref_counter>
cxx::Ref_ptr< T, CNT >::Ref_ptr (
    T * o,
    bool d) [inline], [noexcept]
```

Create a shared pointer from a raw pointer without creating a new reference.

Parameters

<i>o</i>	Pointer to the object.
<i>d</i>	Dummy parameter to select this constructor at compile time. The value may be true or false.

This is the counterpart to [release\(\)](#).

Definition at line 114 of file [ref_ptr](#).

16.280.3 Member Function Documentation

16.280.3.1 get()

```
template<typename T = void, template< typename X > class CNT = Default_ref_counter>
T * cxx::Ref_ptr< T, CNT >::get () const [inline], [noexcept]
```

Return a raw pointer to the object this shared pointer points to.

This does not release the pointer or decrease the reference count.

Definition at line 121 of file [ref_ptr](#).

16.280.3.2 ptr()

```
template<typename T = void, template< typename X > class CNT = Default_ref_counter>
T * cxx::Ref_ptr< T, CNT >::ptr () const [inline], [noexcept]
```

Return a raw pointer to the object this shared pointer points to.

This does not release the pointer or decrease the reference count.

Definition at line 127 of file [ref_ptr](#).

16.280.3.3 release()

```
template<typename T = void, template< typename X > class CNT = Default_ref_counter>
T * cxx::Ref_ptr< T, CNT >::release () [inline], [noexcept]
```

Release the shared pointer without removing the reference.

Returns

A raw pointer to the managed object.

Definition at line 138 of file [ref_ptr](#).

The documentation for this class was generated from the following file:

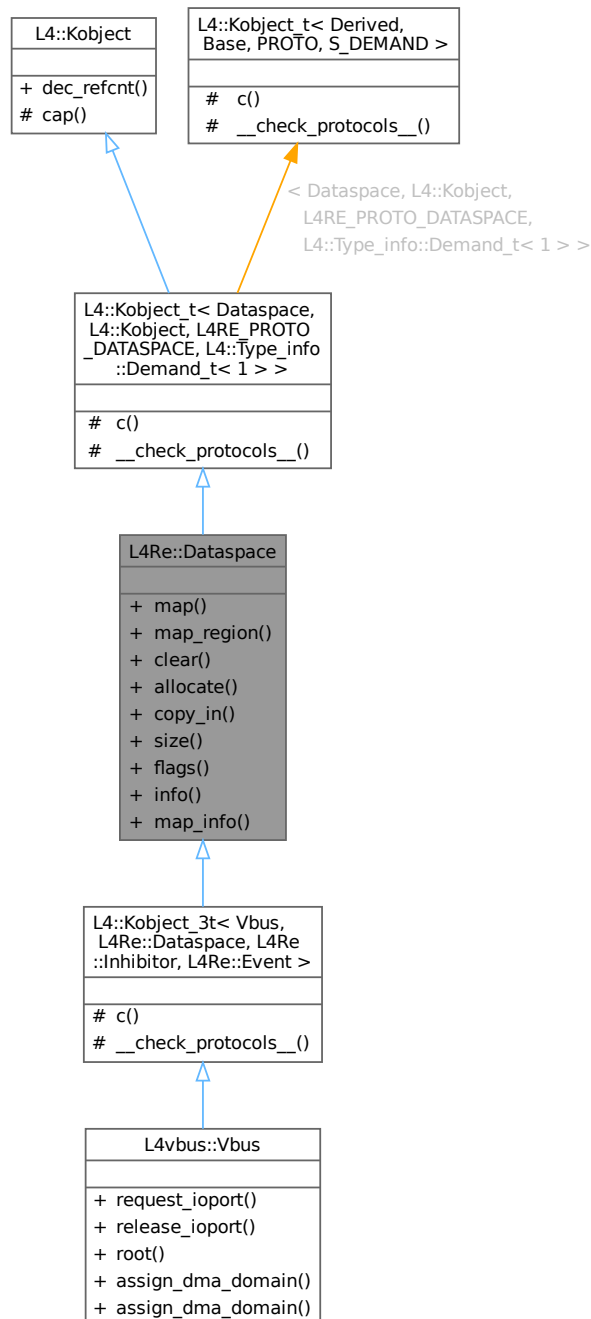
- [l4/cxx/ref_ptr](#)

16.281 L4Re::Dataspace Class Reference

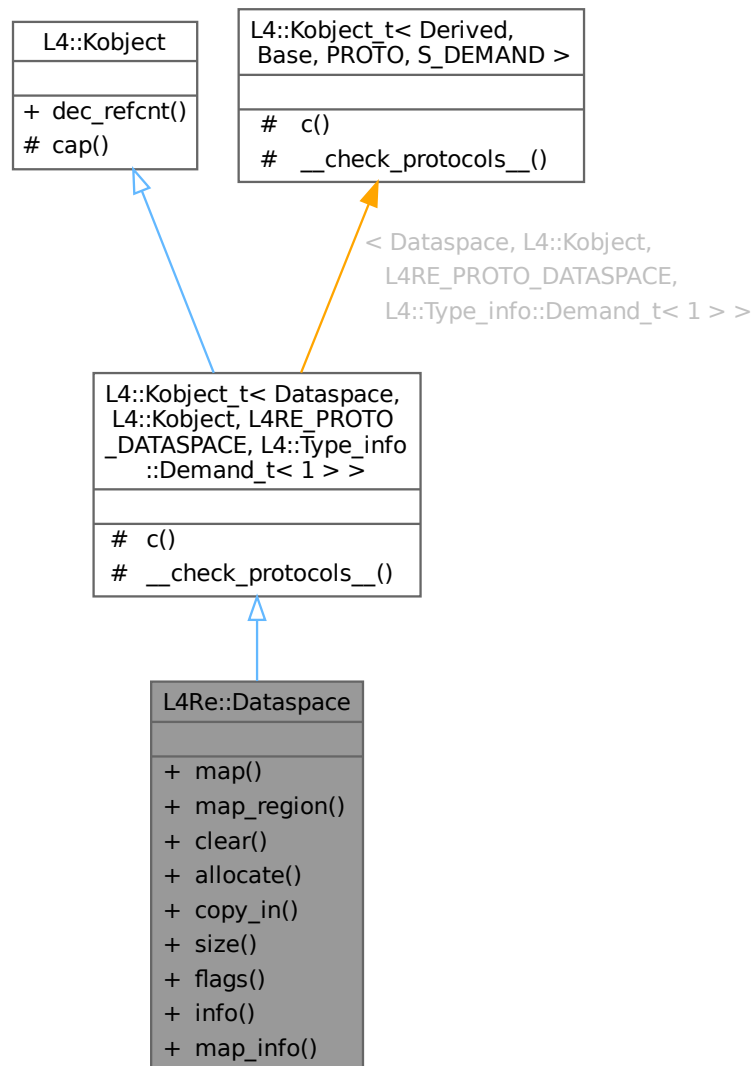
Interface for memory-like objects.

```
#include <dataspace>
```

Inheritance diagram for L4Re::Dataspace:



Collaboration diagram for L4Re::Dataspace:



Data Structures

- struct [F](#)
Dataspace flags definitions.
- struct [Stats](#)
Information about the dataspace.

Public Member Functions

- long [map](#) (Offset offset, Flags [flags](#), Map_addr local_addr, Map_addr min_addr, Map_addr max_addr, [L4::Cap](#)< [L4::Task](#) > dst=[L4::Cap](#)< [L4::Task](#) >::Invalid) const noexcept
Request a flexpage mapping from the dataspace.

- long [map_region](#) (Offset offset, Flags [flags](#), Map_addr min_addr, Map_addr max_addr, [L4::Cap](#)< [L4::Task](#) > dst=[L4::Cap](#)< [L4::Task](#) >::Invalid) const noexcept
Map a part of a dataspace into a local memory area.
- long [clear](#) (Offset offset, Size [size](#))
Clear parts of a dataspace.
- long [allocate](#) (Offset offset, Size [size](#))
Allocate a range in the dataspace.
- long [copy_in](#) (Offset dst_offs, [L4::lpc::Cap](#)< [Dataspace](#) > src, Offset src_offs, Size [size](#))
Copy contents from another dataspace.
- Size [size](#) () const noexcept
Get size of a dataspace.
- Flags [flags](#) () const noexcept
Get flags of the dataspace.
- long [info](#) ([Stats](#) *stats)
Get information on the dataspace.
- long [map_info](#) ([l4_addr_t](#) *start_addr, [l4_addr_t](#) *end_addr)
Get mapping range of dataspace.

Public Member Functions inherited from [L4::Kobject](#)

- [l4_msgtag_t dec_refcnt](#) ([l4_mword_t](#) diff, [l4_utcb_t](#) *utcb=[l4_utcb](#)())
Decrement the in kernel reference counter for the object.

Additional Inherited Members

Protected Types inherited from

[L4::Kobject_t](#)< [Dataspace](#), [L4::Kobject](#), [L4RE_PROTO_DATASPACE](#), [L4::Type_info::Demand_t](#)< 1 > >

- typedef [Dataspace](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef [Typeid::Iface](#)< [PROTO](#), [Dataspace](#) > **__Iface**
The interface description for the derived class.
- typedef [Typeid::Merge_list](#)< [Typeid::Iface_list](#)< [__Iface](#) >, typename [L4::Kobject::__Iface_list](#) > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from

[L4::Kobject_t](#)< [Dataspace](#), [L4::Kobject](#), [L4RE_PROTO_DATASPACE](#), [L4::Type_info::Demand_t](#)< 1 > >

- [L4::Cap](#)< [Class](#) > **c** () const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from [L4::Kobject](#)

- [l4_cap_idx_t cap](#) () const noexcept
Return capability selector.

Static Protected Member Functions inherited from**L4::Kobject_t< Dataspace, L4::Kobject, L4RE_PROTO_DATASPACE, L4::Type_info::Demand_t< 1 > >**

- static void **__check_protocols__** () noexcept
Helper to check for protocol conflicts.

16.281.1 Detailed Description

Interface for memory-like objects.

Dataspaces are a central abstraction provided by [L4Re](#). A dataspace is an abstraction for any thing that is available via usual memory access instructions. A dataspace can be a file, as well as the memory-mapped registers of a device, or anonymous memory, such as a heap.

The dataspace interface defines a set of methods that allow any kind of dataspace to be attached (mapped) to the virtual address space of an [L4](#) task and then be accessed via memory-access instructions. The [L4Re::Rm](#) interface can be used to attach a dataspace to a virtual address space of a task paged by a certain instance of a region map.

Include File

```
#include <l4/re/dataspace>
```

Examples

[examples/libs/l4re/c++/mem_alloc/ma+rm.cc](#), [examples/libs/l4re/c++/shared_ds/ds_clnt.cc](#), and [examples/libs/l4re/c++/shared](#)

Definition at line 50 of file [dataspace](#).

16.281.2 Member Function Documentation**16.281.2.1 allocate()**

```
long L4Re::Dataspace::allocate (
    Offset offset,
    Size size)
```

Allocate a range in the dataspace.

Parameters

<i>offset</i>	Offset in the dataspace, in bytes.
<i>size</i>	Size of the range, in bytes.

Return values

<i>L4_EOK</i>	Success
---------------	---------

<code>-L4_ERANGE</code>	Given range is outside the dataspace. (A dataspace provider may also silently ignore areas outside the dataspace.)
<code>-L4_ENOMEM</code>	Not enough memory available.
<code>< 0</code>	IPC errors

On success, at least the given range is guaranteed to be allocated. The dataspace manager may also allocate more memory due to page granularity.

The memory is allocated with the same rights as the dataspace capability.

References [allocate\(\)](#), [copy_in\(\)](#), and [size\(\)](#).

Referenced by [allocate\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.281.2.2 clear()

```

long L4Re::Dataspace::clear (
    Offset offset,
    Size size)

```

Clear parts of a dataspace.

Parameters

<i>offset</i>	Offset within dataspace (in bytes).
<i>size</i>	Size of region to clear (in bytes).

Return values

≥ 0	Success.
<code>-L4_ERANGE</code>	Given range is outside the dataspace. (A dataspace provider may also silently ignore areas outside the dataspace.)
<code>-L4_EACCESS</code>	No <code>L4_CAP_FPAGE_W</code> right on dataspace capability.
< 0	IPC errors

Zeroes out the memory. Depending on the type of memory the memory could also be deallocated and replaced by a shared zero-page.

References [clear\(\)](#), and [size\(\)](#).

Referenced by [clear\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.281.2.3 copy_in()

```

long L4Re::Dataspace::copy_in (
    Offset dst_offs,
    L4::Ipc::Cap< Dataspace > src,
    Offset src_offs,
    Size size)

```

Copy contents from another dataspace.

Parameters

<code>dst_offs</code>	Offset in destination dataspace.
-----------------------	----------------------------------

<i>src</i>	Source dataspace to copy from.
<i>src_offs</i>	Offset in the source dataspace.
<i>size</i>	Size to copy (in bytes).

Return values

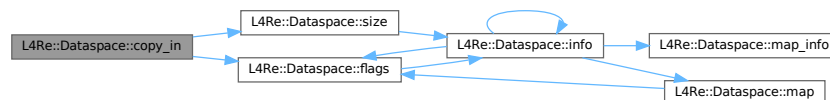
<i>L4_EOK</i>	Success
<i>-L4_EACCESS</i>	No L4_CAP_FPAGE_W right on the destination dataspace.
<i>-L4_EINVAL</i>	Invalid parameter supplied.
<i>< 0</i>	IPC errors

The copy operation may use copy-on-write mechanisms. The operation may also fail if both dataspace managers are not from the same dataspace manager or the dataspace managers do not cooperate.

References [flags\(\)](#), and [size\(\)](#).

Referenced by [allocate\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.281.2.4 flags()

```
Dataspace::Flags Dataspace::flags () const [noexcept]
```

Get flags of the dataspace.

Return values

≥ 0	Flags of the dataspace
< 0	IPC errors

See also

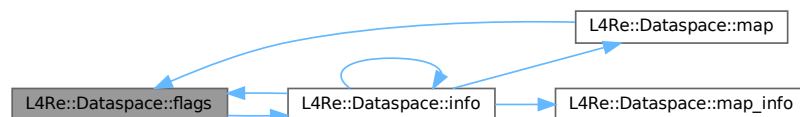
[L4Re::Dataspace::F::Flags](#)

Definition at line 111 of file [dataspace_impl.h](#).

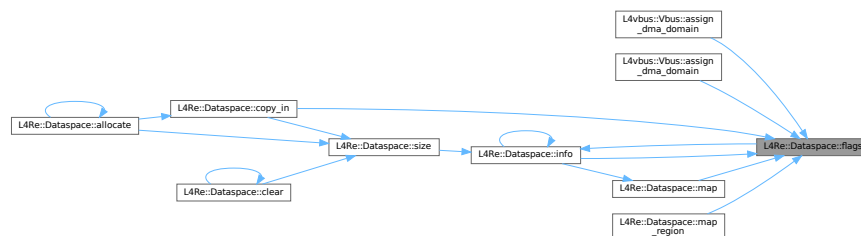
References [L4Re::Dataspace::Stats::flags](#), and [info\(\)](#).

Referenced by [L4vbus::Vbus::assign_dma_domain\(\)](#), [L4vbus::Vbus::assign_dma_domain\(\)](#), [copy_in\(\)](#), [info\(\)](#), [map\(\)](#), and [map_region\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.281.2.5 info()

```
long L4Re::Dataspace::info (
    Stats * stats)
```

Get information on the dataspace.

Parameters

out	stats	Dataspace information
-----	-------	---------------------------------------

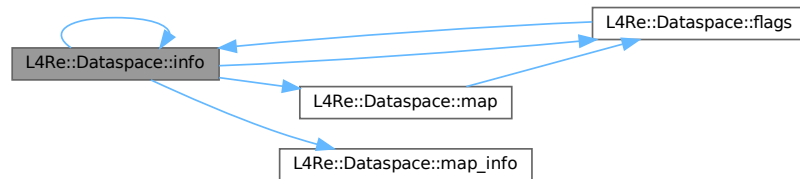
Return values

0	Success
<0	IPC errors

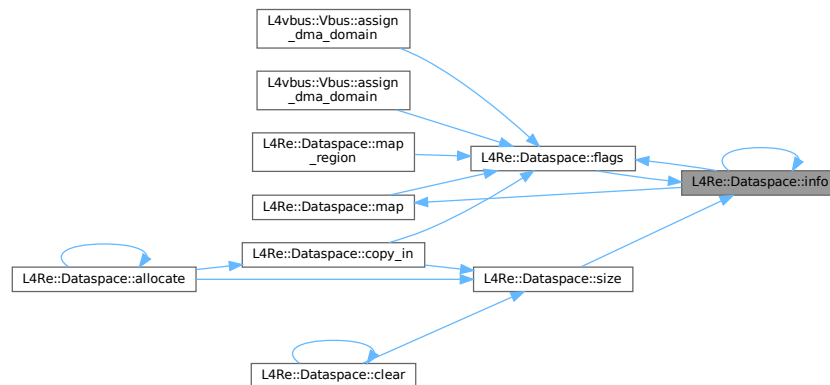
References [flags\(\)](#), [info\(\)](#), [L4_RPC](#), [L4_RPC_NF](#), [map\(\)](#), and [map_info\(\)](#).

Referenced by [flags\(\)](#), [info\(\)](#), and [size\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.281.2.6 map()

```

long Dataspace::map (
    Dataspace::Offset offset,
    Dataspace::Flags flags,
    Dataspace::Map_addr local_addr,
    Dataspace::Map_addr min_addr,
    Dataspace::Map_addr max_addr,
    L4::Cap< L4::Task > dst = L4::Cap<L4::Task>::Invalid) const [noexcept]

```

Request a flexpage mapping from the dataspace.

Parameters

<i>offset</i>	Offset to start within dataspace
<i>flags</i>	Dataspace flags, see L4Re::Dataspace::F::Flags .
<i>local_addr</i>	Local address to map to.
<i>min_addr</i>	Defines start of receive window. (Rounded down to page size.)
<i>max_addr</i>	Defines end of receive window. (Rounded up to page size.)
<i>dst</i>	Optional destination task of the mapping. If invalid, the callers task is implicitly the destination.

Return values

<i>L4_EOK</i>	Success
<i>-L4_ERANGE</i>	Invalid offset.
<i>-L4_EPERM</i>	Insufficient permission to map with requested rights.
<i><0</i>	IPC errors

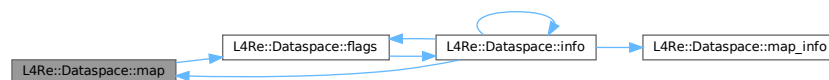
The map call will attempt to map the largest possible flexpage that covers the given local address and still fits into the region defined by `min_addr` and `max_addr`. If the given region is invalid or does not overlap the local address, the smallest valid page size is used.

Definition at line 85 of file [dataspace_impl.h](#).

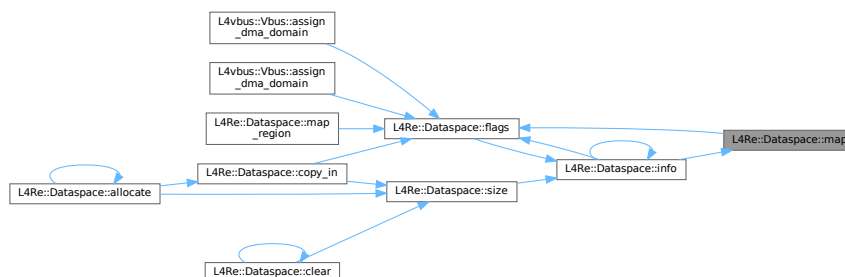
References [flags\(\)](#), and [L4_LOG2_PAGESIZE](#).

Referenced by [info\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.281.2.7 map_info()

```
long L4Re::Dataspace::map_info (
    l4_addr_t * start_addr,
    l4_addr_t * end_addr) [inline]
```

Get mapping range of dataspace.

In case of a MMU-less system, the dataspace must be mapped at the correct address in the task because virtual and physical address must match. This method returns the start and end address of the physically contiguous buffer backing the dataspace.

On MMU-enabled system any page aligned address is permissible. On such systems the method is just a stub.

Parameters

out	<i>start_addr</i>	Start address of dataspace.
out	<i>end_addr</i>	End address (inclusive) of dataspace.

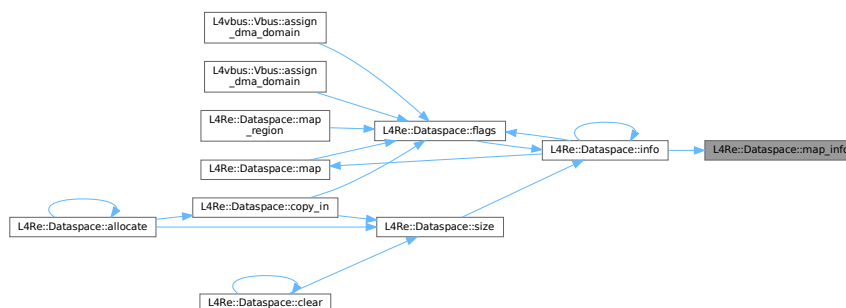
Return values

>0	Start/end address have been set and need to be obeyed.
0	No constraint of mapping address.
-L4_EPERM	Cannot infer mapping address. Dataspace not mappable.
<0	IPC errors.

Definition at line 305 of file [dataspace](#).

Referenced by [info\(\)](#).

Here is the caller graph for this function:



16.281.2.8 map_region()

```
long Dataspace::map_region (
    Dataspace::Offset offset,
    Dataspace::Flags flags,
    Dataspace::Map_addr min_addr,
    Dataspace::Map_addr max_addr,
    L4::Cap< L4::Task > dst = L4::Cap<L4::Task>::Invalid) const [noexcept]
```

Map a part of a dataspace into a local memory area.

Parameters

<i>offset</i>	Offset to start within dataspace.
<i>flags</i>	Dataspace flags, see L4Re::Dataspace::F::Flags .
<i>min_addr</i>	(Inclusive) start of the receive area.
<i>max_addr</i>	(Exclusive) end of receive area.
<i>dst</i>	Optional destination task of the mapping. If invalid, the callers task is implicitly the destination.

Return values

<i>L4_EOK</i>	Success
<i>-L4_ERANGE</i>	Invalid offset or receive area larger than the dataspace.
<i>-L4_EPERM</i>	Insufficient permission to map with requested rights.
<i><0</i>	IPC errors

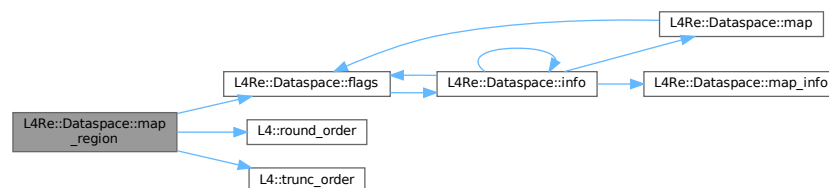
This is a convenience function which maps flexpages consecutively into the given memory area in the local task. The area is expected to be filled completely. If the dataspace is not large enough to provide the mappings for the entire size of the area, then an error is returned. Mappings may or may not have been already established at that point.

offset and *min_addr* are rounded down to the next [L4_PAGESIZE](#) boundary when necessary. *max_addr* is rounded up to the page boundary. If the resulting maximum address is less or equal than the minimum address, then the function is a noop.

Definition at line 45 of file [dataspace_impl.h](#).

References [flags\(\)](#), [L4_LOG2_PAGESIZE](#), [L4_UNLIKELY](#), [L4::round_order\(\)](#), and [L4::trunc_order\(\)](#).

Here is the call graph for this function:



16.281.2.9 size()

```
Dataspace::Size Dataspace::size () const [noexcept]
```

Get size of a dataspace.

Returns

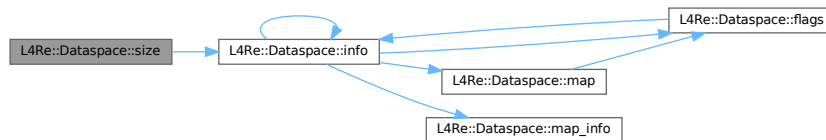
Size of the dataspace in bytes.

Definition at line 101 of file [dataspace_impl.h](#).

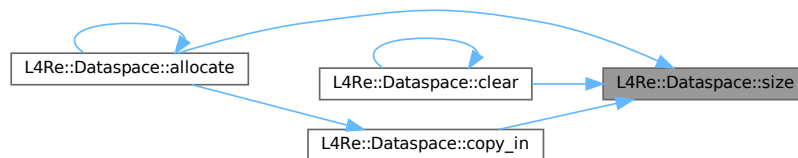
References [info\(\)](#), and [L4Re::Dataspace::Stats::size](#).

Referenced by [allocate\(\)](#), [clear\(\)](#), and [copy_in\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following files:

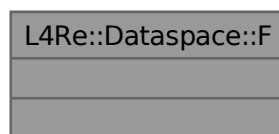
- [l4/re/dataspace](#)
- [l4/re/impl/dataspace_impl.h](#)

16.282 L4Re::Dataspace::F Struct Reference

[Dataspace](#) flags definitions.

```
#include <dataspace>
```

Collaboration diagram for L4Re::Dataspace::F:



Public Types

- enum { [Caching_shift](#) = 4 }
- enum [Flags](#) {
[R](#) = L4_FPAGE_RO , [Ro](#) = L4_FPAGE_RO , [RW](#) = L4_FPAGE_RW , [W](#) = L4_FPAGE_W ,
[X](#) = L4_FPAGE_X , [RX](#) = L4_FPAGE_RX , [RWX](#) = L4_FPAGE_RWX , [Rights_mask](#) = 0x0f ,
[Normal](#) = 0x00 , [Cacheable](#) = Normal , [Bufferable](#) = 0x10 , [Uncacheable](#) = 0x20 ,
[Caching_mask](#) = 0x30 }
[Flags](#) for map operations.

16.282.1 Detailed Description

[Dataspace](#) flags definitions.

Definition at line 57 of file [dataspace](#).

16.282.2 Member Enumeration Documentation

16.282.2.1 anonymous enum

anonymous enum

Enumerator

Caching_shift	shift value for caching flags
---------------	-------------------------------

Definition at line 59 of file [dataspace](#).

16.282.2.2 Flags

enum [L4Re::Dataspace::F::Flags](#)

[Flags](#) for map operations.

A dataspace implementation must check the requested flags during the map and other operations against the dataspace rights.

Enumerator

R	Request read-only mapping.
Ro	Request read-only mapping.
RW	Request read-write mapping.
W	Request write-only mapping.
X	Request execute-only mapping.
RX	Request read-execute mapping.
RWX	Request read-write-execute mapping.

Rights_mask	All rights bits available for mappings.
Normal	Request normal (cached) memory mapping. This is the default if no other cache-related flag was specified.
Cacheable	Request normal memory mapping.
Bufferable	Request bufferable (write buffered) mappings.
Uncacheable	Request uncacheable memory mappings.
Caching_mask	Mask for caching flags.

Definition at line 70 of file [dataspace](#).

The documentation for this struct was generated from the following file:

- [l4/re/dataspace](#)

16.283 L4Re::Dataspace::Stats Struct Reference

Information about the dataspace.

```
#include <dataspace>
```

Collaboration diagram for L4Re::Dataspace::Stats:

L4Re::Dataspace::Stats	
+	size
+	flags

Data Fields

- Size **size**
size
- Flags **flags**
flags

16.283.1 Detailed Description

Information about the dataspace.

Definition at line 126 of file [dataspace](#).

The documentation for this struct was generated from the following file:

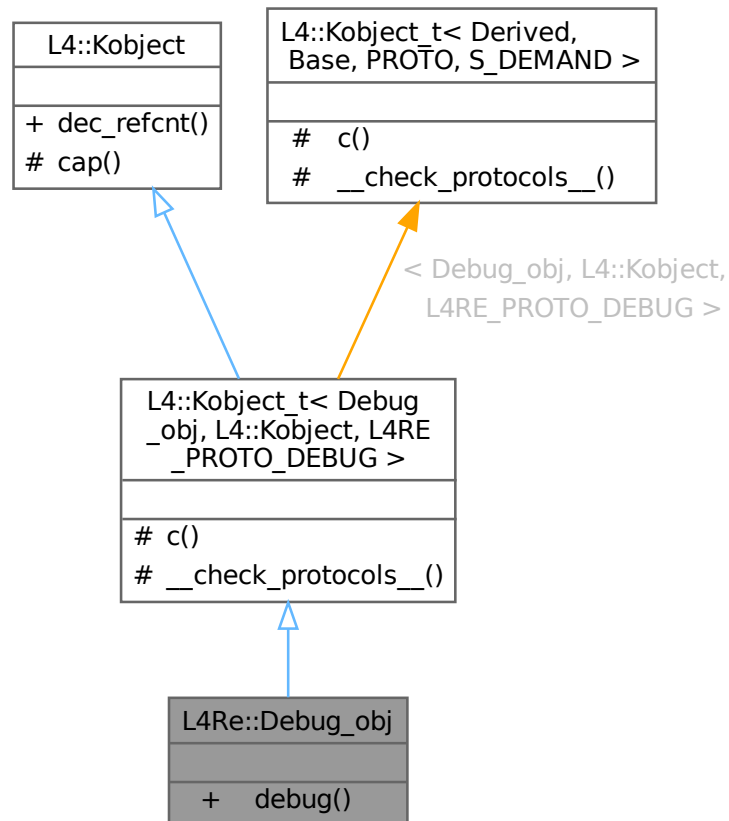
- [l4/re/dataspace](#)

16.284 L4Re::Debug_obj Class Reference

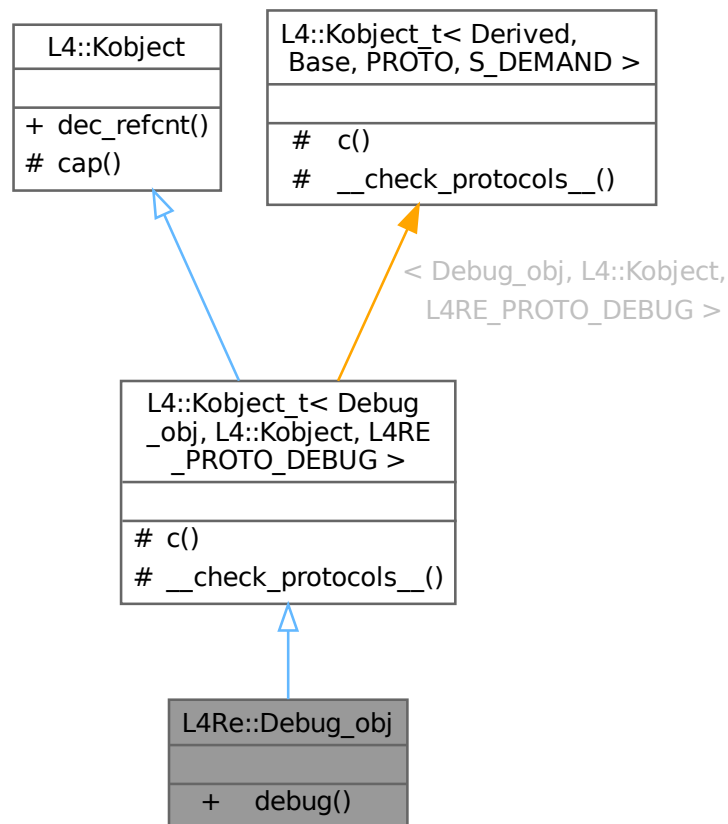
Debug interface.

```
#include <debug>
```

Inheritance diagram for L4Re::Debug_obj:



Collaboration diagram for L4Re::Debug_obj:



Public Member Functions

- long [debug](#) (unsigned long function)
Debug call.

Public Member Functions inherited from [L4::Kobject](#)

- [l4_msgtag_t dec_refcnt](#) ([l4_mword_t diff](#), [l4_utcb_t *utcb=l4_utcb\(\)](#))
Decrement the in kernel reference counter for the object.

Additional Inherited Members

Protected Types inherited from

[L4::Kobject_t< Debug_obj, L4::Kobject, L4RE_PROTO_DEBUG >](#)

- typedef [Debug_obj](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef [Typeid::Iface< PROTO, Debug_obj >](#) **__Iface**
The interface description for the derived class.
- typedef [Typeid::Merge_list< Typeid::Iface_list< \[__Iface\]\(#\) >, typename L4::Kobject::__Iface_list >](#) **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from [L4::Kobject_t< Debug_obj, L4::Kobject, L4RE_PROTO_DEBUG >](#)

- [L4::Cap< Class > c \(\)](#) const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from [L4::Kobject](#)

- [l4_cap_idx_t cap \(\)](#) const noexcept
Return capability selector.

Static Protected Member Functions inherited from [L4::Kobject_t< Debug_obj, L4::Kobject, L4RE_PROTO_DEBUG >](#)

- static void [__check_protocols__ \(\)](#) noexcept
Helper to check for protocol conflicts.

16.284.1 Detailed Description

Debug interface.

See also

[Debugging API](#) .

Definition at line 40 of file [debug](#).

16.284.2 Member Function Documentation

16.284.2.1 debug()

```
long L4Re::Debug_obj::debug (
    unsigned long function)
```

Debug call.

Parameters

<i>function</i>	Function to call.
-----------------	-------------------

Returns

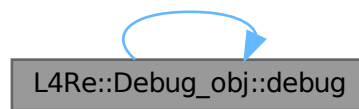
- L4_EOK
- IPC errors

An object can provide a number of debug functions, each identified by some integer. This method is used to fan out to these functions from a common entry point.

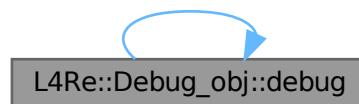
References [debug\(\)](#).

Referenced by [debug\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

- [l4/re/debug](#)

16.285 L4Re::Default_event_payload Struct Reference

Default event stream payload.

```
#include <event>
```

Collaboration diagram for L4Re::Default_event_payload:

L4Re::Default_event_payload	
+	type
+	code
+	value
+	stream_id

Data Fields

- unsigned short **type**
Type of event.
- unsigned short **code**
Code of event.
- int **value**
Value of event.
- [l4_umword_t](#) **stream_id**
Stream ID.

16.285.1 Detailed Description

Default event stream payload.

Definition at line [232](#) of file [event](#).

The documentation for this struct was generated from the following file:

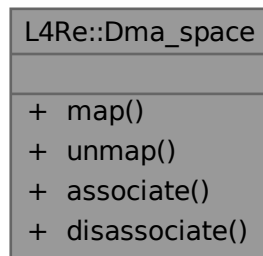
- [l4/re/event](#)

16.286 L4Re::Dma_space Class Reference

Managed DMA Address Space.

```
#include <dma_space>
```

Collaboration diagram for L4Re::Dma_space:



Public Types

- enum [Direction](#) { [Bidirectional](#) , [To_device](#) , [From_device](#) , [None](#) }
Direction of the DMA transfers.
- enum [Attribute](#) { [No_sync](#) }
Attributes used for the memory region during the transfer.
- enum [Space_attrib](#) { [Coherent](#) , [Phys_space](#) }
Attributes assigned to the DMA space when associated with a specific device.
- typedef [l4_uint64_t](#) [Dma_addr](#)
Data type for DMA addresses.
- typedef [L4::Types::Flags](#)< [Attribute](#) > [Attributes](#)
Attributes for DMA mappings.
- typedef [L4::Types::Flags](#)< [Space_attrib](#) > [Space_attribs](#)
Attributes used when configuring the DMA space.

Public Member Functions

- long [map](#) ([L4::lpc::Cap](#)< [L4Re::Dataspace](#) > src, [L4Re::Dataspace::Offset](#) offset, [L4::lpc::ln_out](#)< [l4_size_t](#) * > size, [Attributes](#) attrs, [Direction](#) dir, [Dma_addr](#) *dma_addr)
Map the given part of this data space into the DMA address space.
- long [unmap](#) ([Dma_addr](#) dma_addr, [l4_size_t](#) size, [Attributes](#) attrs, [Direction](#) dir)
Unmap the given part of this data space from the DMA address space.
- long [associate](#) ([L4::lpc::Opt](#)< [L4::lpc::Cap](#)< [L4::Task](#) > > dma_task, [Space_attribs](#) attr)
Associate a (kernel) DMA space for a device to this DMA_space.
- long [disassociate](#) ()
Disassociate the (kernel) DMA space from this DMA_space.

16.286.1 Detailed Description

Managed DMA Address Space.

A managed [Dma_space](#) represents the [L4Re](#) abstraction of an DMA address space of one or several devices. Devices are assigned to a managed [Dma_space](#) by binding the [Dma_space](#) to the respective DMA domain (see [L4vbus::Vbus::assign_dma_domain\(\)](#)), which might link the [Dma_space](#) with a kernel [DMA space](#). Note that several DMA domains can be bound to the same [Dma_space](#). Whenever a device needs direct access to parts of an [L4Re::Dataspace](#), that part of the data space must be mapped to the managed [Dma_space](#) that is assigned to that device. Binding to DMA domains must happen before mapping. After the DMA accesses to the memory are finished the memory must be unmapped from the device's DMA address space.

Mapping to a managed DMA address space, using [map\(\)](#), makes the given parts of the data space visible to the associated device at the returned DMA address. As long as the memory is mapped into a DMA space it is 'pinned' and cannot be subject to dynamic memory management such as swapping. Additionally, [map\(\)](#) is responsible for the necessary syncing operations before the DMA.

[unmap\(\)](#) is the reverse operation to [map\(\)](#) and unmaps the given data-space part for the DMA address space. [unmap\(\)](#) is responsible for the necessary sync operations after the DMA.

Definition at line 52 of file [dma_space](#).

16.286.2 Member Typedef Documentation

16.286.2.1 Attributes

```
typedef L4::Types::Flags<Attribute> L4Re::Dma_space::Attributes
```

[Attributes](#) for DMA mappings.

See also

[Attribute](#)

Definition at line 97 of file [dma_space](#).

16.286.3 Member Enumeration Documentation

16.286.3.1 Attribute

```
enum L4Re::Dma_space::Attribute
```

[Attributes](#) used for the memory region during the transfer.

See also

[Attributes](#)

Enumerator

No_sync	Do not sync the memory hierarchy. When this flag is <i>not set</i> (default) the memory region shall be made coherent to the point-of-coherency of the device associated with this Dma_space . When using this attribute the client is responsible for syncing the memory hierarchy for DMA. This can either be done using the cache API or by another map() or unmap() operation of the same part of the data space (without the No_sync attribute).
---------	---

Definition at line 76 of file [dma_space](#).

16.286.3.2 Direction

enum [L4Re::Dma_space::Direction](#)

[Direction](#) of the DMA transfers.

Enumerator

Bidirectional	device reads and writes to the memory
To_device	device reads the memory
From_device	device writes to the memory
None	device is coherently connected to the memory

Definition at line 64 of file [dma_space](#).

16.286.3.3 Space_attrib

enum [L4Re::Dma_space::Space_attrib](#)

[Attributes](#) assigned to the DMA space when associated with a specific device.

See also

[Space_attribs](#)

Enumerator

Coherent	The device is connected coherently with the cache. This means that the map() and unmap() do not need to sync CPU caches before and after DMA.
Phys_space	The DMA space has no DMA task assigned and uses the CPUs physical memory.

Definition at line 104 of file [dma_space](#).

16.286.4 Member Function Documentation

16.286.4.1 associate()

```
long L4Re::Dma_space::associate (
    L4::Ipc::Opt< L4::Ipc::Cap< L4::Task > > dma_task,
    Space_attribs attr)
```

Associate a (kernel) [DMA space](#) for a device to this [Dma_space](#).

Parameters

in	<i>dma_task</i>	The (kernel) DMA space used for the device that shall be associated with this DMA space. In case no IOMMU is present or configured, the <i>dma_task</i> might be an invalid capability when L4Re::Dma_space::Phys_space is set in <i>attr</i> , in this case the CPUs physical memory is used as DMA address space.
in	<i>attr</i>	Attributes for this DMA space. See L4Re::Dma_space::Space_attr .

Return values

<i>L4_EOK</i>	Operation successful.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i>-L4_EINVAL</i>	
<i>-L4_ENOENT</i>	

Precondition

The invoked [Dma_space](#) capability must have the permission [L4_CAP_FPAGE_W](#).

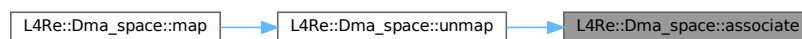
References [disassociate\(\)](#).

Referenced by [unmap\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.286.4.2 disassociate()

```
long L4Re::Dma_space::disassociate ()
```

Disassociate the (kernel) [DMA space](#) from this [Dma_space](#).

Return values

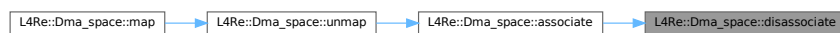
<code>L4_EOK</code>	Operation successful.
<code>-L4_EPERM</code>	Insufficient permissions; see precondition.
<code>-L4_ENOENT</code>	

Precondition

The invoked [Dma_space](#) capability must have the permission [L4_CAP_FPAGE_W](#).

Referenced by [associate\(\)](#).

Here is the caller graph for this function:



16.286.4.3 map()

```

long L4Re::Dma_space::map (
    L4::Ipc::Cap< L4Re::Dataspace > src,
    L4Re::Dataspace::Offset offset,
    L4::Ipc::In_out< l4_size_t * > size,
    Attributes attrs,
    Direction dir,
    Dma_addr * dma_addr)

```

Map the given part of this data space into the DMA address space.

Parameters

in	<i>src</i>	Source data space (that describes the memory). Caller needs write right to the data space.
in	<i>offset</i>	The offset (bytes) within <i>src</i> .
in, out	<i>size</i>	The size (bytes) of the region to be mapped for DMA, after successful mapping the size returned is the size mapped for DMA as a single block. This size might be smaller than the original input size, in this case the caller might call map() again with a new offset and the remaining size.
in	<i>attrs</i>	The attributes used for this DMA mapping (a combination of Dma_space::Attribute values).
in	<i>dir</i>	The direction of the DMA transfer issued with this mapping. The same value must later be passed to unmap() .
out	<i>dma_addr</i>	The DMA address to use for DMA with the associated device.

Return values

<i>L4_EOK</i>	Operation successful.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i>-L4_EINVAL</i>	The capability <i>src</i> is invalid or does not refer to a valid dataspace.
<i>-L4_EEXIST</i>	The specified region overlaps an existing mapping.
<i>-L4_ENOMEM</i>	Not enough memory to allocate internal datastructures.
<i>-L4_ERANGE</i>	<i>offset</i> is larger than the size of the dataspace.

Precondition

The capability *src* must have the permission [L4_CAP_FPAGE_W](#).

Note

[associate\(\)](#) must be called prior to mapping memory. Usually this is done implicitly when binding the managed [Dma_space](#) to a DMA domain (see [L4vbus::Vbus::assign_dma_domain\(\)](#)).

References [unmap\(\)](#).

Here is the call graph for this function:



16.286.4.4 unmap()

```

long L4Re::Dma_space::unmap (
    Dma_addr dma_addr,
    l4_size_t size,
    Attributes attrs,
    Direction dir)
  
```

Unmap the given part of this data space from the DMA address space.

Parameters

<i>dma_addr</i>	The DMA address (returned by Dma_space::map()).
<i>size</i>	The size (bytes) of the memory region to unmap.
<i>attrs</i>	The attributes for the unmap (currently none).
<i>dir</i>	The direction of the finished DMA operation.

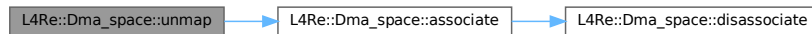
Returns

0 in the case of success, a negative error code otherwise.

References [associate\(\)](#).

Referenced by [map\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

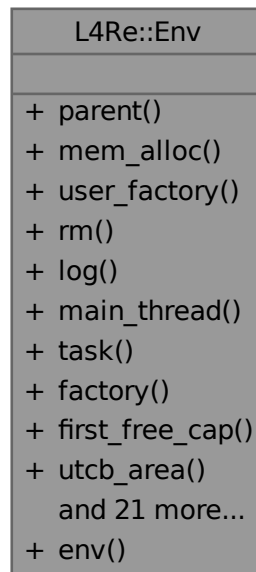
- [l4/re/dma_space](#)

16.287 L4Re::Env Class Reference

C++ interface of the initial environment that is provided to an [L4](#) task.

```
#include <env>
```

Collaboration diagram for L4Re::Env:



Public Types

- typedef [l4re_env_cap_entry_t](#) **Cap_entry**
C++ type for an entry in the initial objects array.

Public Member Functions

- [L4::Cap](#)< [Parent](#) > [parent](#) () const noexcept
Object-capability to the parent.
- [L4::Cap](#)< [Mem_alloc](#) > [mem_alloc](#) () const noexcept
Object-capability to the memory allocator.
- [L4::Cap](#)< [L4::Factory](#) > [user_factory](#) () const noexcept
Object-capability to the user-level object factory.
- [L4::Cap](#)< [Rm](#) > [rm](#) () const noexcept
Object-capability to the region map.
- [L4::Cap](#)< [Log](#) > [log](#) () const noexcept
Object-capability to the logging service.
- [L4::Cap](#)< [L4::Thread](#) > [main_thread](#) () const noexcept
Object-capability of the first user thread.
- [L4::Cap](#)< [L4::Task](#) > [task](#) () const noexcept
Object-capability of the user task.
- [L4::Cap](#)< [L4::Factory](#) > [factory](#) () const noexcept
Object-capability to the factory object available to the task.
- [l4_cap_idx_t](#) [first_free_cap](#) () const noexcept

- First available capability selector.*

 - `l4_fpage_t utcb_area` () const noexcept

UTCB area of the task.
- `l4_addr_t first_free_utcb` () const noexcept

First free UTCB.
- `Cap_entry` const * `initial_caps` () const noexcept

Get a pointer to the first entry in the initial objects array.
- `Cap_entry` const * `get` (char const *name, unsigned l) const noexcept

Get the `Cap_entry` for the object named name.
- template<typename T>
 - `L4::Cap`< T > `get_cap` (char const *name, unsigned l) const noexcept

Get the capability selector for the object named name.
- template<typename T>
 - `L4::Cap`< T > `get_cap` (char const *name) const noexcept

Get the capability selector for the object named name.
- void `parent` (`L4::Cap`< `Parent` > const &c) noexcept

Set parent object-capability.
- void `mem_alloc` (`L4::Cap`< `Mem_alloc` > const &c) noexcept

Set memory allocator object-capability.
- void `rm` (`L4::Cap`< `Rm` > const &c) noexcept

Set region map object-capability.
- void `log` (`L4::Cap`< `Log` > const &c) noexcept

Set log object-capability.
- void `main_thread` (`L4::Cap`< `L4::Thread` > const &c) noexcept

Set object-capability of first user thread.
- void `factory` (`L4::Cap`< `L4::Factory` > const &c) noexcept

Set factory object-capability.
- void `first_free_cap` (`l4_cap_idx_t` c) noexcept

Set first available capability selector.
- void `utcb_area` (`l4_fpage_t` utcb) noexcept

Set UTCB area of the task.
- void `first_free_utcb` (`l4_addr_t` u) noexcept

Set first free UTCB.
- `L4::Cap`< `L4::Scheduler` > `scheduler` () const noexcept

Get the scheduler capability for the task.
- void `scheduler` (`L4::Cap`< `L4::Scheduler` > const &c) noexcept

Set the scheduler capability.
- `L4::Cap`< `Itas` > `itas` () const noexcept

Object-capability to the ITAS services.
- void `itas` (`L4::Cap`< `Itas` > const &c) noexcept

Set the ITAS capability.
- `L4::Cap`< `Dbg_events` > `dbg_events` () const noexcept

Object-capability to a debugger events service.
- void `dbg_events` (`L4::Cap`< `Dbg_events` > const &dbg_events) noexcept

Set the dbg_events capability.
- void `initial_caps` (`Cap_entry` *first) noexcept

Set the pointer to the first `Cap_entry` in the initial objects array.

Static Public Member Functions

- static `Env` const * `env` () noexcept

Returns the initial environment for the current task.

16.287.1 Detailed Description

C++ interface of the initial environment that is provided to an [L4](#) task.

The initial environment is provided to each [L4](#) task that is started by an [L4Re](#) conform loader, such as the Moe root task. The initial environment provides access to a set of initial capabilities and some additional information about the available resources, such as free UTCBs (see [Virtual Registers](#)) and available entries in capability table (provided by the micro kernel).

Each of the initial capabilities is stored at a fixed index in the task's capability table and the [L4](#) runtime environment provides convenience functions to retrieve the capabilities. See the table below for an comprehensive overview.

Name	Object Type	Convenience Function
parent	L4Re::Parent	L4Re::Env::parent()
user_factory	L4::Factory	L4Re::Env::user_factory()
log	L4Re::Log	L4Re::Env::log()
main_thread	L4::Thread	L4Re::Env::main_thread()
rm	L4Re::Rm	L4Re::Env::rm()
factory	L4::Factory	L4Re::Env::factory()
task	L4::Task	L4Re::Env::task()
scheduler	L4::Scheduler	L4Re::Env::scheduler()
itas	L4Re::Itas	L4Re::Env::itas()

Additional information found in the initial environment is:

- First free entry in capability table
- The [UTCB](#) area (as flexpage)
- First free UTCB (address in the UTCB area)

Include File

```
#include <l4/re/env>
```

For an explanation of the default task capabilities see [l4_default_caps_t](#).

For the C interface refer to [Initial Environment](#).

Definition at line [78](#) of file [env](#).

16.287.2 Member Function Documentation

16.287.2.1 dbg_events() [1/2]

```
L4::Cap< Dbg_events > L4Re::Env::dbg_events () const [inline], [noexcept]
```

Object-capability to a debugger events service.

Returns

Dbg_events object-capability

This capability can be invalid.

Definition at line 310 of file [env](#).

Referenced by [dbg_events\(\)](#).

Here is the caller graph for this function:

**16.287.2.2 dbg_events() [2/2]**

```
void L4Re::Env::dbg_events (
    L4::Cap< Dbg_events > const & dbg_events) [inline], [noexcept]
```

Set the dbg_events capability.

Parameters

<i>dbg_events</i>	is the capability to be set for the debug events service.
-------------------	---

Note that the capability can be invalid.

Definition at line 320 of file [env](#).

References [dbg_events\(\)](#).

Here is the call graph for this function:



16.287.2.3 env()

```
Env const * L4Re::Env::env () [inline], [static], [noexcept]
```

Returns the initial environment for the current task.

Returns

Pointer to the initial environment class.

A typical use of this function is `L4Re::Env::env()-><member>()`

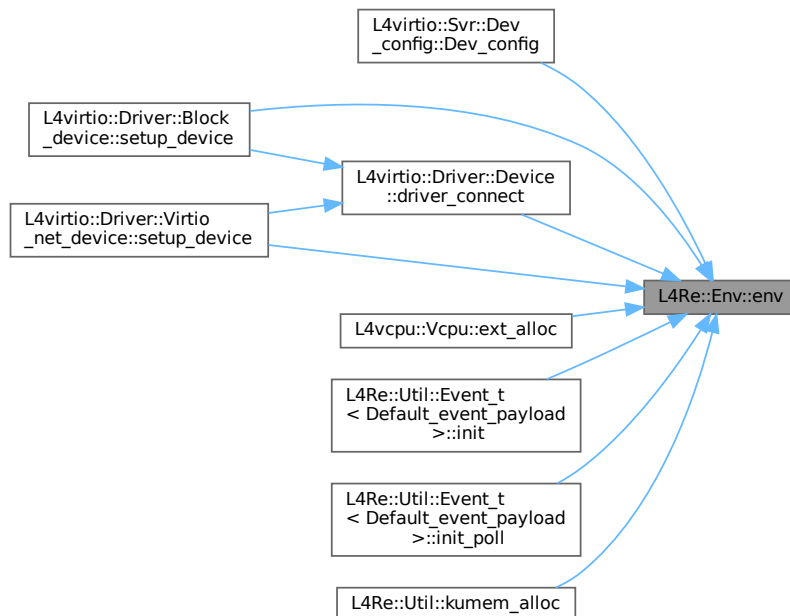
Examples

`examples/clntsrv/src/client.cc`, `examples/libs/l4re/c++/mem_alloc/ma+rm.cc`, `examples/libs/l4re/c++/shared_ds/ds_clnt.cc`, `examples/libs/l4re/c++/shared_ds/ds_srv.cc`, `examples/libs/l4re/streammap/client.cc`, and `examples/sys/migrate/thread_migrate.cc`

Definition at line 96 of file `env`.

Referenced by `L4virtio::Svr::Dev_config::Dev_config()`, `L4virtio::Driver::Device::driver_connect()`, `L4vcpu::Vcpu::ext_alloc()`, `L4Re::Util::Event_t< Default_event_payload >::init()`, `L4Re::Util::Event_t< Default_event_payload >::init_poll()`, `L4Re::Util::kumem_alloc()`, `L4virtio::Driver::Block_device::setup_device()`, and `L4virtio::Driver::Virtio_net_device::setup_device()`.

Here is the caller graph for this function:



16.287.2.4 factory() [1/2]

```
L4::Cap< L4::Factory > L4Re::Env::factory () const [inline], [noexcept]
```

Object-capability to the factory object available to the task.

Returns

Factory object-capability

Definition at line 144 of file [env](#).

16.287.2.5 factory() [2/2]

```
void L4Re::Env::factory (
    L4::Cap< L4::Factory > const & c) [inline], [noexcept]
```

Set factory object-capability.

Parameters

<i>c</i>	Factory object-capability
----------	---------------------------

Definition at line 249 of file [env](#).

16.287.2.6 first_free_cap() [1/2]

```
l4_cap_idx_t L4Re::Env::first_free_cap () const [inline], [noexcept]
```

First available capability selector.

Returns

First capability selector.

First capability selector available for use for in the application.

Definition at line 152 of file [env](#).

16.287.2.7 first_free_cap() [2/2]

```
void L4Re::Env::first_free_cap (
    l4_cap_idx_t c) [inline], [noexcept]
```

Set first available capability selector.

Parameters

<i>c</i>	First capability selector available to the application.
----------	---

Definition at line 255 of file [env](#).

16.287.2.8 first_free_utcb() [1/2]

```
l4_addr_t L4Re::Env::first_free_utcb () const [inline], [noexcept]
```

First free UTCB.

Returns

object-capability

First free UTCB within the UTCB area available for the application to use.

Definition at line 167 of file [env](#).

16.287.2.9 first_free_utcb() [2/2]

```
void L4Re::Env::first_free_utcb (
    l4_addr_t u) [inline], [noexcept]
```

Set first free UTCB.

Parameters

<i>u</i>	First UTCB available for the application to use.
----------	--

Definition at line 267 of file [env](#).

16.287.2.10 get()

```
Cap_entry const * L4Re::Env::get (
    char const * name,
    unsigned l) const [inline], [noexcept]
```

Get the [Cap_entry](#) for the object named *name*.

Parameters

<i>name</i>	is the name of the object.
<i>l</i>	is the length of the name, thus <i>name</i> might not be zero terminated.

Returns

A pointer to the [Cap_entry](#) for the object named *name*, or NULL if no such object was found.

Definition at line 185 of file [env](#).

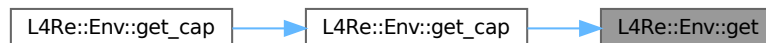
References [l4re_env_get_cap_l\(\)](#).

Referenced by [get_cap\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**16.287.2.11 get_cap() [1/2]**

```
template<typename T>
L4::Cap< T > L4Re::Env::get_cap (
    char const * name) const [inline], [noexcept]
```

Get the capability selector for the object named *name*.

Parameters

<i>name</i>	is the name of the object (zero terminated).
-------------	--

Returns

A capability selector for the object named *name*, or an invalid capability selector if no such object was found.

Definition at line 212 of file [env](#).

References [get_cap\(\)](#).

Here is the call graph for this function:



16.287.2.12 `get_cap()` [2/2]

```
template<typename T>
L4::Cap< T > L4Re::Env::get_cap (
    char const * name,
    unsigned l) const [inline], [noexcept]
```

Get the capability selector for the object named *name*.

Parameters

<i>name</i>	is the name of the object.
<i>l</i>	is the length of the name, thus <i>name</i> might not be zero terminated.

Returns

A capability selector for the object named *name*, or an invalid capability selector if no such object was found.

Examples

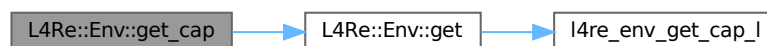
[examples/clntsrv/src/client.cc](#), [examples/libs/l4re/c++/shared_ds/ds_clnt.cc](#), and [examples/libs/l4re/streammap/client.cc](#).

Definition at line 197 of file [env](#).

References [get\(\)](#), and [L4_ENOENT](#).

Referenced by [get_cap\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.287.2.13 initial_caps() [1/2]

```
Cap_entry const * L4Re::Env::initial_caps () const [inline], [noexcept]
```

Get a pointer to the first entry in the initial objects array.

Returns

A pointer to the first entry in the initial objects array.

Definition at line 174 of file [env](#).

16.287.2.14 initial_caps() [2/2]

```
void L4Re::Env::initial_caps (
    Cap_entry * first) [inline], [noexcept]
```

Set the pointer to the first [Cap_entry](#) in the initial objects array.

Parameters

<i>first</i>	is the first element in the array.
--------------	------------------------------------

Definition at line 327 of file [env](#).

16.287.2.15 itas() [1/2]

```
L4::Cap< Itas > L4Re::Env::itas () const [inline], [noexcept]
```

Object-capability to the ITAS services.

Returns

ITAS object-capability

Attention: this capability might be invalid, depending on the system configuration. Regular applications must not use it directly as it is an implementation detail of the [L4Re](#) libc that is subject to change without notice!

Definition at line 294 of file [env](#).

16.287.2.16 itas() [2/2]

```
void L4Re::Env::itas (
    L4::Cap< Itas > const & c) [inline], [noexcept]
```

Set the ITAS capability.

Parameters

<code>c</code>	is the capability to be set as ITAS.
----------------	--------------------------------------

Definition at line 301 of file [env](#).

16.287.2.17 `log()` [1/2]

```
L4::Cap< Log > L4Re::Env::log () const [inline], [noexcept]
```

Object-capability to the logging service.

Returns

[Log](#) object-capability

Definition at line 126 of file [env](#).

16.287.2.18 `log()` [2/2]

```
void L4Re::Env::log (
    L4::Cap< Log > const & c) [inline], [noexcept]
```

Set log object-capability.

Parameters

<code>c</code>	Log object-capability
----------------	---------------------------------------

Definition at line 237 of file [env](#).

16.287.2.19 `main_thread()` [1/2]

```
L4::Cap< L4::Thread > L4Re::Env::main_thread () const [inline], [noexcept]
```

Object-capability of the first user thread.

Returns

Object-capability of the first user thread.

Definition at line 132 of file [env](#).

16.287.2.20 `main_thread()` [2/2]

```
void L4Re::Env::main_thread (
    L4::Cap< L4::Thread > const & c) [inline], [noexcept]
```

Set object-capability of first user thread.

Parameters

<code>c</code>	First thread's object-capability
----------------	----------------------------------

Definition at line 243 of file [env](#).

16.287.2.21 `mem_alloc()` [1/2]

```
L4::Cap< Mem_alloc > L4Re::Env::mem_alloc () const [inline], [noexcept]
```

Object-capability to the memory allocator.

Returns

Memory allocator object-capability

Examples

[examples/libs/l4re/c++/shared_ds/ds_srv.cc](#).

Definition at line 109 of file [env](#).

16.287.2.22 `mem_alloc()` [2/2]

```
void L4Re::Env::mem_alloc (
    L4::Cap< Mem_alloc > const & c) [inline], [noexcept]
```

Set memory allocator object-capability.

Parameters

<code>c</code>	Memory allocator object-capability
----------------	------------------------------------

Definition at line 225 of file [env](#).

16.287.2.23 `parent()` [1/2]

```
L4::Cap< Parent > L4Re::Env::parent () const [inline], [noexcept]
```

Object-capability to the parent.

Returns

[Parent](#) object-capability

Definition at line 103 of file [env](#).

16.287.2.24 `parent()` [2/2]

```
void L4Re::Env::parent (
    L4::Cap< Parent > const & c) [inline], [noexcept]
```

Set parent object-capability.

Parameters

c	Parent object-capability
---	--------------------------

Definition at line 219 of file [env](#).

16.287.2.25 `rm()` [1/2]

```
L4::Cap< Rm > L4Re::Env::rm () const [inline], [noexcept]
```

Object-capability to the region map.

Returns

Region map object-capability

Examples

[examples/libs/l4re/c++/shared_ds/ds_clnt.cc](#), and [examples/libs/l4re/c++/shared_ds/ds_srv.cc](#).

Definition at line 120 of file [env](#).

16.287.2.26 `rm()` [2/2]

```
void L4Re::Env::rm (
    L4::Cap< Rm > const & c) [inline], [noexcept]
```

Set region map object-capability.

Parameters

c	Region map object-capability
---	------------------------------

Definition at line 231 of file [env](#).

16.287.2.27 `scheduler()` [1/2]

```
L4::Cap< L4::Scheduler > L4Re::Env::scheduler () const [inline], [noexcept]
```

Get the scheduler capability for the task.

Returns

The capability selector for the default scheduler used for this task.

Definition at line 275 of file [env](#).

16.287.2.28 `scheduler()` [2/2]

```
void L4Re::Env::scheduler (
    L4::Cap< L4::Scheduler > const & c) [inline], [noexcept]
```

Set the scheduler capability.

Parameters

<code>c</code>	is the capability to be set as scheduler.
----------------	---

Definition at line 282 of file [env](#).

16.287.2.29 task()

```
L4::Cap< L4::Task > L4Re::Env::task () const [inline], [noexcept]
```

Object-capability of the user task.

Returns

Object-capability of the user task.

Definition at line 138 of file [env](#).

16.287.2.30 utcb_area() [1/2]

```
l4_fpage_t L4Re::Env::utcb_area () const [inline], [noexcept]
```

UTCB area of the task.

Returns

UTCB area

Definition at line 158 of file [env](#).

16.287.2.31 utcb_area() [2/2]

```
void L4Re::Env::utcb_area (
    l4_fpage_t utcbs) [inline], [noexcept]
```

Set UTCB area of the task.

Parameters

<code>utcbs</code>	UTCB area
--------------------	-----------

Definition at line 261 of file [env](#).

The documentation for this class was generated from the following file:

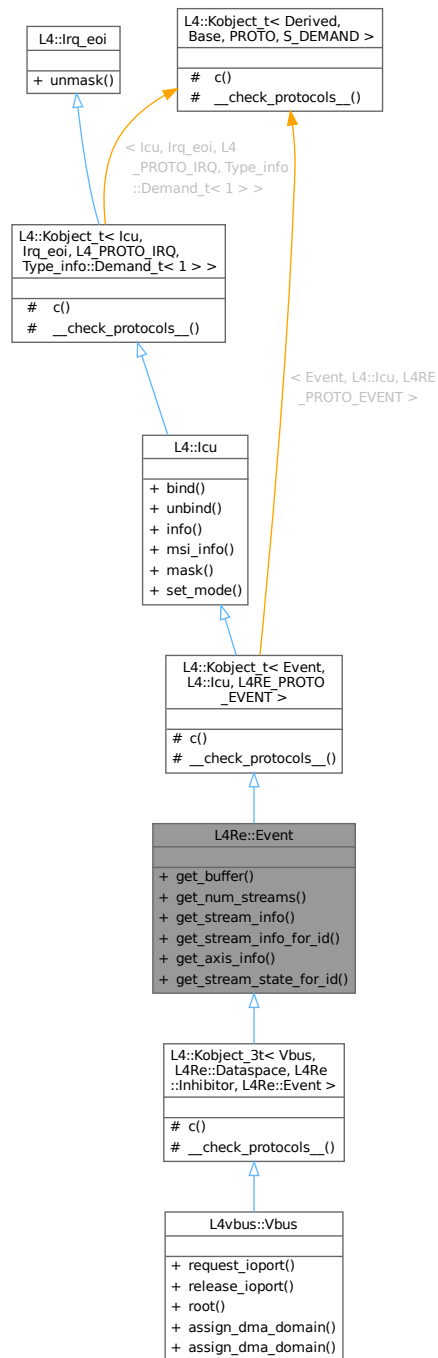
- [l4/re/env](#)

16.288 L4Re::Event Class Reference

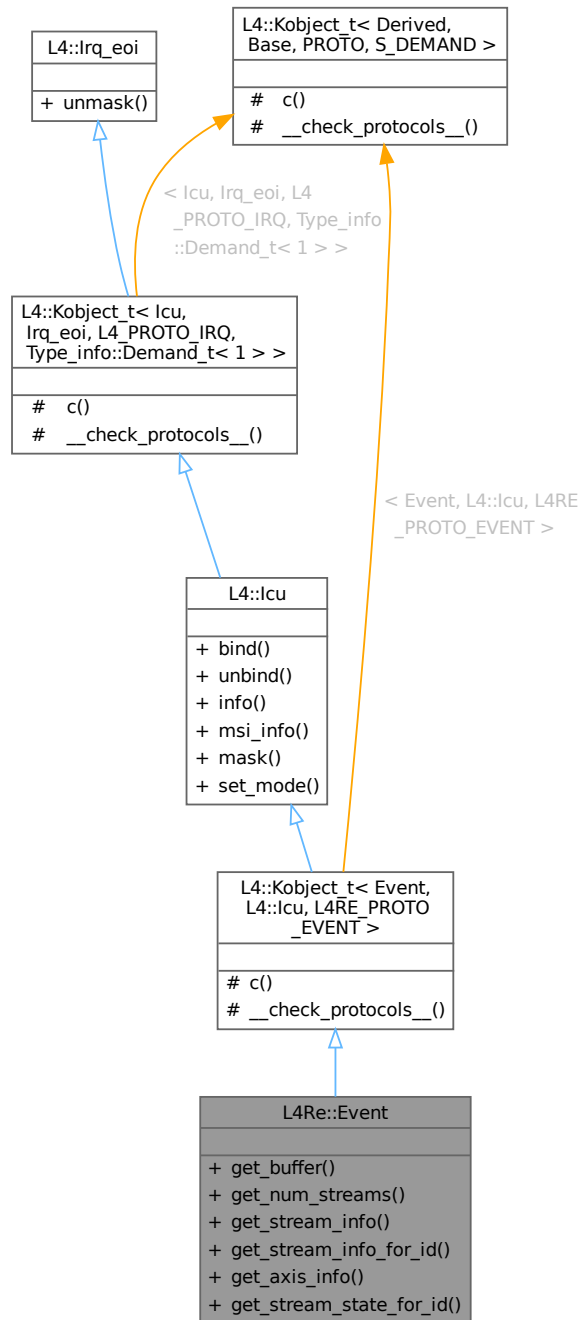
Event class.

```
#include <event>
```

Inheritance diagram for L4Re::Event:



Collaboration diagram for L4Re::Event:



Public Member Functions

- long `get_buffer` (`L4::lpc::Out< L4::Cap< Dataspace > > ds`)
Get event signal buffer.
- long `get_num_streams` ()
Get number of event streams.
- long `get_stream_info` (int idx, Event_stream_info *info)

Get event stream infos.

- long [get_stream_info_for_id](#) ([l4_umword_t](#) stream_id, [Event_stream_info](#) *info)

Get event stream infos.

- long [get_axis_info](#) ([l4_umword_t](#) stream_id, unsigned naxes, unsigned const *axis, [Event_absinfo](#) *info) const noexcept

Get event stream axis infos.

- long [get_stream_state_for_id](#) ([l4_umword_t](#) stream_id, [Event_stream_state](#) *state)

Get event stream state.

Public Member Functions inherited from [L4::lcu](#)

- [l4_msgtag_t](#) bind (unsigned irqnum, [L4::Cap](#)< [Triggerable](#) > irq, [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept

Bind an interrupt line of an interrupt controller to an interrupt object.

- [l4_msgtag_t](#) unbind (unsigned irqnum, [L4::Cap](#)< [Triggerable](#) > irq, [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept

Remove binding of an interrupt line from the interrupt controller object.

- [l4_msgtag_t](#) info ([l4_icu_info_t](#) *info, [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept

Get information about the ICU features.

- [l4_msgtag_t](#) msi_info ([l4_umword_t](#) irqnum, [l4_uint64_t](#) source, [l4_icu_msi_info_t](#) *msi_info)

Get MSI info about IRQ.

- [l4_msgtag_t](#) mask (unsigned irqnum, [l4_umword_t](#) *label=0, [l4_timeout_t](#) to=[L4_IPC_NEVER](#), [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept

Mask an IRQ line.

- [l4_msgtag_t](#) set_mode (unsigned irqnum, [l4_umword_t](#) mode, [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept

Set interrupt mode.

Public Member Functions inherited from [L4::lrq_eoi](#)

- [l4_msgtag_t](#) unmask (unsigned irqnum, [l4_umword_t](#) *label=0, [l4_timeout_t](#) to=[L4_IPC_NEVER](#), [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept

Unmask the given interrupt line.

Additional Inherited Members

Protected Types inherited from [L4::Kobject_t](#)< [Event](#), [L4::lcu](#), [L4RE_PROTO_EVENT](#) >

- typedef [Event](#) **Class**

The target interface type (inheriting from [Kobject_t](#)).

- typedef [Typeid::Iface](#)< [PROTO](#), [Event](#) > **__Iface**

The interface description for the derived class.

- typedef [Typeid::Merge_list](#)< [Typeid::Iface_list](#)< **__Iface** >, typename [L4::lcu::__Iface_list](#) > **__Iface_list**

The list of all RPC interfaces provided directly or through inheritance.

Protected Types inherited from

[L4::Kobject_t](#)< [lcu](#), [lrq_eoi](#), [L4_PROTO_IRQ](#), [Type_info::Demand_t](#)< 1 > >

- typedef [lcu](#) **Class**

The target interface type (inheriting from [Kobject_t](#)).

- typedef [Typeid::Iface](#)< [PROTO](#), [lcu](#) > **__Iface**

The interface description for the derived class.

- typedef [Typeid::Merge_list](#)< [Typeid::Iface_list](#)< **__Iface** >, typename [lrq_eoi::__Iface_list](#) > **__Iface_list**

The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from [L4::Kobject_t< Event, L4::lcu, L4RE_PROTO_EVENT >](#)

- [L4::Cap< Class > c \(\)](#) const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from [L4::Kobject_t< lcu, Irq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > >](#)

- [L4::Cap< Class > c \(\)](#) const noexcept
Get the capability to ourselves.

Static Protected Member Functions inherited from [L4::Kobject_t< Event, L4::lcu, L4RE_PROTO_EVENT >](#)

- static void [__check_protocols__ \(\)](#) noexcept
Helper to check for protocol conflicts.

Static Protected Member Functions inherited from [L4::Kobject_t< lcu, Irq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > >](#)

- static void [__check_protocols__ \(\)](#) noexcept
Helper to check for protocol conflicts.

16.288.1 Detailed Description

[Event](#) class.

See also

[L4Re Event API](#)

Definition at line 137 of file [event](#).

16.288.2 Member Function Documentation

16.288.2.1 [get_axis_info\(\)](#)

```
long L4Re::Event::get_axis_info (
    l4_umword_t stream_id,
    unsigned naxes,
    unsigned const * axis,
    Event_absinfo * info) const [inline], [noexcept]
```

Get event stream axis infos.

Parameters

	<i>stream</i> ↔ <i>_id</i>	ID of the event stream.
in	<i>naxes</i>	Number of axis IDs and axis infos.
in	<i>axis</i>	Array of axis IDs.
out	<i>info</i>	Array of axis infos.

Return values

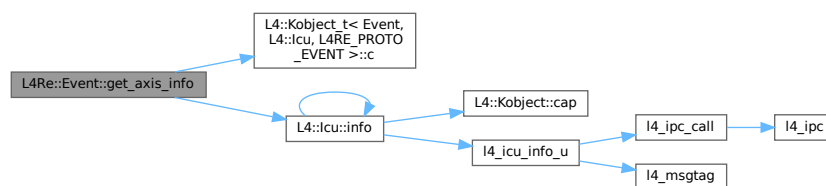
≥ 0	Number of returned axes infos.
< 0	Error code.

Definition at line 198 of file [event](#).

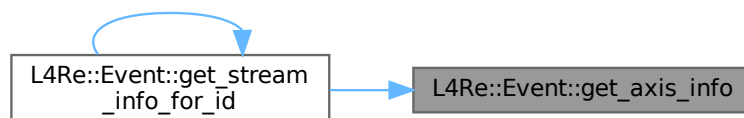
References [L4::Kobject_t< Event, L4::lcu, L4RE_PROTO_EVENT >::c\(\)](#), and [L4::lcu::info\(\)](#).

Referenced by [get_stream_info_for_id\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.288.2.2 get_buffer()

```

long L4Re::Event::get_buffer (
    L4::Ipc::Out< L4::Cap< Dataspace > > ds)

```

Get event signal buffer.

Parameters

out	ds	Event buffer.
-----	----	---------------

Return values

0	Success
<0	Error

References [get_buffer\(\)](#).

Referenced by [get_buffer\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**16.288.2.3 get_num_streams()**

```
long L4Re::Event::get_num_streams ()
```

Get number of event streams.

Return values

≥ 0	Number of streams.
<0	Error code.

References [get_num_streams\(\)](#).

Referenced by [get_num_streams\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.288.2.4 `get_stream_info()`

```
long L4Re::Event::get_stream_info (  
    int idx,  
    Event_stream_info * info)
```

Get event stream infos.

Deprecated. Use [get_stream_info_for_id\(\)](#).

Parameters

	<i>idx</i>	ID of the event stream.
out	<i>info</i>	Event stream info.

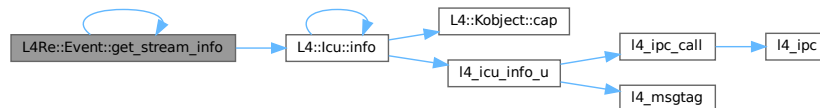
Return values

0	Success
<0	Error

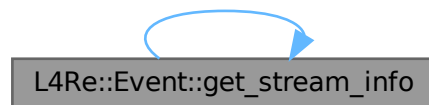
References [get_stream_info\(\)](#), and [L4::Icu::info\(\)](#).

Referenced by [get_stream_info\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.288.2.5 get_stream_info_for_id()

```

long L4Re::Event::get_stream_info_for_id (
    l4_umword_t stream_id,
    Event_stream_info * info)

```

Get event stream infos.

Parameters

	<i>stream_id</i>	ID of the event stream.
out	<i>info</i>	Event stream info.

Return values

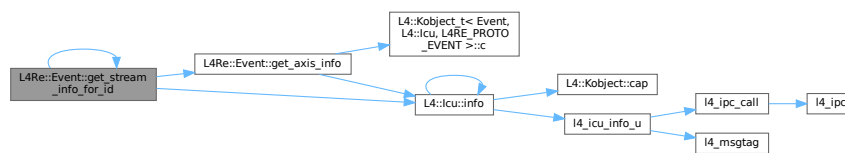
0	Success
---	---------

<0	Error
----	-------

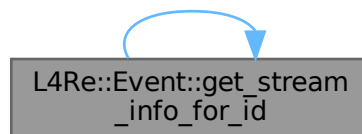
References [get_axis_info\(\)](#), [get_stream_info_for_id\(\)](#), and [L4::lcu::info\(\)](#).

Referenced by [get_stream_info_for_id\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.288.2.6 `get_stream_state_for_id()`

```

long L4Re::Event::get_stream_state_for_id (
    l4_umword_t stream_id,
    Event_stream_state * state)

```

Get event stream state.

Parameters

	<i>stream_id</i>	ID of the event stream.
out	<i>state</i>	Event stream state.

Return values

0	Success
---	---------

<0	Error
----	-------

The documentation for this class was generated from the following file:

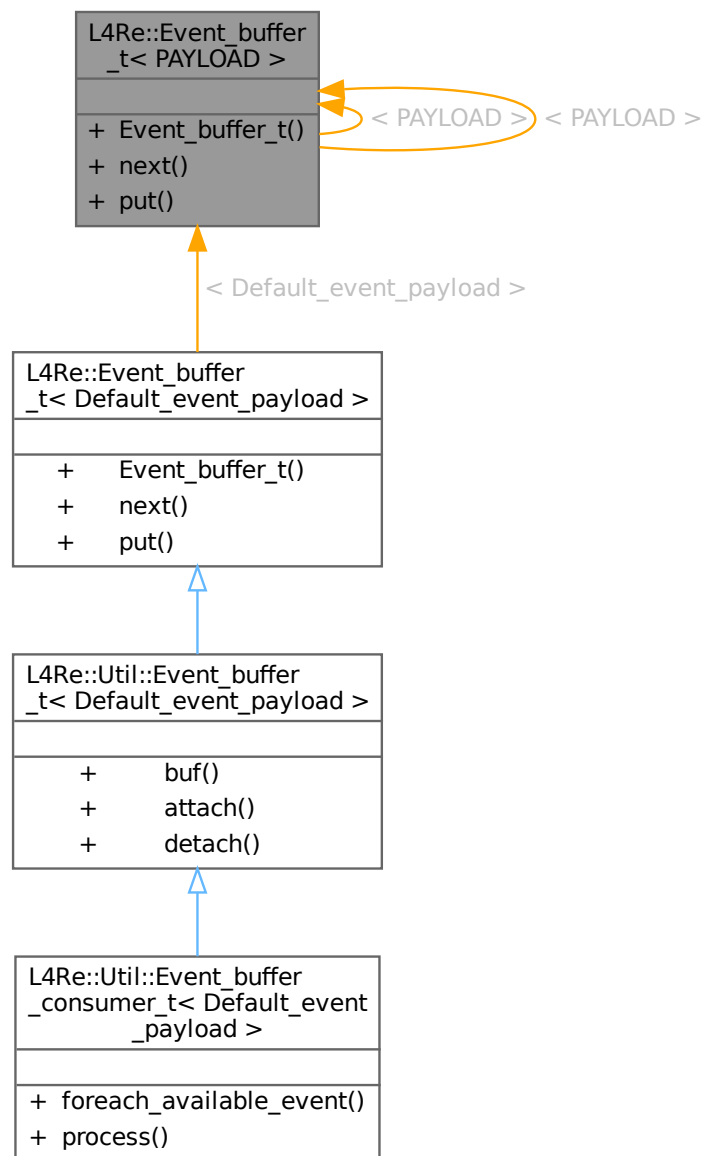
- l4/re/event

16.289 L4Re::Event_buffer_t< PAYLOAD > Class Template Reference

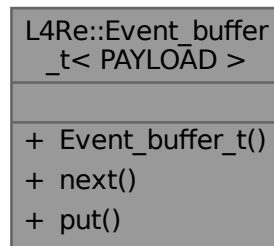
[Event](#) buffer class.

```
#include <event>
```

Inheritance diagram for L4Re::Event_buffer_t< PAYLOAD >:



Collaboration diagram for L4Re::Event_buffer_t< PAYLOAD >:



Data Structures

- struct [Event](#)
Event structure used in buffer.

Public Member Functions

- [Event_buffer_t](#) (void *buffer, [l4_addr_t](#) size)
Initialize event buffer.
- [Event](#) * [next](#) () noexcept
Next event in buffer.
- bool [put](#) ([Event](#) const &ev) noexcept
Put event into buffer at current position.

16.289.1 Detailed Description

```
template<typename PAYLOAD = Default_event_payload>
class L4Re::Event_buffer_t< PAYLOAD >
```

[Event](#) buffer class.

Definition at line [246](#) of file [event](#).

16.289.2 Constructor & Destructor Documentation

16.289.2.1 Event_buffer_t()

```
template<typename PAYLOAD = Default_event_payload>
L4Re::Event_buffer_t< PAYLOAD >::Event_buffer_t (
    void * buffer,
    l4_addr_t size) [inline]
```

Initialize event buffer.

Parameters

<i>buffer</i>	Pointer to buffer.
<i>size</i>	Size of buffer in bytes.

Definition at line 293 of file [event](#).

16.289.3 Member Function Documentation

16.289.3.1 next()

```
template<typename PAYLOAD = Default_event_payload>
Event * L4Re::Event_buffer_t< PAYLOAD >::next () [inline], [noexcept]
```

Next event in buffer.

Returns

0 if no event available, event otherwise.

Definition at line 303 of file [event](#).

16.289.3.2 put()

```
template<typename PAYLOAD = Default_event_payload>
bool L4Re::Event_buffer_t< PAYLOAD >::put (
    Event const & ev) [inline], [noexcept]
```

Put event into buffer at current position.

Parameters

<i>ev</i>	Event to put into the buffer.
-----------	---

Returns

false if buffer is full and entry could not be added.

Definition at line 320 of file [event](#).

The documentation for this class was generated from the following file:

- [l4/re/event](#)

16.290 L4Re::Event_buffer_t< PAYLOAD >::Event Struct Reference

[Event](#) structure used in buffer.

```
#include <event>
```

Collaboration diagram for L4Re::Event_buffer_t< PAYLOAD >::Event:

L4Re::Event_buffer _t< PAYLOAD >::Event	
+	time
+	free()

Public Member Functions

- void **free** () noexcept
Free the entry.

Data Fields

- long long **time**
[Event](#) time stamp.

16.290.1 Detailed Description

```
template<typename PAYLOAD = Default_event_payload>  
struct L4Re::Event_buffer_t< PAYLOAD >::Event
```

[Event](#) structure used in buffer.

Definition at line 253 of file [event](#).

The documentation for this struct was generated from the following file:

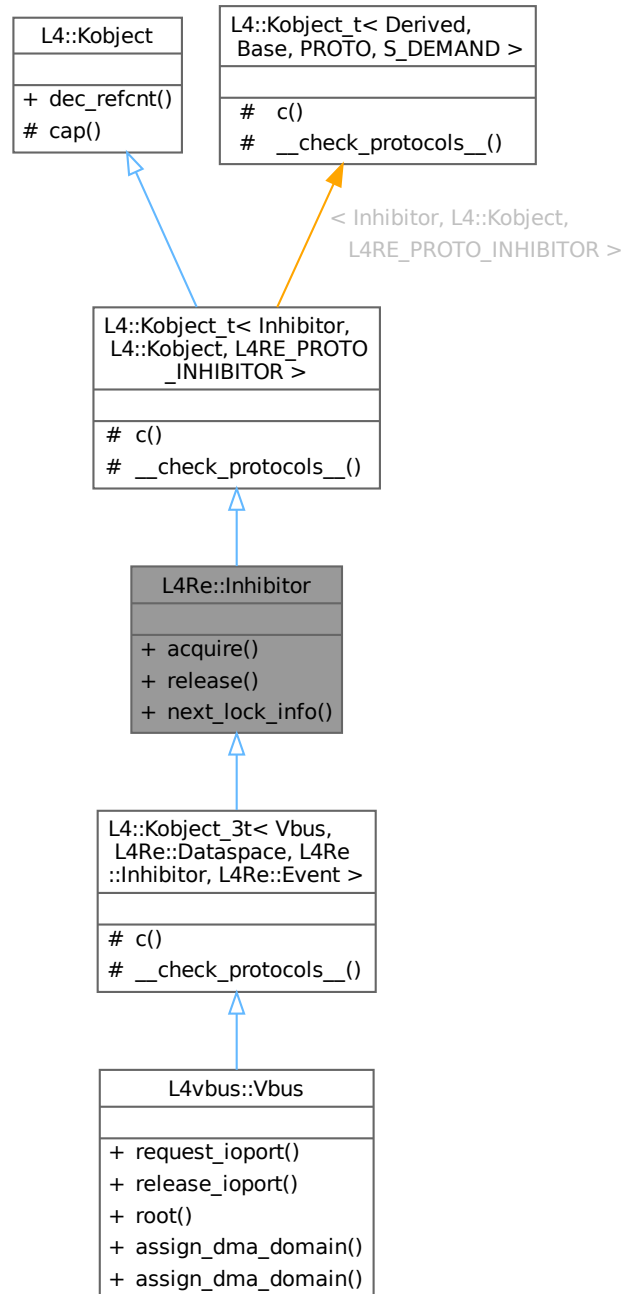
- l4/re/event

16.291 L4Re::Inhibitor Class Reference

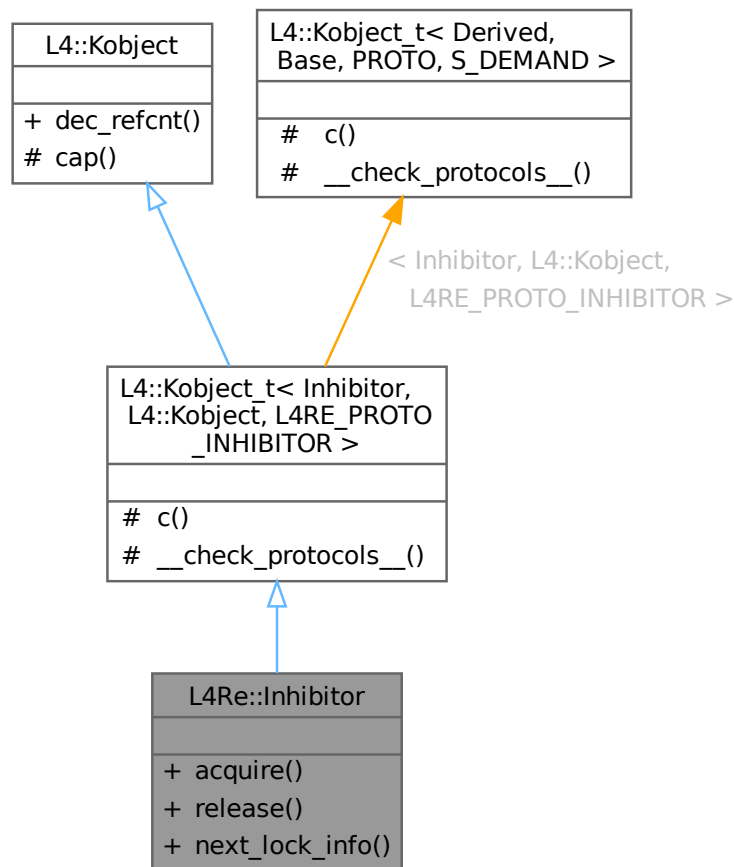
Set of inhibitor locks, which inhibit specific actions when held.

```
#include <inhibitor>
```

Inheritance diagram for L4Re::Inhibitor:



Collaboration diagram for L4Re::Inhibitor:



Public Types

- enum { `Name_max` = 20 }

Public Member Functions

- long `acquire` (`l4_umword_t` id, `L4::lpc::String<>` reason)
Acquire a specific inhibitor lock.
- long `release` (`l4_umword_t` id)
Release a specific inhibitor lock.
- long `next_lock_info` (char *name, unsigned len, `l4_mword_t` current_id=-1, `l4_utcb_t` *utcb=`l4_utcb()`)
Get information for the next available inhibitor lock.

Public Member Functions inherited from L4::Kobject

- `l4_msgtag_t` `dec_refcnt` (`l4_mword_t` diff, `l4_utcb_t` *utcb=`l4_utcb()`)
Decrement the in kernel reference counter for the object.

Additional Inherited Members

Protected Types inherited from

[L4::Kobject_t< Inhibitor, L4::Kobject, L4RE_PROTO_INHIBITOR >](#)

- typedef Inhibitor **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< PROTO, Inhibitor > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__Iface** >, typename L4::Kobject::__Iface_list > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from

[L4::Kobject_t< Inhibitor, L4::Kobject, L4RE_PROTO_INHIBITOR >](#)

- [L4::Cap< Class > c](#) () const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from [L4::Kobject](#)

- [l4_cap_idx_t cap](#) () const noexcept
Return capability selector.

Static Protected Member Functions inherited from

[L4::Kobject_t< Inhibitor, L4::Kobject, L4RE_PROTO_INHIBITOR >](#)

- static void **__check_protocols** () noexcept
Helper to check for protocol conflicts.

16.291.1 Detailed Description

Set of inhibitor locks, which inhibit specific actions when held.

This interface provides access to a set of inhibitor locks, each determined by an ID that is specific to the [Inhibitor](#) object. Each individual lock shall prevent, a specific (implementation defined) action to be executed, as long as the lock is held.

For example there can be an inhibitor lock to prevent a transition to suspend-to-RAM state and a different one to prevent shutdown.

A client shall take an inhibitor lock if it needs to execute code before the action is taken. For example a lock-screen application shall grab an inhibitor lock for the suspend action to be able to lock the screen before the system goes to sleep.

[Inhibitor](#) locks are usually closely related to specific events. Usually a server automatically subscribes a client holding a lock to the corresponding event. The server shall send the event to inform the client that an action is pending. Upon reception of the event, the client is supposed to release the corresponding inhibitor lock.

Definition at line 38 of file [inhibitor](#).

16.291.2 Member Enumeration Documentation

16.291.2.1 anonymous enum

anonymous enum

Enumerator

Name_max	The maximum length of a lock's name.
----------	--------------------------------------

Definition at line 42 of file [inhibitor](#).

16.291.3 Member Function Documentation

16.291.3.1 acquire()

```
long L4Re::Inhibitor::acquire (
    l4_umword_t id,
    L4::Ipc::String<> reason)
```

Acquire a specific inhibitor lock.

Parameters

<i>id</i>	ID of the inhibitor lock that the client intends to acquire
<i>reason</i>	The reason why you need the lock. Used for informing the user or debugging.

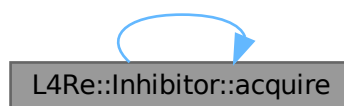
Return values

0	Success
-L4_ENODEV	The specified <i>id</i> does not exist.

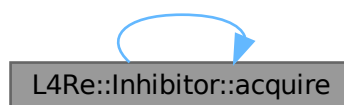
References [acquire\(\)](#).

Referenced by [acquire\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.291.3.2 next_lock_info()

```
long L4Re::Inhibitor::next_lock_info (
    char * name,
    unsigned len,
    l4_mword_t current_id = -1,
    l4_utcb_t * utcb = l4_utcb()) [inline]
```

Get information for the next available inhibitor lock.

Parameters

<i>name</i>	A pointer to a buffer for the name of the lock.
<i>len</i>	The length of the available buffer (usually Name_max is used).
<i>current_id</i>	The ID of the last available lock, use -1 to get the first lock.
<i>utcb</i>	The UTCB to use for the message.

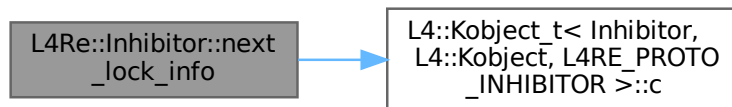
Return values

<i>>0</i>	The ID of the next available lock if there is one (in this case <i>name</i> shall contain the name of the inhibitor lock).
<i>-L4_ENODEV</i>	There are no more locks.

Definition at line 84 of file [inhibitor](#).

References [L4::Kobject_t< Inhibitor, L4::Kobject, L4RE_PROTO_INHIBITOR >::c\(\)](#).

Here is the call graph for this function:



16.291.3.3 release()

```
long L4Re::Inhibitor::release (
    l4_umword_t id)
```

Release a specific inhibitor lock.

Parameters

<i>id</i>	The ID of the inhibitor lock to release.
-----------	--

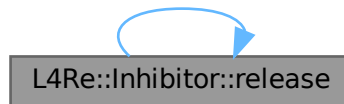
Return values

0	Success
-L4_ENODEV	Lock with the given <i>id</i> does not exist.

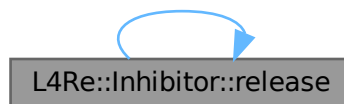
References [release\(\)](#).

Referenced by [release\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

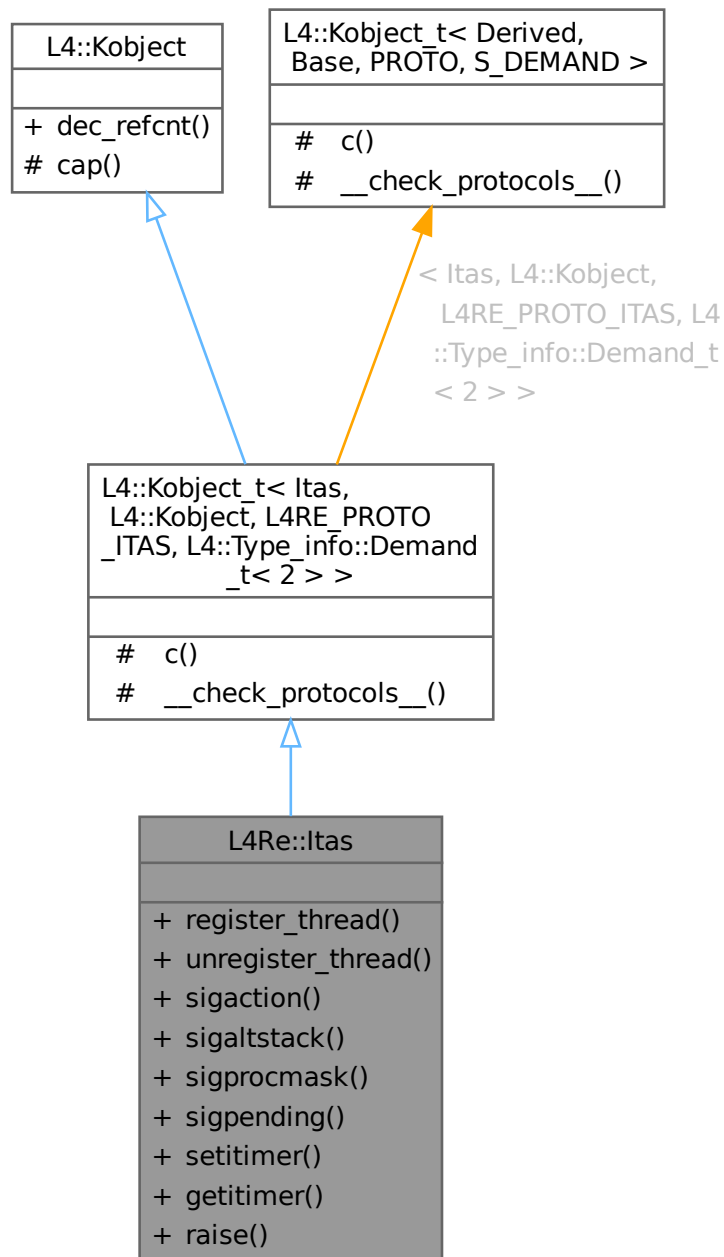
- `I4/re/inhibitor`

16.292 L4Re::Itas Class Reference

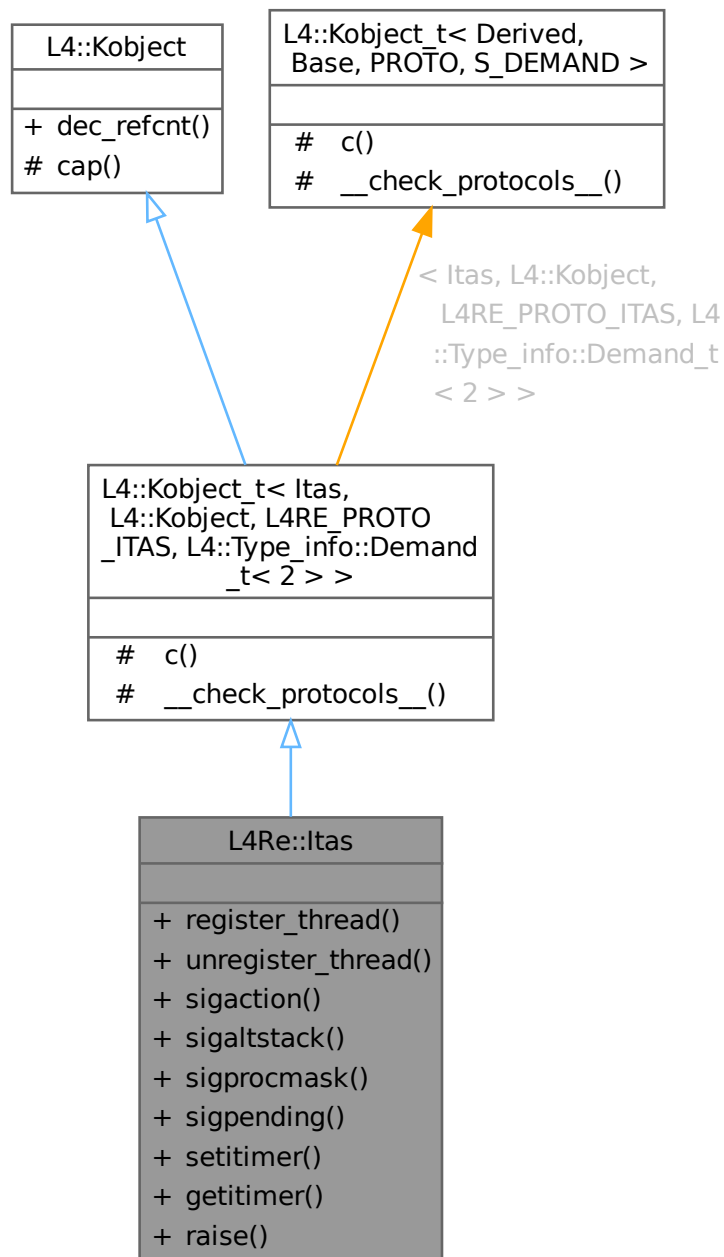
Interface to the ITAS.

```
#include <itas>
```

Inheritance diagram for L4Re::Itas:



Collaboration diagram for L4Re::Itas:



Public Types

- enum : unsigned { `ignore_sigaction` = ~0U }

Public Member Functions

- int `register_thread` (L4::lpc::Cap< L4::Thread > parent, L4::lpc::Cap< L4::Thread > thread_cap, l4_addr_t thread_utcb)

- *Register new thread.*
- `int unregister_thread (L4::lpc::Cap< L4::Thread > thread)`
- *Unregister a thread.*
- `int sigaction (int signum, const struct sigaction *act, struct sigaction *oldact)`
- *Examine and change a POSIX signal action.*
- `int sigaltstack (L4::lpc::Cap< L4::Thread > thread, const struct sigaltstack *ss, struct sigaltstack *oss)`
- *Examine or set alternate POSIX signal stack.*
- `int sigprocmask (L4::lpc::Cap< L4::Thread > thread, int how, sigset_t const *set, sigset_t *oldset)`
- *Examine or set process signal mask.*
- `int sigpending (L4::lpc::Cap< L4::Thread > thread, sigset_t *set)`
- *Query pending signals.*
- `int setitimer (int which, const struct itimerval *new_value, struct itimerval *old_value)`
- *Set process interval timer.*
- `int getitimer (int which, struct itimerval *curr_value)`
- *Get process interval timer.*
- `int raise (L4::lpc::Cap< L4::Thread > thread, int sig)`
- *Send a signal to the calling thread.*

Public Member Functions inherited from L4::Kobject

- `l4_msgtag_t dec_refcnt (l4_mword_t diff, l4_utcb_t *utcb=l4_utcb())`
- *Decrement the in kernel reference counter for the object.*

Additional Inherited Members

Protected Types inherited from

L4::Kobject_t< Itas, L4::Kobject, L4RE_PROTO_ITAS, L4::Type_info::Demand_t< 2 > >

- `typedef Itas Class`
- *The target interface type (inheriting from [Kobject_t](#)).*
- `typedef Typeid::Iface< PROTO, Itas > __Iface`
- *The interface description for the derived class.*
- `typedef Typeid::Merge_list< Typeid::Iface_list< __Iface >, typename L4::Kobject::__Iface_list > __Iface_list`
- *The list of all RPC interfaces provided directly or through inheritance.*

Protected Member Functions inherited from

L4::Kobject_t< Itas, L4::Kobject, L4RE_PROTO_ITAS, L4::Type_info::Demand_t< 2 > >

- `L4::Cap< Class > c () const noexcept`
- *Get the capability to ourselves.*

Protected Member Functions inherited from L4::Kobject

- `l4_cap_idx_t cap () const noexcept`
- *Return capability selector.*

Static Protected Member Functions inherited from

[L4::Kobject_t< Itas, L4::Kobject, L4RE_PROTO_ITAS, L4::Type_info::Demand_t< 2 > >](#)

- static void `__check_protocols__()` noexcept
Helper to check for protocol conflicts.

16.292.1 Detailed Description

Interface to the ITAS.

This is an internal interface between libc and the `l4re_itas`. Do not use it. It is subject to change.

Definition at line 30 of file [itas](#).

16.292.2 Member Enumeration Documentation

16.292.2.1 anonymous enum

anonymous enum : unsigned

Enumerator

Ignore_sigaction	Ignore new action of sigaction() call.
------------------	--

Definition at line 63 of file [itas](#).

16.292.3 Member Function Documentation

16.292.3.1 getitimer()

```
int L4Re::Itas::getitimer (
    int which,
    struct itimerval * curr_value)
```

Get process interval timer.

See IEEE Std 1003.1-2017 [getitimer\(\)](#) for details.

Parameters

in	<i>which</i>	Timer type (ITIMER_REAL).
out	<i>curr_value</i>	Old timer value.

References [getitimer\(\)](#).

Referenced by [getitimer\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.292.3.2 raise()

```
int L4Re::Itas::raise (
    L4::Ipc::Cap< L4::Thread > thread,
    int sig)
```

Send a signal to the calling thread.

Parameters

in	<i>thread</i>	Thread cap of the calling thread.
in	<i>sig</i>	Signal that shall be raised.

References [raise\(\)](#).

Referenced by [raise\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.292.3.3 register_thread()

```
int L4Re::Itas::register_thread (
    L4::Ipc::Cap< L4::Thread > parent,
    L4::Ipc::Cap< L4::Thread > thread_cap,
    l4_addr_t thread_utcb)
```

Register new thread.

Makes the newly created thread known to ITAS. The ITAS will do the `thread_cap->control()` to bind the thread to the task and attach the gates for pager and exception handler.

Parameters

<i>parent</i>	The capability of the thread that created the new thread.
<i>thread_cap</i>	The capability of the new thread.
<i>thread_utcb</i>	The address of the allocated UTCB of the new thread.

16.292.3.4 setitimer()

```
int L4Re::Itas::setitimer (
    int which,
    const struct itimerval * new_value,
    struct itimerval * old_value)
```

Set process interval timer.

See IEEE Std 1003.1-2017 [setitimer\(\)](#) for details.

Parameters

in	<i>which</i>	Timer type (ITIMER_REAL).
in	<i>new_value</i>	New timer value.
out	<i>old_value</i>	Old timer value.

Referenced by [sigpending\(\)](#).

Here is the caller graph for this function:



16.292.3.5 sigaction()

```
int L4Re::Itas::sigaction (
    int signum,
    const struct sigaction * act,
    struct sigaction * oldact)
```

Examine and change a POSIX signal action.

See IEEE Std 1003.1-2024 [sigaction\(\)](#) for the behaviour of the method.

If `act->sa_flags` is [Ignore_sigaction](#), the new action is ignored.

Parameters

in	<i>signum</i>	Signal number to be examined and/or modified.
in	<i>act</i>	New signal action.
out	<i>oldact</i>	Old signal action.

References [sigaction\(\)](#), and [sigaltstack\(\)](#).

Referenced by [sigaction\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.292.3.6 sigaltstack()

```
int L4Re::Itas::sigaltstack (
    L4::Ipc::Cap< L4::Thread > thread,
    const struct sigaltstack * ss,
    struct sigaltstack * oss)
```

Examine or set alternate POSIX signal stack.

See IEEE Std 1003.1-2024 [sigaltstack\(\)](#) for the behaviour of the method.

If `ss->ss_flags` is -1, the new sigaltstack will be ignored.

Parameters

in	<i>thread</i>	Thread cap of the thread whose sigaltstack is examined and/or modified.
in	<i>ss</i>	The new sigaltstack.
out	<i>oss</i>	The old sigaltstack.

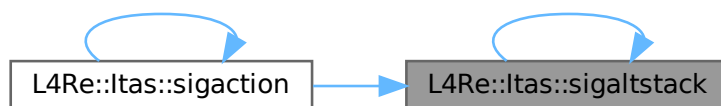
References [sigaltstack\(\)](#), and [sigprocmask\(\)](#).

Referenced by [sigaction\(\)](#), and [sigaltstack\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.292.3.7 sigpending()

```
int L4Re::Itas::sigpending (
    L4::Ipc::Cap< L4::Thread > thread,
    sigset_t * set)
```

Query pending signals.

See IEEE Std 1003.1-2024 [sigpending\(\)](#) for details.

Parameters

in	<i>thread</i>	Thread cap of the thread whose pending signal are examined.
out	<i>set</i>	Pending signals of thread.

References [setitimer\(\)](#).

Referenced by [sigprocmask\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.292.3.8 sigprocmask()

```

int L4Re::Itas::sigprocmask (
    L4::Ipc::Cap< L4::Thread > thread,
    int how,
    sigset_t const * set,
    sigset_t * oldset)

```

Examine or set process signal mask.

See IEEE Std 1003.1-2024 [sigprocmask\(\)](#) for the behaviour or the method.

If `how` is `-1`, the signal mask is left unchanged.

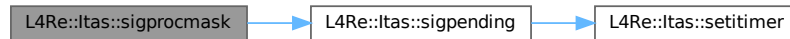
Parameters

in	<i>thread</i>	Thread cap of the thread whose signal mask is examined and/or modified.
in	<i>how</i>	Operation (<code>SIG_BLOCK</code> , <code>SIG_UNBLOCK</code> , <code>SIG_SETMASK</code> or <code>-1</code>).
in	<i>set</i>	The new signal mask.
out	<i>oldset</i>	The old signal mask.

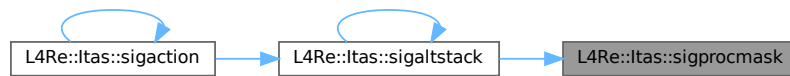
References [sigpending\(\)](#).

Referenced by [sigaltstack\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.292.3.9 unregister_thread()

```
int L4Re::Itas::unregister_thread (
    L4::Ipc::Cap< L4::Thread > thread)
```

Unregister a thread.

The gates for the thread's pager and exception handler will be destroyed. Thus, the thread must be destroyed after the call.

Parameters

<i>thread</i>	The destroyed thread.
---------------	-----------------------

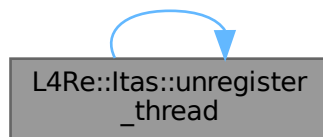
References [unregister_thread\(\)](#).

Referenced by [unregister_thread\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

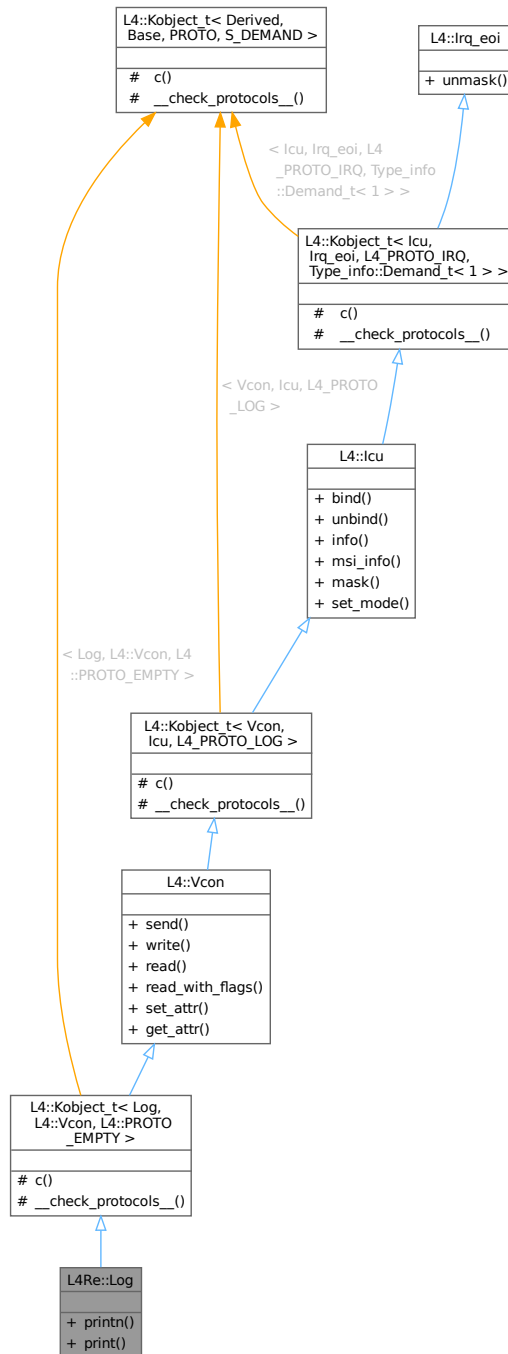
- `l4/re/itas`

16.293 L4Re::Log Class Reference

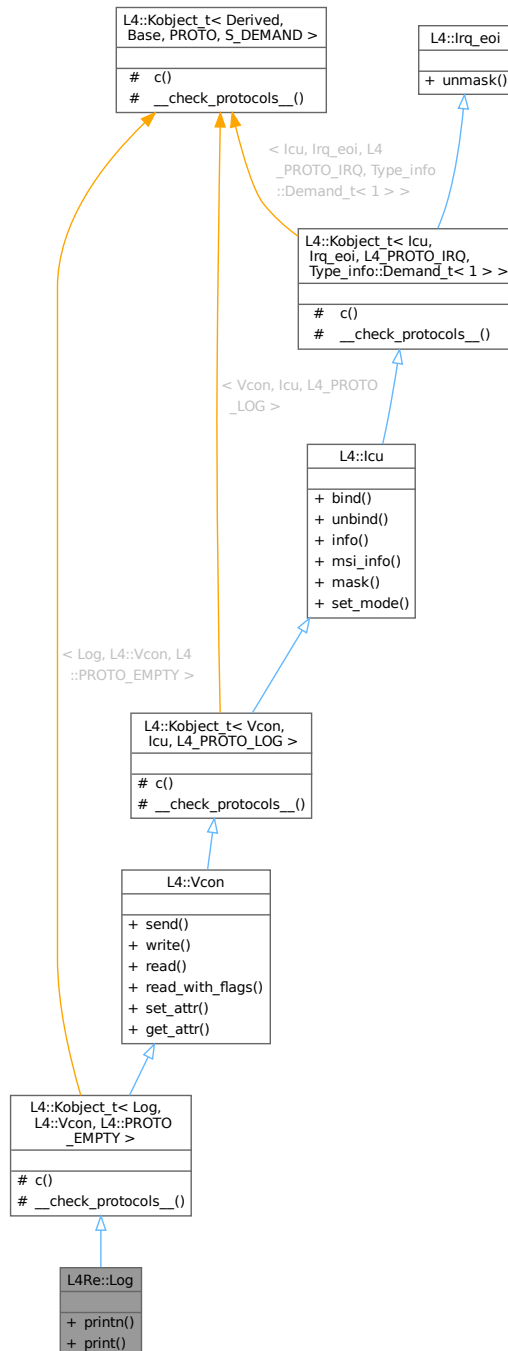
[Log](#) interface class.

```
#include <log>
```

Inheritance diagram for L4Re::Log:



Collaboration diagram for L4Re::Log:



Public Member Functions

- void `printn` (char const *string, int len) const noexcept
Print string with length len, NULL characters don't matter.
- void `print` (char const *string) const noexcept
Print NULL-terminated string.

Public Member Functions inherited from [L4::Vcon](#)

- [l4_msgtag_t send](#) (char const *buf, unsigned size, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) const noexcept
Send data to `this` virtual console.
- long [write](#) (char const *buf, unsigned size, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) const noexcept
Write data to `this` virtual console.
- int [read](#) (char *buf, unsigned size, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) const noexcept
Read data from `this` virtual console.
- int [read_with_flags](#) (char *buf, unsigned size, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) const noexcept
Read data from `this` virtual console which also returns flags.
- [l4_msgtag_t set_attr](#) ([l4_vcon_attr_t](#) const *attr, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) const noexcept
Set the attributes of `this` virtual console.
- [l4_msgtag_t get_attr](#) ([l4_vcon_attr_t](#) *attr, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) const noexcept
Get attributes of `this` virtual console.

Public Member Functions inherited from [L4::Icu](#)

- [l4_msgtag_t bind](#) (unsigned irqnum, [L4::Cap< Triggerable >](#) irq, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept
Bind an interrupt line of an interrupt controller to an interrupt object.
- [l4_msgtag_t unbind](#) (unsigned irqnum, [L4::Cap< Triggerable >](#) irq, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept
Remove binding of an interrupt line from the interrupt controller object.
- [l4_msgtag_t info](#) ([l4_icu_info_t](#) *info, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept
Get information about the ICU features.
- [l4_msgtag_t msi_info](#) ([l4_umword_t](#) irqnum, [l4_uint64_t](#) source, [l4_icu_msi_info_t](#) *msi_info)
Get MSI info about IRQ.
- [l4_msgtag_t mask](#) (unsigned irqnum, [l4_umword_t](#) *label=0, [l4_timeout_t](#) to=[L4_IPC_NEVER](#), [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept
Mask an IRQ line.
- [l4_msgtag_t set_mode](#) (unsigned irqnum, [l4_umword_t](#) mode, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept
Set interrupt mode.

Public Member Functions inherited from [L4::Irq_eoi](#)

- [l4_msgtag_t unmask](#) (unsigned irqnum, [l4_umword_t](#) *label=0, [l4_timeout_t](#) to=[L4_IPC_NEVER](#), [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept
Unmask the given interrupt line.

Additional Inherited Members

Protected Types inherited from [L4::Kobject_t< Log, L4::Vcon, L4::PROTO_EMPTY >](#)

- typedef Log **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< PROTO, Log > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__Iface** >, typename L4::Vcon::__Iface_list > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Types inherited from [L4::Kobject_t< Vcon, Icu, L4_PROTO_LOG >](#)

- typedef [Vcon](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< PROTO, [Vcon](#) > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< [__Iface](#) >, typename Icu::__Iface_list > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Types inherited from

[L4::Kobject_t< Icu, Irq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > >](#)

- typedef [Icu](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< PROTO, [Icu](#) > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< [__Iface](#) >, typename Irq_eoi::__Iface_list > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from

[L4::Kobject_t< Log, L4::Vcon, L4::PROTO_EMPTY >](#)

- [L4::Cap< Class > c](#) () const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from

[L4::Kobject_t< Vcon, Icu, L4_PROTO_LOG >](#)

- [L4::Cap< Class > c](#) () const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from

[L4::Kobject_t< Icu, Irq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > >](#)

- [L4::Cap< Class > c](#) () const noexcept
Get the capability to ourselves.

Static Protected Member Functions inherited from

[L4::Kobject_t< Log, L4::Vcon, L4::PROTO_EMPTY >](#)

- static void [__check_protocols](#) () noexcept
Helper to check for protocol conflicts.

Static Protected Member Functions inherited from

[L4::Kobject_t< Vcon, Icu, L4_PROTO_LOG >](#)

- static void [__check_protocols](#) () noexcept
Helper to check for protocol conflicts.

Static Protected Member Functions inherited from

[L4::Kobject_t< lcu, Irq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > >](#)

- static void `__check_protocols__()` noexcept
Helper to check for protocol conflicts.

16.293.1 Detailed Description

[Log](#) interface class.

Definition at line 33 of file [log](#).

16.293.2 Member Function Documentation

16.293.2.1 `print()`

```
void L4Re::Log::print (
    char const * string) const    [noexcept]
```

Print NULL-terminated string.

Parameters

<i>string</i>	string to print
---------------	-----------------

16.293.2.2 `println()`

```
void L4Re::Log::println (
    char const * string,
    int len) const    [noexcept]
```

Print string with length *len*, NULL characters don't matter.

Parameters

<i>string</i>	string to print
<i>len</i>	length of string

The documentation for this class was generated from the following file:

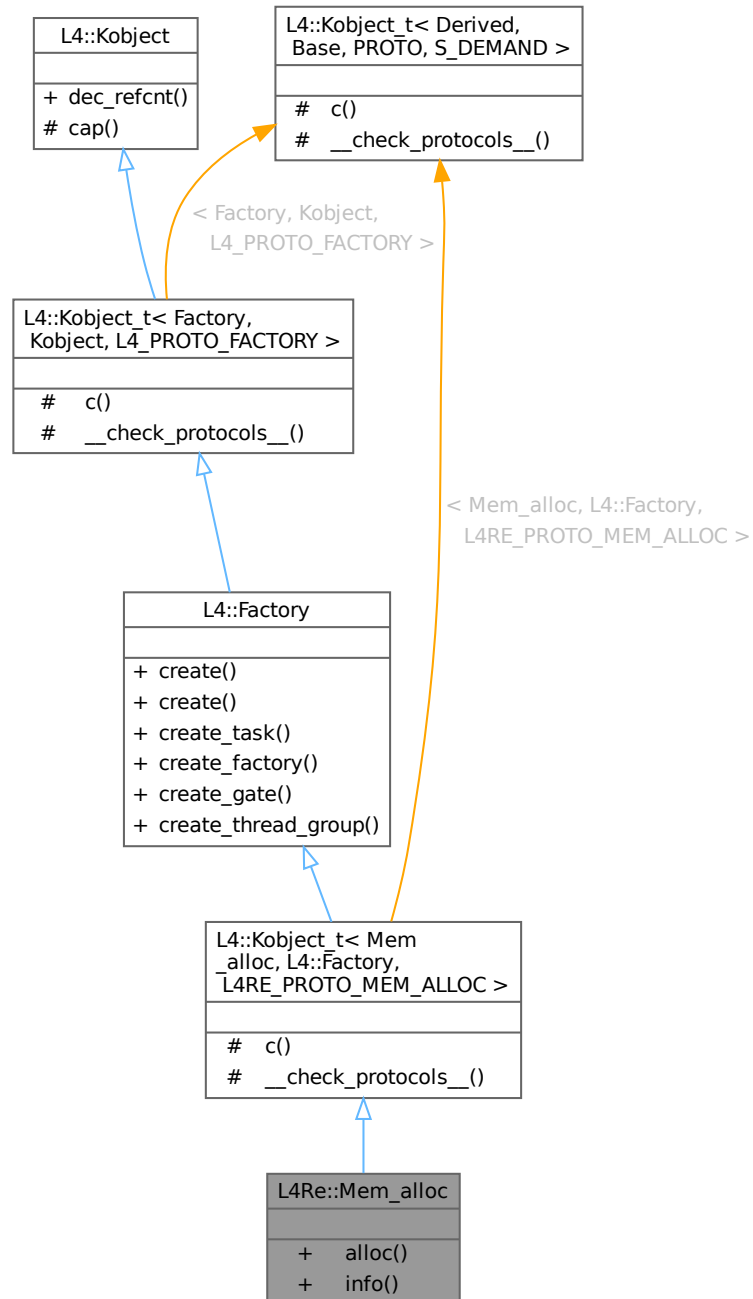
- [l4/re/log](#)

16.294 L4Re::Mem_alloc Class Reference

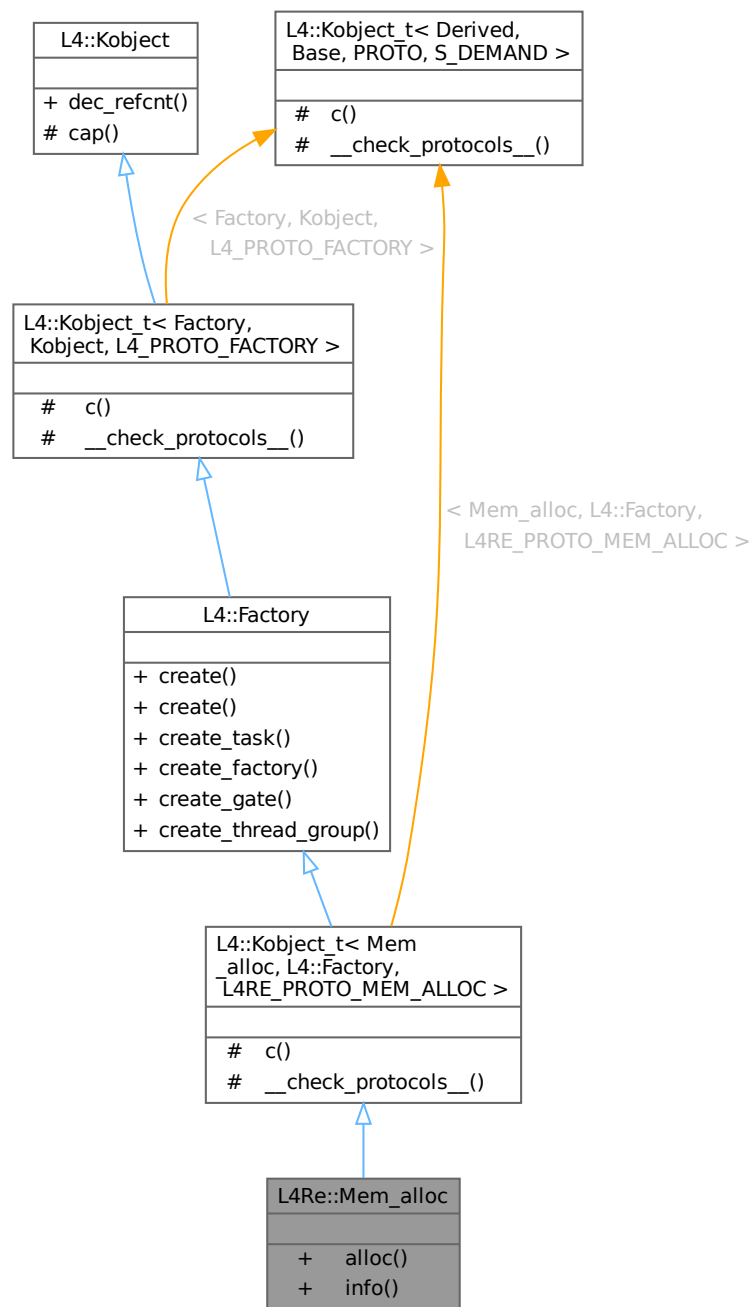
Memory allocation interface.

```
#include <mem_alloc>
```

Inheritance diagram for L4Re::Mem_alloc:



Collaboration diagram for L4Re::Mem_alloc:



Data Structures

- struct [Stats](#)
Statistics about memory-allocator.

Public Types

- enum [Mem_alloc_flags](#) { [Continuous](#) = 0x01 , [Pinned](#) = 0x02 , [Super_pages](#) = 0x04 , [Fixed_paddr](#) = 0x08 }

Flags for the allocator.

Public Member Functions

- long [alloc](#) (long size, [L4::Cap](#)< [Dataspace](#) > mem, unsigned long flags=0, unsigned long align=0, [l4_addr_t](#) paddr=0) const noexcept
Allocate anonymous memory.
- long [info](#) ([Stats](#) &stats)
Get allocator information.

Public Member Functions inherited from [L4::Factory](#)

- [S](#) [create](#) ([Cap](#)< void > target, long obj, [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept
Generic create call to the factory.
- template<typename OBJ>
[S](#) [create](#) ([Cap](#)< OBJ > target, [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept
Create call for typed capabilities.
- [l4_msgtag_t](#) [create_task](#) ([Cap](#)< [Task](#) > const &target_cap, [l4_fpage_t](#) *utcb_area, [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept
Create a new task.
- [l4_msgtag_t](#) [create_factory](#) ([Cap](#)< [Factory](#) > const &target_cap, unsigned long limit, [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept
Create a new factory.
- [l4_msgtag_t](#) [create_gate](#) ([Cap](#)< void > const &target_cap, [Cap](#)< [Snd_destination](#) > const &snd_dst_cap, [l4_umword_t](#) label, [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept
Create a new IPC gate, optionally bound to a send destination (a thread or thread group).
- [l4_msgtag_t](#) [create_thread_group](#) ([Cap](#)< [Thread_group](#) > const &target_cap, unsigned policy, [l4_utcb_t](#) *utcb=[l4_utcb](#)()) noexcept
Create a new thread group.

Public Member Functions inherited from [L4::Kobject](#)

- [l4_msgtag_t](#) [dec_refcnt](#) ([l4_mword_t](#) diff, [l4_utcb_t](#) *utcb=[l4_utcb](#)())
Decrement the in kernel reference counter for the object.

Additional Inherited Members

Protected Types inherited from

[L4::Kobject_t](#)< [Mem_alloc](#), [L4::Factory](#), [L4RE_PROTO_MEM_ALLOC](#) >

- typedef [Mem_alloc](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef [Typeid::Iface](#)< [PROTO](#), [Mem_alloc](#) > **__iface**
The interface description for the derived class.
- typedef [Typeid::Merge_list](#)< [Typeid::Iface_list](#)< **__iface** >, typename [L4::Factory::__iface_list](#) > **__iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Types inherited from [L4::Kobject_t< Factory, Kobject, L4_PROTO_FACTORY >](#)

- typedef [Factory](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< PROTO, [Factory](#) > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__Iface** >, typename Kobject::__Iface_list > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from [L4::Kobject_t< Mem_alloc, L4::Factory, L4RE_PROTO_MEM_ALLOC >](#)

- [L4::Cap< Class > c](#) () const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from [L4::Kobject_t< Factory, Kobject, L4_PROTO_FACTORY >](#)

- [L4::Cap< Class > c](#) () const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from [L4::Kobject](#)

- [l4_cap_idx_t cap](#) () const noexcept
Return capability selector.

Static Protected Member Functions inherited from [L4::Kobject_t< Mem_alloc, L4::Factory, L4RE_PROTO_MEM_ALLOC >](#)

- static void [__check_protocols__](#) () noexcept
Helper to check for protocol conflicts.

Static Protected Member Functions inherited from [L4::Kobject_t< Factory, Kobject, L4_PROTO_FACTORY >](#)

- static void [__check_protocols__](#) () noexcept
Helper to check for protocol conflicts.

16.294.1 Detailed Description

Memory allocation interface.

The memory-allocator API is the basic API to allocate memory from the [L4Re](#) subsystem. The memory is allocated in terms of dataspace (see [L4Re::Dataspace](#)). The provided dataspace have at least the property that data written to such a dataspace is available as long as the dataspace is not freed or the data is not overwritten. In particular, the memory backing a dataspace from an allocator need not be allocated instantly, but may be allocated lazily on demand.

A memory allocator can provide dataspace with additional properties, such as physically contiguous memory, pre-allocated memory, or pinned memory. To request memory with an additional property the [L4Re::Mem_alloc::alloc\(\)](#) method provides a flags parameter. If the concrete implementation of a memory allocator does not support or allow allocation of memory with a certain property, the allocation may be refused.

Definition at line 52 of file [mem_alloc](#).

16.294.2 Member Enumeration Documentation

16.294.2.1 Mem_alloc_flags

enum L4Re::Mem_alloc::Mem_alloc_flags

Flags for the allocator.

They describe requested properties of the allocated memory. Support of these properties by the dataspace provider is optional.

Enumerator

Continuous	Allocate physically contiguous memory.
Pinned	Deprecated, use L4Re::Dma_space instead.
Super_pages	Allocate super pages.
Fixed_paddr	Allocate at fixed physical address. Only honored on no-MMU systems. Will fail on MMU systems.

Definition at line 62 of file [mem_alloc](#).

16.294.3 Member Function Documentation

16.294.3.1 alloc()

```
long L4Re::Mem_alloc::alloc (
    long size,
    L4::Cap< Dataspace > mem,
    unsigned long flags = 0,
    unsigned long align = 0,
    l4_addr_t paddr = 0) const [noexcept]
```

Allocate anonymous memory.

Parameters

	<i>size</i>	Size in bytes to be requested. Allocation granularity is (super)pages, however, the allocator will store the byte-granular given size as the size of the dataspace and consecutively will use this byte-granular size for servicing the dataspace. Allocators may optionally also implement a maximum allocation strategy: if <i>size</i> is a negative value and <i>flags</i> set the Mem_alloc_flags::Continuous bit, the allocator tries to allocate as much memory as possible leaving an amount of at least <i>-size</i> bytes within the associated quota.
out	<i>mem</i>	Capability slot where the capability to the dataspace is received.
	<i>flags</i>	Special dataspace properties, see Mem_alloc_flags
	<i>align</i>	Log2 alignment of dataspace if supported by allocator, will be at least L4_PAGESHIFT, with Super_pages flag set at least L4_SUPERPAGESHIFT
	<i>paddr</i>	The physical address where the dataspace should be allocated if Mem_alloc_flags::Fixed flag is set.

Return values

<i>0</i>	Success
<i>-L4_ERANGE</i>	Given size not supported.
<i>-L4_ENOMEM</i>	Not enough memory available.
<i><0</i>	IPC error

Definition at line 24 of file [mem_alloc_impl.h](#).

References [L4::Kobject::cap\(\)](#), [Fixed_paddr](#), and [l4_error\(\)](#).

Here is the call graph for this function:



16.294.3.2 info()

```
long L4Re::Mem_alloc::info (
    Stats & stats)
```

Get allocator information.

Parameters

out	stats	Allocator information
-----	-------	-----------------------

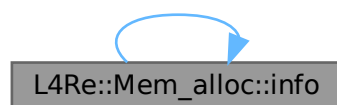
Return values

<i>0</i>	Success
<i><0</i>	IPC error

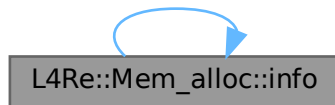
References [info\(\)](#), and [L4_INLINE_RPC](#).

Referenced by [info\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following files:

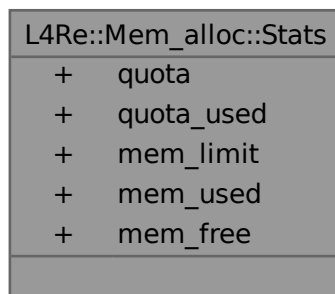
- [l4/re/mem_alloc](#)
- [l4/re/impl/mem_alloc_impl.h](#)

16.295 L4Re::Mem_alloc::Stats Struct Reference

Statistics about memory-allocator.

```
#include <mem_alloc>
```

Collaboration diagram for L4Re::Mem_alloc::Stats:



Data Fields

- [l4_size_t quota](#)
Memory quota of this allocator.
- [l4_size_t quota_used](#)
Amount of currently used quota of this allocator.
- [l4_size_t mem_limit](#)
Maximum amount of memory that can be allocated by this allocator.
- [l4_size_t mem_used](#)
Amount of currently allocated memory.
- [l4_size_t mem_free](#)
Amount of memory that is still available for allocation.

16.295.1 Detailed Description

Statistics about memory-allocator.

Definition at line 74 of file [mem_alloc](#).

16.295.2 Field Documentation

16.295.2.1 mem_free

```
l4_size_t L4Re::Mem_alloc::Stats::mem_free
```

Amount of memory that is still available for allocation.

This field can be lower than [mem_limit](#) - [mem_used](#). In this case the system may be over-committed and there is globally not enough memory left. Also, if the quota is already used up for sub-factories (see [quota_used](#)), there may be not enough quota left.

Definition at line 126 of file [mem_alloc](#).

16.295.2.2 mem_limit

```
l4_size_t L4Re::Mem_alloc::Stats::mem_limit
```

Maximum amount of memory that can be allocated by this allocator.

Will never exceed the [quota](#) but may be smaller if the system has less memory installed.

Definition at line 102 of file [mem_alloc](#).

16.295.2.3 mem_used

```
l4_size_t L4Re::Mem_alloc::Stats::mem_used
```

Amount of currently allocated memory.

This field represents the amount of memory that is in use by this allocator. It recursively includes the memory used by sub-factories, if any.

Will never exceed [mem_limit](#) or [quota_used](#).

Note

Dataspaces may allocate memory lazily! As such, the field will increase only after pages have been allocated to a dataspace.

Definition at line 116 of file [mem_alloc](#).

16.295.2.4 quota

```
l4_size_t L4Re::Mem_alloc::Stats::quota
```

Memory quota of this allocator.

Strictly limits the amount of memory that can be allocated. This may be larger than there is actual physical memory available. In particular, the root factory has an artificial quota and returns -1 in this field.

Definition at line 83 of file [mem_alloc](#).

16.295.2.5 quota_used

```
l4_size_t L4Re::Mem_alloc::Stats::quota_used
```

Amount of currently used quota of this allocator.

The amount of used quota is not necessarily linked to the current memory usage. See [mem_used](#) for this information. The quota of a factory is immediately and fully accounted to the parent factory quota.

This value may even exceed [mem_limit](#) if the system is over-committed.

Definition at line 94 of file [mem_alloc](#).

The documentation for this struct was generated from the following file:

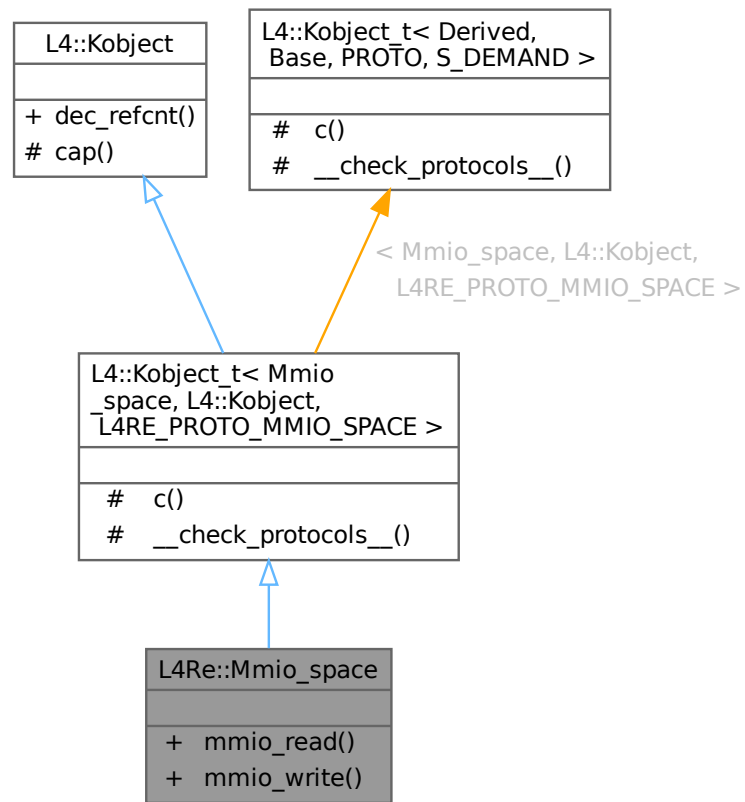
- [l4/re/mem_alloc](#)

16.296 L4Re::Mmio_space Struct Reference

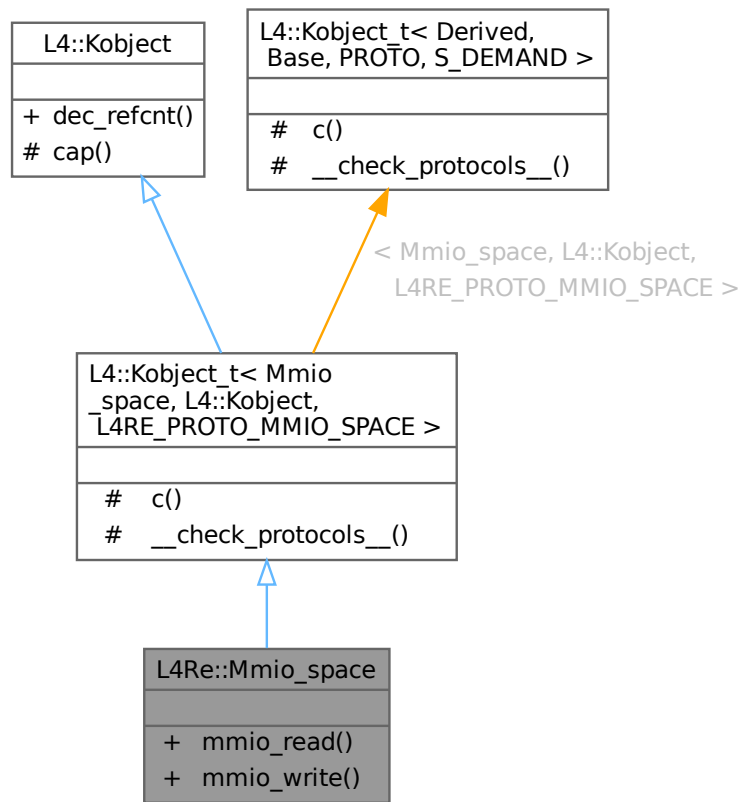
Interface for memory-like address space accessible via IPC.

```
#include <mmio_space>
```

Inheritance diagram for L4Re::Mmio_space:



Collaboration diagram for L4Re::Mmio_space:



Public Types

- enum `Access_width` { `Wd_8bit` = 0 , `Wd_16bit` = 1 , `Wd_32bit` = 2 , `Wd_64bit` = 3 }
Actual size of the value to read or write.
- typedef `l4_uint64_t` `Addr`
Device address.

Public Member Functions

- long `mmio_read` (`Addr` addr, char width, `l4_uint64_t` *value)
Read a value from the given address.
- long `mmio_write` (`Addr` addr, char width, `l4_uint64_t` value)
Write a value to the given address.

Public Member Functions inherited from L4::Kobject

- `l4_msgtag_t` `dec_refcnt` (`l4_mword_t` diff, `l4_utcb_t` *utcb=`l4_utcb()`)
Decrement the in kernel reference counter for the object.

Additional Inherited Members

Protected Types inherited from

[L4::Kobject_t](#)< [Mmio_space](#), [L4::Kobject](#), [L4RE_PROTO_MMIO_SPACE](#) >

- typedef [Mmio_space](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< [PROTO](#), [Mmio_space](#) > **__iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< [__iface](#) >, typename [L4::Kobject::__iface_list](#) > **__iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from

[L4::Kobject_t](#)< [Mmio_space](#), [L4::Kobject](#), [L4RE_PROTO_MMIO_SPACE](#) >

- [L4::Cap](#)< [Class](#) > **c** () const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from [L4::Kobject](#)

- [l4_cap_idx_t](#) **cap** () const noexcept
Return capability selector.

Static Protected Member Functions inherited from

[L4::Kobject_t](#)< [Mmio_space](#), [L4::Kobject](#), [L4RE_PROTO_MMIO_SPACE](#) >

- static void **__check_protocols__** () noexcept
Helper to check for protocol conflicts.

16.296.1 Detailed Description

Interface for memory-like address space accessible via IPC.

This interface defines methods for indirect access to MMIO regions.

Memory mapped IO (MMIO) is used by device drivers to control hardware devices. Access to MMIO regions is assigned to user-level device drivers via mappings of memory pages.

However, there are hardware platforms where MMIO regions for different devices share the same memory page. With respect to security and safety, it is often not allowed to map a memory page to multiple device drivers because the driver of one device could then influence operation of another device, which violates security boundaries.

A solution to that problem is to implement a third (trusted) component that gets exclusive access to the shared memory page, and that drivers can access via IPC with the [Mmio_space](#) protocol. This proxy-component can then enforce an access policy.

Include File

```
#include <l4/re/mmio_space>
```

Definition at line 45 of file [mmio_space](#).

16.296.2 Member Enumeration Documentation

16.296.2.1 Access_width

enum [L4Re::Mmio_space::Access_width](#)

Actual size of the value to read or write.

Enumerator

Wd_8bit	Value is a byte.
Wd_16bit	Value is a 2-byte word.
Wd_32bit	Value is a 4-byte word.
Wd_64bit	Value is a 8-byte word.

Definition at line 49 of file [mmio_space](#).

16.296.3 Member Function Documentation

16.296.3.1 mmio_read()

```
long L4Re::Mmio_space::mmio_read (
    Addr addr,
    char width,
    l4_uint64_t * value)
```

Read a value from the given address.

Parameters

	<i>addr</i>	Device virtual address to read from. The address must be aligned relative to the access width.
	<i>width</i>	Access width of value to be read, see Access_width .
out	<i>value</i>	Return value. If width is smaller than 64 bit, the upper bits are guaranteed to be 0.

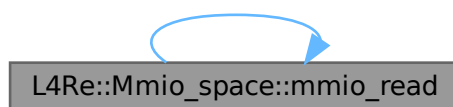
Return values

L4_EOK	Success.
-L4_EPERM	Insufficient read rights.
-L4_EINVAL	Address does not exist or cannot be accessed with the given width.

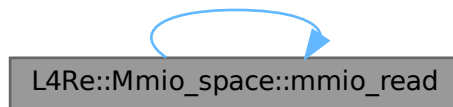
References [mmio_read\(\)](#).

Referenced by [mmio_read\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.296.3.2 mmio_write()

```
long L4Re::Mmio_space::mmio_write (  
    Addr addr,  
    char width,  
    l4_uint64_t value)
```

Write a value to the given address.

Parameters

<i>addr</i>	Device virtual address to write to. The address must be aligned relative to the access width.
<i>width</i>	Access width of value to write, see Access_width .
<i>value</i>	Value to write. If width is smaller than 64 bit, the upper bits are ignored.

Return values

L4_EOK	Success.
<code>-L4_EPERM</code>	Insufficient write rights.
<code>-L4_EINVAL</code>	Address does not exist or cannot be accessed with the given width.

References [mmio_write\(\)](#).

Referenced by [mmio_write\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this struct was generated from the following file:

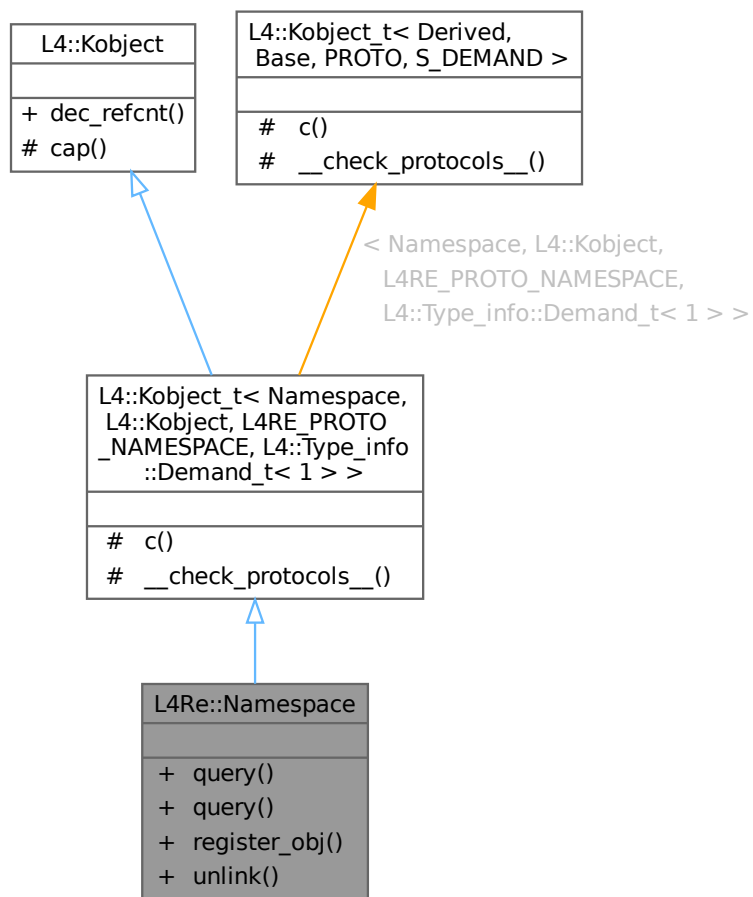
- [l4/re/mmio_space](#)

16.297 L4Re::Namespace Class Reference

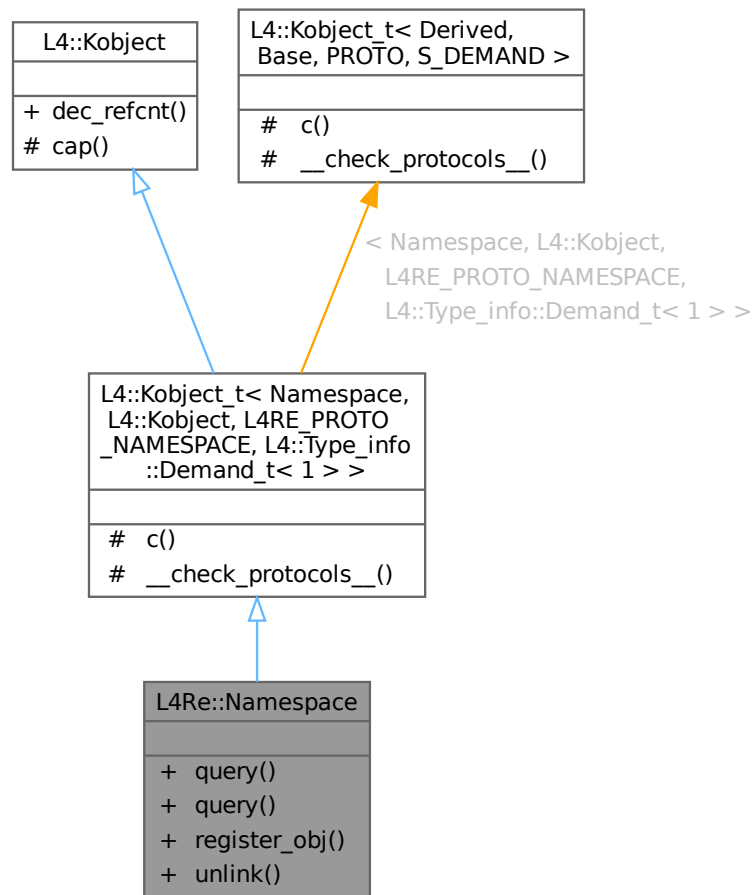
Name-space interface.

```
#include <namespace>
```

Inheritance diagram for L4Re::Namespace:



Collaboration diagram for L4Re::Namespace:



Public Types

- enum `Register_flags` {
`Ro` = `L4_CAP_FPAGE_RO` , `Rw` = `L4_CAP_FPAGE_RW` , `Rs` = `L4_CAP_FPAGE_RS` , `Rws` = `L4_CAP_FPAGE_RWS` ,
`Strong` = `L4_CAP_FPAGE_S` , `Trusted` = `0x008` , `Cap_flags` = `Ro | Rw | Strong | Trusted` , `Link` = `0x100` ,
`Overwrite` = `0x200` }
Flags for registering name spaces.
- enum `Query_result_flags` { `Partly_resolved` = `0x020` }
Flags returned by query IPC, only used internally.
- enum `Query_timeout` { `To_default` = `3600000` , `To_non_blocking` = `0` }
Timeout values for query operation.

Public Member Functions

- long `query` (char const *name, `L4::Cap`< void > const &`cap`, int timeout=`To_default`, `l4_umword_t` *local_id=0, bool iterate=true) const noexcept

Query the name space for a named object.

- long [query](#) (char const *name, unsigned len, [L4::Cap](#)< void > const &cap, int timeout=[To_default](#), [l4_umword_t](#) *local_id=0, bool iterate=true) const noexcept

Query the name space for a named object.

- long [register_obj](#) (char const *name, [L4::lpc::Cap](#)< void > obj, unsigned flags=[Rw](#)) const noexcept

Register an object with a name.

- long [unlink](#) (char const *name)

Remove an entry from the name space.

Public Member Functions inherited from [L4::Kobject](#)

- [l4_msgtag_t dec_refcnt](#) ([l4_mword_t](#) diff, [l4_utcb_t](#) *utcb=[l4_utcb](#)())

Decrement the in kernel reference counter for the object.

Additional Inherited Members

Protected Types inherited from

[L4::Kobject_t](#)< [Namespace](#), [L4::Kobject](#), [L4RE_PROTO_NAMESPACE](#), [L4::Type_info::Demand_t](#)< 1 >

- typedef [Namespace](#) **Class**

The target interface type (inheriting from [Kobject_t](#)).

- typedef [Typeid::Iface](#)< [PROTO](#), [Namespace](#) > **__iface**

The interface description for the derived class.

- typedef [Typeid::Merge_list](#)< [Typeid::Iface_list](#)< **__iface** >, typename [L4::Kobject::__iface_list](#) > **__iface_list**

The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from

[L4::Kobject_t](#)< [Namespace](#), [L4::Kobject](#), [L4RE_PROTO_NAMESPACE](#), [L4::Type_info::Demand_t](#)< 1 >

- [L4::Cap](#)< [Class](#) > **c** () const noexcept

Get the capability to ourselves.

Protected Member Functions inherited from [L4::Kobject](#)

- [l4_cap_idx_t cap](#) () const noexcept

Return capability selector.

Static Protected Member Functions inherited from

[L4::Kobject_t](#)< [Namespace](#), [L4::Kobject](#), [L4RE_PROTO_NAMESPACE](#), [L4::Type_info::Demand_t](#)< 1 >

- static void **__check_protocols** () noexcept

Helper to check for protocol conflicts.

16.297.1 Detailed Description

Name-space interface.

All name space objects must provide this interface. However, it is not mandatory that a name space object allows to register new capabilities.

The name lookup is done iteratively, this means the hierarchical names are resolved component wise by the client itself.

Definition at line 49 of file [namespace](#).

16.297.2 Member Enumeration Documentation

16.297.2.1 Query_result_flags

```
enum L4Re::Namespace::Query_result_flags
```

Flags returned by query IPC, only used internally.

Enumerator

Partly_resolved	Name was only partly resolved.
-----------------	--------------------------------

Definition at line 77 of file [namespace](#).

16.297.2.2 Query_timeout

```
enum L4Re::Namespace::Query_timeout
```

Timeout values for query operation.

Enumerator

To_default	Default timeout.
To_non_blocking	Expect callee to answer immediately.

Definition at line 83 of file [namespace](#).

16.297.2.3 Register_flags

```
enum L4Re::Namespace::Register_flags
```

Flags for registering name spaces.

Enumerator

Ro	Read-only.
Rw	Read-write.
Rs	Read-only + strong.
Rws	Read-write + strong.
Strong	Strong.
Trusted	Obsolete, do not use.
Link	Obsolete, do not use.
Overwrite	If entry already exists, overwrite it.

Definition at line 57 of file [namespace](#).

16.297.3 Member Function Documentation

16.297.3.1 `query()` [1/2]

```
long L4Re::Namespace::query (
    char const * name,
    L4::Cap< void > const & cap,
    int timeout = To_default,
    l4_umword_t * local_id = 0,
    bool iterate = true) const [noexcept]
```

Query the name space for a named object.

Parameters

in	<i>name</i>	String to query (without any leading slashes).
out	<i>cap</i>	Capability slot where the received capability will be put.
in	<i>timeout</i>	Timeout of query in milliseconds. The client will only wait if a name has already been registered with the server but no object has yet been attached.
out	<i>local_id</i>	If given, L4_RCV_ITEM_LOCAL_ID will be set for the IPC from the name space, so that if the capability that was received is a local item, the capability ID will be returned with this parameter.
in	<i>iterate</i>	If true, the client will try to resolve names by iteratively calling the name spaces until the name is fully resolved.

Return values

0	Name could be fully resolved.
>0	Name could only be partly resolved. The number of remaining characters is returned.
-L4_ENOENT	Entry could not be found.
-L4_EAGAIN	Entry exists but no object is yet attached. Try again later.
<0	IPC errors, see l4_error_code_t .

Definition at line 114 of file [namespace_impl.h](#).

References [query\(\)](#).

Referenced by [query\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.297.3.2 `query()` [2/2]

```

long L4Re::Namespace::query (
    char const * name,
    unsigned len,
    L4::Cap< void > const & cap,
    int timeout = To_default,
    l4_umword_t * local_id = 0,
    bool iterate = true) const [noexcept]
  
```

Query the name space for a named object.

The query string does not necessarily need to be null-terminated.

Parameters

in	<i>len</i>	Length of the string to query without any terminating null characters.
in	<i>name</i>	String to query (without any leading slashes).
out	<i>cap</i>	Capability slot where the received capability will be put.

in	<i>timeout</i>	Timeout of query in milliseconds. The client will only wait if a name has already been registered with the server but no object has yet been attached.
out	<i>local_id</i>	If given, L4_RCV_ITEM_LOCAL_ID will be set for the IPC from the name space, so that if the capability that was received is a local item, the capability ID will be returned with this parameter.
in	<i>iterate</i>	If true, the client will try to resolve names by iteratively calling the name spaces until the name is fully resolved.

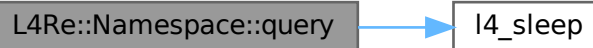
Return values

<i>0</i>	Name could be fully resolved.
<i>>0</i>	Name could only be partly resolved. The number of remaining characters is returned.
<i>-L4_ENOENT</i>	Entry could not be found.
<i>-L4_EAGAIN</i>	Entry exists but no object is yet attached. Try again later.
<i><0</i>	IPC errors, see l4_error_code_t .

Definition at line 66 of file [namespace_impl.h](#).

References [L4_EAGAIN](#), [L4_EINVAL](#), [l4_sleep\(\)](#), and [L4_UNLIKELY](#).

Here is the call graph for this function:



16.297.3.3 register_obj()

```

long L4Re::Namespace::register_obj (
    char const * name,
    L4::Ipc::Cap< void > obj,
    unsigned flags = Rw) const [inline], [noexcept]
  
```

Register an object with a name.

Parameters

<i>name</i>	Name under which the object should be registered.
<i>obj</i>	Capability to object to register. An invalid capability may be given to only reserve the name for later use.

<i>flags</i>	Flags to assign to the entry, see L4Re::Namespace::Register_flags . Note that the rights that are assigned to a capability are not only determined by the rights given in these flags but also by the rights with which the <code>obj</code> capability was mapped to the name space.
--------------	---

Return values

<i>0</i>	Object was successfully registered with <i>name</i> .
<i>-L4_EEXIST</i>	Name already registered.
<i>-L4_EPERM</i>	Insufficient permissions; see precondition.
<i>-L4_ENOMEM</i>	Server has insufficient resources.
<i>-L4_EINVAL</i>	Invalid parameter.
<i><0</i>	IPC errors, see l4_error_code_t .

Precondition

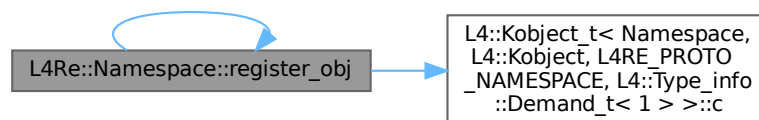
The invoked [Namespace](#) capability must have the permission [L4_CAP_FPAGE_W](#).

Definition at line 165 of file [namespace](#).

References [L4::Kobject_t< Namespace, L4::Kobject, L4RE_PROTO_NAMESPACE, L4::Type_info::Demand_t< 1 > >::c\(\)](#), [register_obj\(\)](#), and [Rw](#).

Referenced by [register_obj\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.297.3.4 unlink()

```
long L4Re::Namespace::unlink (
    char const * name) [inline]
```

Remove an entry from the name space.

Parameters

<i>name</i>	Name of the entry to remove.
-------------	------------------------------

Return values

0	Entry successfully removed.
-L4_ENOENT	Given name does not exist.
-L4_EPERM	Insufficient permissions; see precondition.
-L4_EACCESS	Name cannot be removed.
<0	IPC errors, see l4_error_code_t .

Precondition

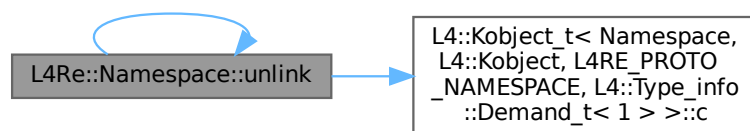
The invoked [Namespace](#) capability must have the permission [L4_CAP_FPAGE_W](#).

Definition at line 191 of file [namespace](#).

References [L4::Kobject_t< Namespace, L4::Kobject, L4RE_PROTO_NAMESPACE, L4::Type_info::Demand_t< 1 > >::c\(\)](#), and [unlink\(\)](#).

Referenced by [unlink\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following files:

- [l4/re/namespace](#)
- [l4/re/impl/namespace_impl.h](#)

16.298 L4Re::Ned::Cmd_control Class Reference

Direct control interface for Ned.

```
#include <cmd_control>
```

Collaboration diagram for L4Re::Ned::Cmd_control:

L4Re::Ned::Cmd_control	
+	execute()
+	execute()

Public Member Functions

- long [execute](#) (L4::lpc::String<> cmd) noexcept
Execute the given Lua code.
- long [execute](#) (L4::lpc::String<> cmd, L4::lpc::String< char > *result) noexcept
Execute the given Lua code.

16.298.1 Detailed Description

Direct control interface for Ned.

Definition at line 19 of file [cmd_control](#).

16.298.2 Member Function Documentation

16.298.2.1 execute() [1/2]

```
long L4Re::Ned::Cmd_control::execute (
    L4::lpc::String<> cmd) [inline], [noexcept]
```

Execute the given Lua code.

Parameters

in	<i>cmd</i>	String with Lua code to execute.
----	------------	----------------------------------

Return values

<i>L4_EOK</i>	Code was successfully executed.
<i>-L4_EINVAL</i>	Code could not be parsed.
<i>-L4_EIO</i>	Error during code execution.

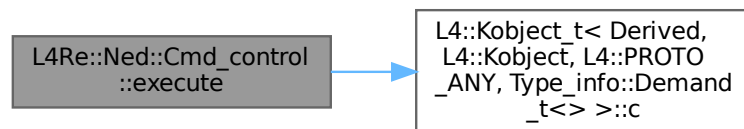
The code is executed using the global Lua state of ned which is retained between successive calls to execute. Thus you may define data in one call to execute and use it in a subsequent call.

This function does not return any results from the execution of the Lua code itself.

Definition at line 42 of file [cmd_control](#).

References [L4::Kobject_t< Derived, L4::Kobject, L4::PROTO_ANY, Type_info::Demand_t<> >::c\(\)](#).

Here is the call graph for this function:

**16.298.2.2 execute() [2/2]**

```

long L4Re::Ned::Cmd_control::execute (
    L4::Ipc::String<> cmd,
    L4::Ipc::String< char > * result) [inline], [noexcept]
  
```

Execute the given Lua code.

Parameters

in	<i>cmd</i>	String with Lua code to execute.
out	<i>result</i>	The first return value of the Lua code block as string.

Return values

<i>L4_EOK</i>	Code was successfully executed.
---------------	---------------------------------

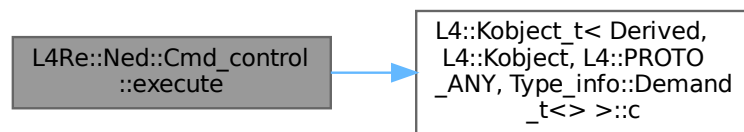
-L4_EINVAL	Code could not be parsed.
-L4_EIO	Error during code execution.

The code is executed using the global Lua state of ned which is retained between successive calls to execute. Thus you may define data in one call to execute and use it in a subsequent call.

Definition at line 64 of file [cmd_control](#).

References [L4::Kobject_t< Derived, L4::Kobject, L4::PROTO_ANY, Type_info::Demand_t<> >::c\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

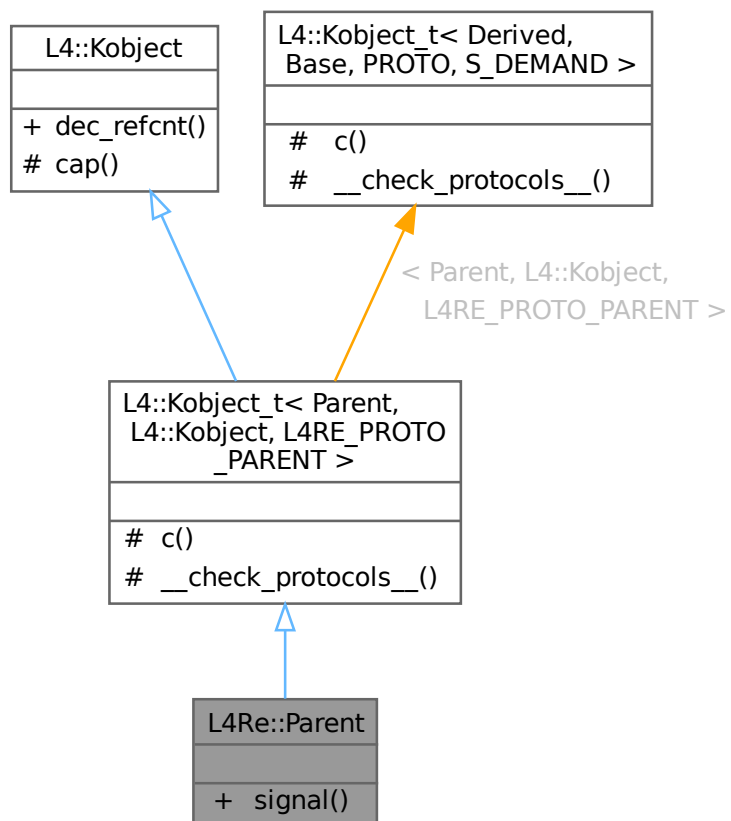
- `pkg/l4re-core/ned/lib/include/cmd_control`

16.299 L4Re::Parent Class Reference

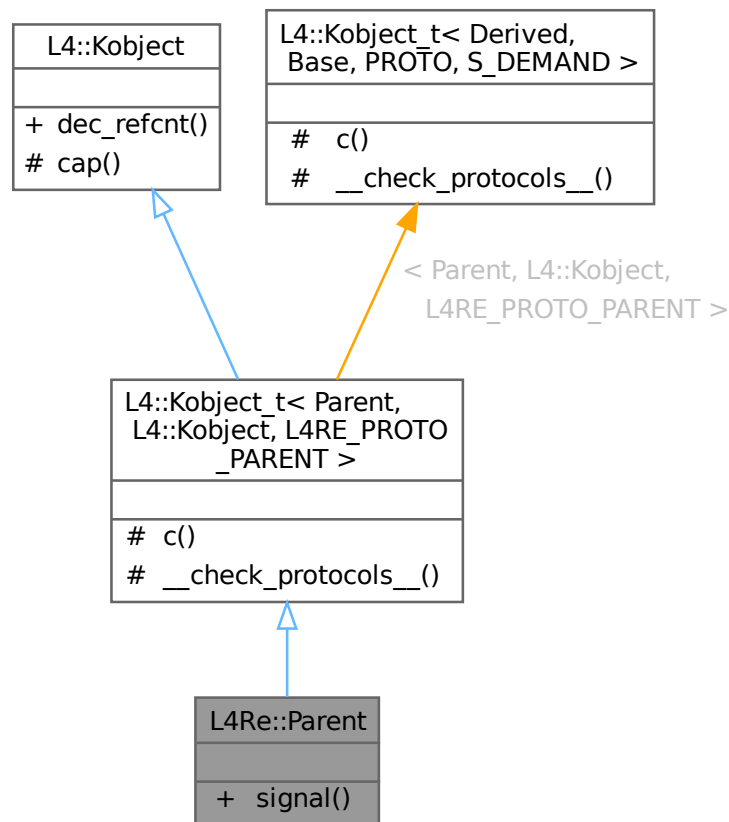
[Parent](#) interface.

```
#include <parent>
```

Inheritance diagram for L4Re::Parent:



Collaboration diagram for L4Re::Parent:



Public Member Functions

- long [signal](#) (unsigned long sig, unsigned long val)
Send a signal to the parent.

Public Member Functions inherited from [L4::Kobject](#)

- [l4_msgtag_t dec_refcnt](#) ([l4_mword_t](#) diff, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#))
Decrement the in kernel reference counter for the object.

Additional Inherited Members

Protected Types inherited from

[L4::Kobject_t < Parent, L4::Kobject, L4RE_PROTO_PARENT >](#)

- typedef Parent **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< PROTO, Parent > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< [__Iface](#) >, typename L4::Kobject::__Iface_list > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from [L4::Kobject_t< Parent, L4::Kobject, L4RE_PROTO_PARENT >](#)

- [L4::Cap< Class > c\(\)](#) const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from [L4::Kobject](#)

- [l4_cap_idx_t cap\(\)](#) const noexcept
Return capability selector.

Static Protected Member Functions inherited from [L4::Kobject_t< Parent, L4::Kobject, L4RE_PROTO_PARENT >](#)

- static void [__check_protocols__\(\)](#) noexcept
Helper to check for protocol conflicts.

16.299.1 Detailed Description

[Parent](#) interface.

See also

[Parent API](#) for more details about the purpose.

Definition at line 42 of file [parent](#).

16.299.2 Member Function Documentation

16.299.2.1 [signal\(\)](#)

```
long L4Re::Parent::signal (
    unsigned long sig,
    unsigned long val)
```

Send a signal to the parent.

Parameters

<i>sig</i>	Signal to send
<i>val</i>	Value of the signal

Return values

0	Success
<0	IPC error

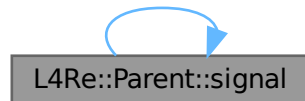
Note

The implementations of this interface in Moe and Ned only recognize the signal 0, in which case they will terminate the application from which the interface was invoked and not return. In this case, `val` is treated as the application's return code. For any other value of `sig`, the method just returns successfully.

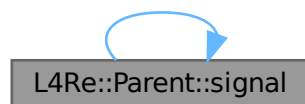
References [signal\(\)](#).

Referenced by [signal\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

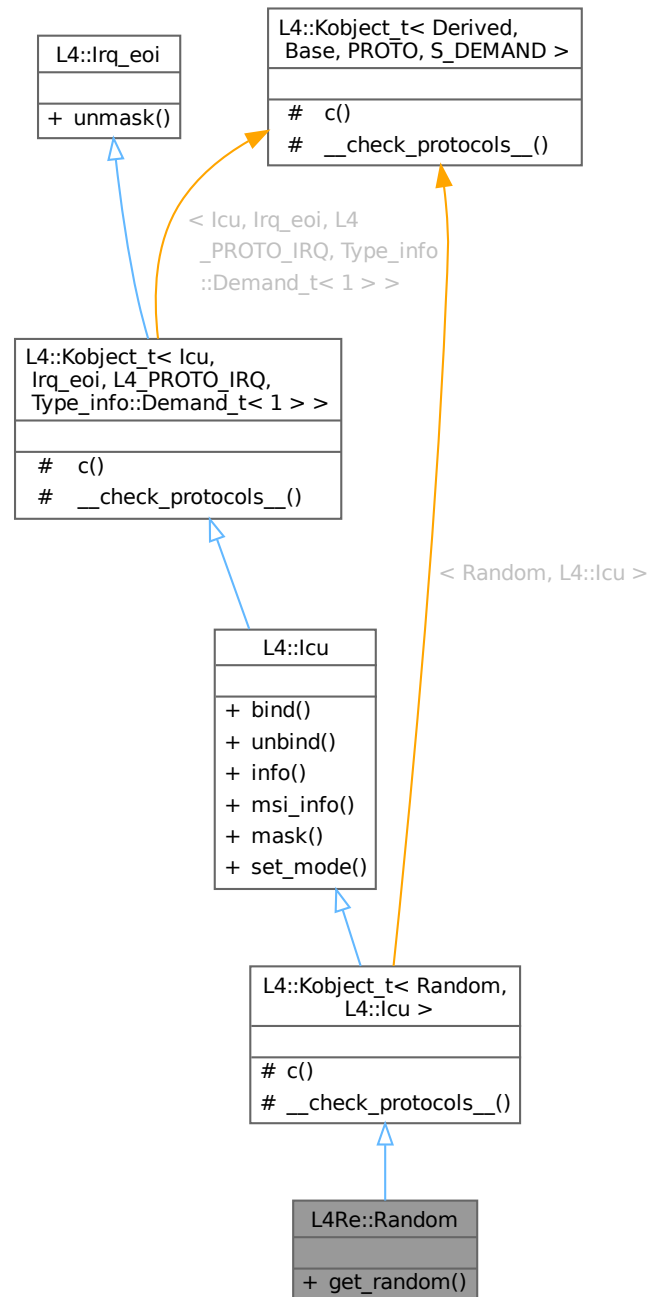
- [l4/re/parent](#)

16.300 L4Re::Random Struct Reference

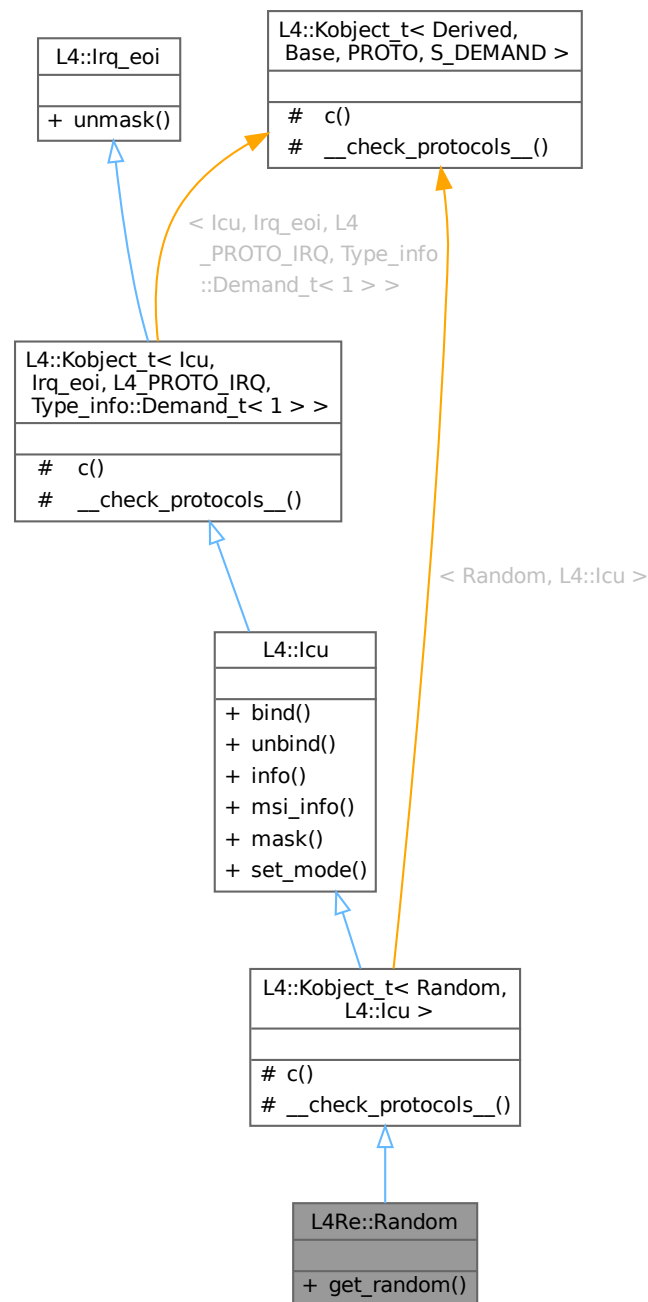
Low-bandwidth interface for random number generators.

```
#include <random>
```

Inheritance diagram for L4Re::Random:



Collaboration diagram for L4Re::Random:



Public Member Functions

- long `get_random` (`l4_size_t` size, `L4::lpc::Array< char, unsigned long > *buffer`)
Get a random number.

Public Member Functions inherited from L4::lcu

- `l4_msgtag_t bind` (unsigned irqnum, `L4::Cap< Triggerable > irq`, `l4_utcb_t *utcb=l4_utcb()`) noexcept

Bind an interrupt line of an interrupt controller to an interrupt object.

- [l4_msgtag_t unbind](#) (unsigned irqnum, [L4::Cap](#)< [Triggerable](#) > irq, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept

Remove binding of an interrupt line from the interrupt controller object.

- [l4_msgtag_t info](#) ([l4_icu_info_t](#) *info, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept

Get information about the ICU features.

- [l4_msgtag_t msi_info](#) ([l4_umword_t](#) irqnum, [l4_uint64_t](#) source, [l4_icu_msi_info_t](#) *msi_info)

Get MSI info about IRQ.

- [l4_msgtag_t mask](#) (unsigned irqnum, [l4_umword_t](#) *label=0, [l4_timeout_t](#) to=[L4_IPC_NEVER](#), [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept

Mask an IRQ line.

- [l4_msgtag_t set_mode](#) (unsigned irqnum, [l4_umword_t](#) mode, [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept

Set interrupt mode.

Public Member Functions inherited from [L4::lrq_eoi](#)

- [l4_msgtag_t unmask](#) (unsigned irqnum, [l4_umword_t](#) *label=0, [l4_timeout_t](#) to=[L4_IPC_NEVER](#), [l4_utcb_t](#) *utcb=[l4_utcb\(\)](#)) noexcept

Unmask the given interrupt line.

Additional Inherited Members

Protected Types inherited from [L4::Kobject_t](#)< [Random](#), [L4::lcu](#) >

- typedef [Random](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< [PROTO_ANY](#), [Random](#) > **__iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__iface** >, typename [L4::lcu](#)::__iface_list > **__iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Types inherited from [L4::Kobject_t](#)< [lcu](#), [lrq_eoi](#), [L4_PROTO_IRQ](#), [Type_info::Demand_t](#)< 1 > >

- typedef [lcu](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< [PROTO](#), [lcu](#) > **__iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__iface** >, typename [lrq_eoi](#)::__iface_list > **__iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from [L4::Kobject_t](#)< [Random](#), [L4::lcu](#) >

- [L4::Cap](#)< [Class](#) > **c** () const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from

[L4::Kobject_t< lcu, Irq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > >](#)

- [L4::Cap< Class > c \(\)](#) const noexcept

Get the capability to ourselves.

Static Protected Member Functions inherited from [L4::Kobject_t< Random, L4::lcu >](#)

- static void [__check_protocols__ \(\)](#) noexcept

Helper to check for protocol conflicts.

Static Protected Member Functions inherited from

[L4::Kobject_t< lcu, Irq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > >](#)

- static void [__check_protocols__ \(\)](#) noexcept

Helper to check for protocol conflicts.

16.300.1 Detailed Description

Low-bandwidth interface for random number generators.

The interface offers an ICU interface where a client can register an interrupt to get notified when entropy is available. Support for notifications is optional. If a service does not implement notification, it must return 0 for the number of interrupts in the [info\(\)](#) call. The notification interrupt must have index 0.

Include File

```
#include <l4/re/random>
```

Definition at line 33 of file [random](#).

16.300.2 Member Function Documentation

16.300.2.1 [get_random\(\)](#)

```
long L4Re::Random::get_random (
    l4_size_t size,
    L4::Ipc::Array< char, unsigned long > * buffer)
```

Get a random number.

Parameters

	<i>size</i>	Number of bytes of entropy requested.
out	<i>buffer</i>	Buffer containing the random number. Each byte in the buffer contains 8 bits of randomness.

Return values

≥ 0	Actual size of the returned random number in bytes. This may be less than the requested size. The return value may also be 0 if temporarily no entropy is available.
$-L4_EIO$	Source of randomness permanently unavailable.
< 0	IPC error.

This function should never block. It should immediately return as much entropy as is available. If the call returns less than the requested bytes and a notification interrupt was installed, then the service triggers an interrupt as soon as the remaining entropy is available. That means that when an interrupt is triggered, the service must guarantee that the next call to [get_random\(\)](#) returns at least the number of missing bytes for the call that initially triggered the notification.

If [get_random\(\)](#) is called while a notification is pending, then the behaviour is implementation-defined.

The documentation for this struct was generated from the following file:

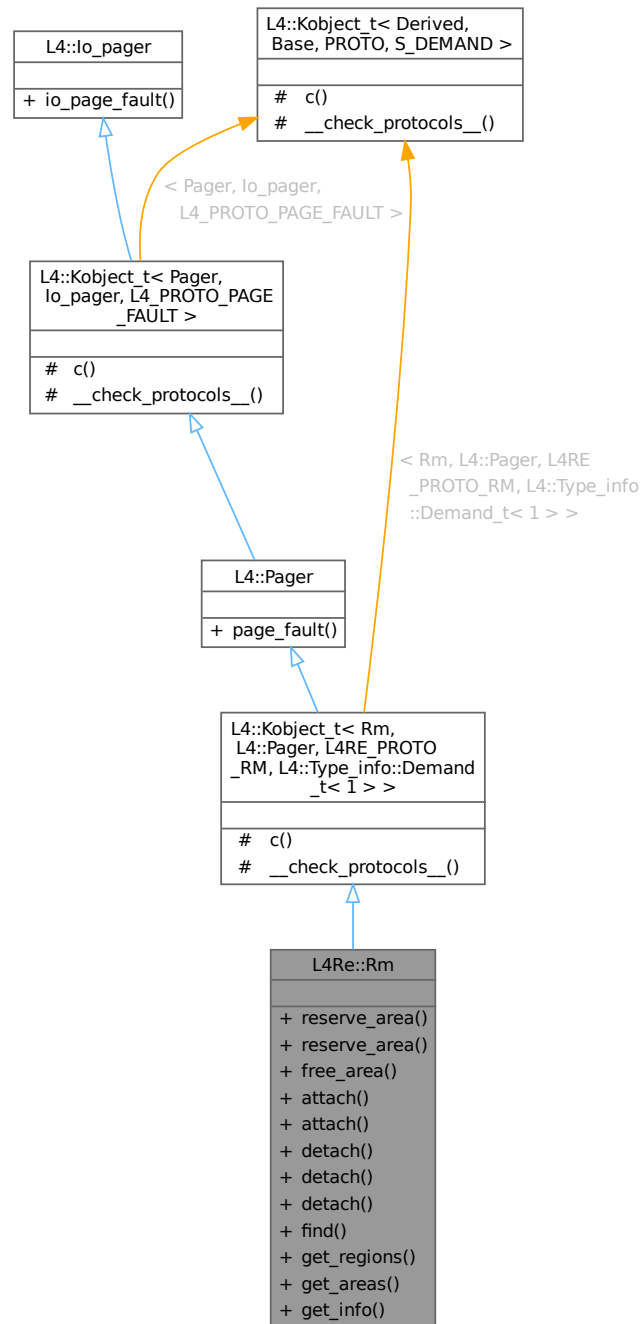
- [l4/re/random](#)

16.301 L4Re::Rm Class Reference

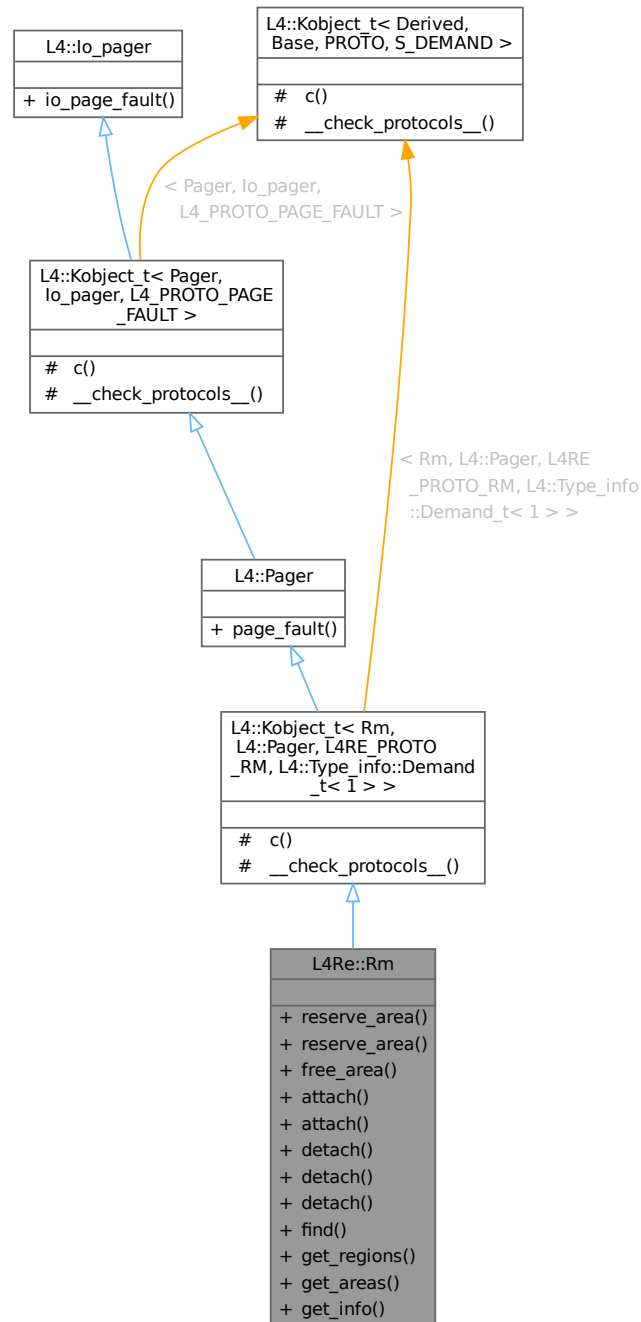
[Region](#) map.

```
#include <l4/re/rm>
```

Inheritance diagram for L4Re::Rm:



Collaboration diagram for L4Re::Rm:



Data Structures

- struct **F**
Rm flags definitions.
- class **Unique_region**
Unique region.
- struct **Region**

A region is a range of virtual addresses which is backed by content.

- struct [Area](#)

An area is a range of virtual addresses which is reserved, see [L4Re::Rm::reserve_area\(\)](#).

Public Types

- enum [Detach_result](#) { [Detached_ds](#) = 0 , [Kept_ds](#) = 1 , [Split_ds](#) = 2 , [Detach_result_mask](#) = 3 , [Detach_again](#) = 4 }
- Result values for detach operation.*
- enum [Region_flag_shifts](#) { [Caching_shift](#) = Dataspace::F::Caching_shift }
- Region flag shifts.*
- enum [Detach_flags](#) { [Detach_exact](#) = 1 , [Detach_overlap](#) = 2 , [Detach_keep](#) = 4 }
- Flags for detach operation.*

Public Member Functions

- long [reserve_area](#) ([l4_addr_t](#) *start, unsigned long size, Flags flags=[Flags\(0\)](#), unsigned char align=[L4_PAGESHIFT](#)) const noexcept
- Reserve the given area in the region map.*
- template<typename T>
long [reserve_area](#) (T **start, unsigned long size, Flags flags=[Flags\(0\)](#), unsigned char align=[L4_PAGESHIFT](#)) const noexcept
- Reserve the given area in the region map.*
- long [free_area](#) ([l4_addr_t](#) addr)
- Free an area from the region map.*
- long [attach](#) ([l4_addr_t](#) *start, unsigned long size, Flags flags, [L4::lpc::Cap](#)< [Dataspace](#) > mem, Offset offs=0, unsigned char align=[L4_PAGESHIFT](#), [L4::Cap](#)< [L4::Task](#) > const task=[L4::Cap](#)< [L4::Task](#) >::Invalid, char const *name=nullptr, Offset backing_offset=0) const noexcept
- Attach a data space to a region.*
- template<typename T>
long [attach](#) (T **start, unsigned long size, Flags flags, [L4::lpc::Cap](#)< [Dataspace](#) > mem, Offset offs=0, unsigned char align=[L4_PAGESHIFT](#), [L4::Cap](#)< [L4::Task](#) > const task=[L4::Cap](#)< [L4::Task](#) >::Invalid, char const *name=nullptr, Offset backing_offset=0) const noexcept
- Attach a data space to a region.*
- int [detach](#) ([l4_addr_t](#) addr, [L4::Cap](#)< [Dataspace](#) > *mem, [L4::Cap](#)< [L4::Task](#) > const &task=[This_task](#)) const noexcept
- Detach and unmap a region from the address space.*
- int [detach](#) (void *addr, [L4::Cap](#)< [Dataspace](#) > *mem, [L4::Cap](#)< [L4::Task](#) > const &task=[This_task](#)) const noexcept
- Detach and unmap a region from the address space.*
- int [detach](#) ([l4_addr_t](#) start, unsigned long size, [L4::Cap](#)< [Dataspace](#) > *mem, [L4::Cap](#)< [L4::Task](#) > const &task) const noexcept
- Detach and unmap all parts of the regions within the specified interval.*
- long [find](#) ([l4_addr_t](#) *addr, unsigned long *size, Offset *offset, [L4Re::Rm::Flags](#) *flags, [L4::Cap](#)< [Dataspace](#) > *m) noexcept
- Find a region given an address and size.*
- long [get_regions](#) ([l4_addr_t](#) start, [L4::lpc::Ret_array](#)< [Region](#) > regions)
- Return the list of regions whose starting addresses are higher or equal to start in the address space managed by this region map.*
- long [get_areas](#) ([l4_addr_t](#) start, [L4::lpc::Ret_array](#)< [Area](#) > areas)
- Return the list of areas whose starting addresses are higher or equal to start in the address space managed by this region map.*
- long [get_info](#) ([l4_addr_t](#) addr, [L4::lpc::String](#)< char > &name, Offset &backing_offset)
- Return auxiliary information of a region.*

Public Member Functions inherited from [L4::Pager](#)

- [l4_msgtag_t page_fault](#) ([l4_umword_t](#) pfa, [l4_umword_t](#) pc, [L4::lpc::Rcv_fpage](#) rwin, [L4::lpc::Opt<L4::lpc::Snd_fpage & >](#) fp)

Page-fault protocol message.

Public Member Functions inherited from [L4::lo_pager](#)

- [l4_msgtag_t io_page_fault](#) ([l4_fpage_t](#) io_pfa, [l4_umword_t](#) pc, [L4::lpc::Rcv_fpage](#) rwin, [L4::lpc::Opt<L4::lpc::Snd_fpage & >](#) fp)

IO page fault protocol message.

Additional Inherited Members

Protected Types inherited from

[L4::Kobject_t< Rm, L4::Pager, L4RE_PROTO_RM, L4::Type_info::Demand_t< 1 > >](#)

- typedef [Rm](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< PROTO, [Rm](#) > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__Iface** >, typename [L4::Pager::__Iface_list](#) > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Types inherited from

[L4::Kobject_t< Pager, lo_pager, L4_PROTO_PAGE_FAULT >](#)

- typedef [Pager](#) **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< PROTO, [Pager](#) > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__Iface** >, typename [lo_pager::__Iface_list](#) > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from

[L4::Kobject_t< Rm, L4::Pager, L4RE_PROTO_RM, L4::Type_info::Demand_t< 1 > >](#)

- [L4::Cap< Class > c](#) () const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from

[L4::Kobject_t< Pager, lo_pager, L4_PROTO_PAGE_FAULT >](#)

- [L4::Cap< Class > c](#) () const noexcept
Get the capability to ourselves.

Static Protected Member Functions inherited from**[L4::Kobject_t< Rm, L4::Pager, L4RE_PROTO_RM, L4::Type_info::Demand_t< 1 > >](#)**

- static void **__check_protocols__** () noexcept

*Helper to check for protocol conflicts.***Static Protected Member Functions inherited from****[L4::Kobject_t< Pager, lo_pager, L4_PROTO_PAGE_FAULT >](#)**

- static void **__check_protocols__** () noexcept

*Helper to check for protocol conflicts.***16.301.1 Detailed Description**[Region](#) map.

See also

[Region map API](#) .Definition at line [81](#) of file [rm](#).**16.301.2 Member Enumeration Documentation****16.301.2.1 Detach_flags**enum [L4Re::Rm::Detach_flags](#)

Flags for detach operation.

Enumerator

Detach_exact	Do an unmap of the exact region given.
Detach_overlap	Do an unmap of all overlapping regions.
Detach_keep	Do not free the detached data space, ignore the F::Detach_free .

Definition at line [221](#) of file [rm](#).**16.301.2.2 Detach_result**enum [L4Re::Rm::Detach_result](#)

Result values for detach operation.

Enumerator

Detached_ds	Detached data sapce.
Kept_ds	Kept data space.
Split_ds	Splitted data space, and done.
Detach_again	Detached data space, more to do.

Definition at line 89 of file [rm](#).

16.301.2.3 Region_flag_shifts

```
enum L4Re::Rm::Region_flag_shifts
```

[Region](#) flag shifts.

Enumerator

Caching_shift	Start of Rm cache bits.
---------------	---

Definition at line 101 of file [rm](#).

16.301.3 Member Function Documentation

16.301.3.1 attach() [1/2]

```
long L4Re::Rm::attach (
    l4_addr_t * start,
    unsigned long size,
    Rm::Flags flags,
    L4::Ipc::Cap< Dataspace > mem,
    Rm::Offset offs = 0,
    unsigned char align = L4_PAGESHIFT,
    L4::Cap< L4::Task > const task = L4::Cap<L4::Task>::Invalid,
    char const * name = nullptr,
    Rm::Offset backing_offset = 0) const [noexcept]
```

Attach a data space to a region.

Parameters

in, out	<i>start</i>	Virtual start address where the region manager shall attach the data space. Will be rounded down to the nearest start of a page. If L4Re::Rm::F::Search_addr is given this value is used as the start address to search for a free virtual memory region and the resulting address is returned here. If L4Re::Rm::F::In_area is given the value is used as a selector for the area (see L4Re::Rm::reserve_area) to attach the data space to.
	<i>size</i>	Size of the data space to attach (in bytes). Will be rounded up to the nearest multiple of the page size.

	<i>flags</i>	The flags control how and with which rights the dataspace is attached to the region. See L4Re::Rm::F::Attach_flags and L4Re::Rm::F::Region_flags . The caller must specify the desired rights of the attached region explicitly. The default set of rights is empty. If the F::Eager_map flag is set this function may also return L4Re::Dataspace::map error codes if the mapping fails.
	<i>mem</i>	Data space.
	<i>offs</i>	Offset into the data space to use.
	<i>align</i>	Alignment of the virtual region, log2-size, default: a page (L4_PAGESHIFT). This is only meaningful if the L4Re::Rm::F::Search_addr flag is used.
	<i>task</i>	Optional destination task of mapping if F::Eager_map flag was passed. If invalid, the mapping is established in the current task. This parameter is only useful if the region manager is for a foreign task.
	<i>name</i>	Optional name of the region.
	<i>backing_offset</i>	Optional value describing an offset into the backing store of this region.

Return values

0	Success
-L4_ENOENT	No area could be found (see L4Re::Rm::F::In_area)
-L4_EPERM	Operation not allowed.
-L4_EINVAL	
-L4_EADDRNOTAVAIL	The given address is not available.
<0	IPC errors

Makes the whole or parts of a data space visible in the virtual memory of the corresponding task. The corresponding region in the virtual address space is backed with the contents of the dataspace.

Note

When searching for a free place in the virtual address space, the space between *start* and the end of the virtual address space is searched.

There is no region object created, instead the region is defined by a virtual address within this range (see [L4Re::Rm::find](#)).

Definition at line 34 of file [rm_impl.h](#).

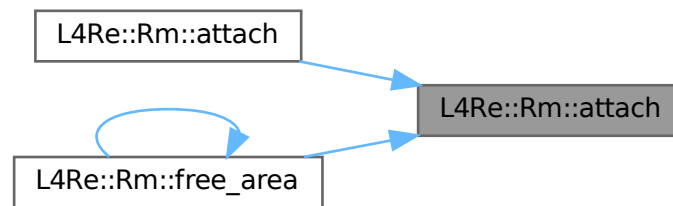
References [L4::Kobject_t< Rm, L4::Pager, L4RE_PROTO_RM, L4::Type_info::Demand_t< 1 > >::c\(\)](#), [L4Re::Rm::F::Eager_map](#), [L4Re::Rm::F::Kernel](#), [L4Re::Rm::F::No_eager_map](#), [L4Re::Rm::F::Reserved](#), and [L4Re::Rm::F::Rights_mask](#).

Referenced by [attach\(\)](#), and [free_area\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.301.3.2 attach() [2/2]

```

template<typename T>
long L4Re::Rm::attach (
    T ** start,
    unsigned long size,
    Flags flags,
    L4::Ipc::Cap< Dataspace > mem,
    Offset offs = 0,
    unsigned char align = L4_PAGESHIFT,
    L4::Cap< L4::Task > const task = L4::Cap<L4::Task>::Invalid,
    char const * name = nullptr,
    Offset backing_offset = 0) const [inline], [noexcept]

```

Attach a data space to a region.

Parameters

<i>in, out</i>	<i>start</i>	Virtual start address where the region manager shall attach the data space. Will be rounded down to the nearest start of a page. If L4Re::Rm::F::Search_addr is given this value is used as the start address to search for a free virtual memory region and the resulting address is returned here. If L4Re::Rm::F::In_area is given the value is used as a selector for the area (see L4Re::Rm::reserve_area) to attach the data space to.
	<i>size</i>	Size of the data space to attach (in bytes). Will be rounded up to the nearest multiple of the page size.
	<i>flags</i>	The flags control how and with which rights the dataspace is attached to the region. See L4Re::Rm::F::Attach_flags and L4Re::Rm::F::Region_flags . The caller must specify the desired rights of the attached region explicitly. The default set of rights is empty. If the F::Eager_map flag is set this function may also return L4Re::Dataspace::map error codes if the mapping fails.
	<i>mem</i>	Data space.
	<i>offs</i>	Offset into the data space to use.
	<i>align</i>	Alignment of the virtual region, log2-size, default: a page (L4_PAGESHIFT). This is only meaningful if the L4Re::Rm::F::Search_addr flag is used.

	<i>task</i>	Optional destination task of mapping if <code>F::Eager_map</code> flag was passed. If invalid, the mapping is established in the current task. This parameter is only useful if the region manager is for a foreign task.
	<i>name</i>	Optional name of the region.
	<i>backing_offset</i>	Optional value describing an offset into the backing store of this region.

Return values

<code>0</code>	Success
<code>-L4_ENOENT</code>	No area could be found (see <code>L4Re::Rm::F::In_area</code>)
<code>-L4_EPERM</code>	Operation not allowed.
<code>-L4_EINVAL</code>	
<code>-L4_EADDRNOTAVAIL</code>	The given address is not available.
<code><0</code>	IPC errors

Makes the whole or parts of a data space visible in the virtual memory of the corresponding task. The corresponding region in the virtual address space is backed with the contents of the dataspace.

Note

When searching for a free place in the virtual address space, the space between *start* and the end of the virtual address space is searched.

There is no region object created, instead the region is defined by a virtual address within this range (see `L4Re::Rm::find`).

Definition at line 409 of file `rm`.

References `attach()`, and `L4_PAGESHIFT`.

Here is the call graph for this function:



16.301.3.3 detach() [1/3]

```

int L4Re::Rm::detach (
    l4_addr_t addr,
    L4::Cap< Dataspace > * mem,
    L4::Cap< L4::Task > const & task = This_task) const [inline], [noexcept]
  
```

Detach and unmap a region from the address space.

Parameters

	<i>addr</i>	Virtual address of region, any address within the region is valid.
out	<i>mem</i>	Dataspace that is affected. Give 0 if not interested.
	<i>task</i>	This argument specifies the task where the pages are unmapped. Provide L4::Cap<L4::Task>::Invalid for none. The default is the current task.

Return values

L4Re::Rm::Detach_result	On success.
-L4_ENOENT	No region found.
<0	IPC errors

Frees a region in the virtual address space given by *addr* (address type). The corresponding part of the address space is now available again.

Definition at line 765 of file [rm](#).

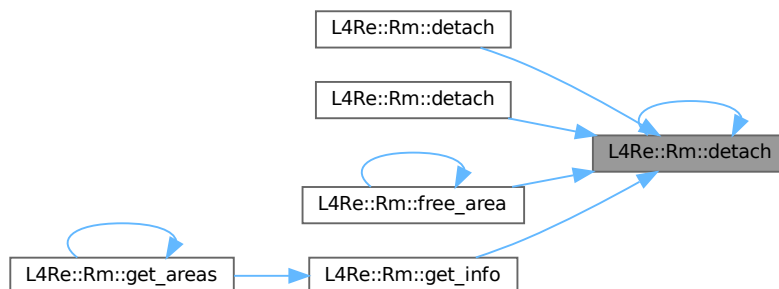
References [detach\(\)](#), and [Detach_overlap](#).

Referenced by [detach\(\)](#), [detach\(\)](#), [detach\(\)](#), [free_area\(\)](#), and [get_info\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.301.3.4 detach() [2/3]

```
int L4Re::Rm::detach (
    l4_addr_t start,
    unsigned long size,
    L4::Cap< Dataspace > * mem,
    L4::Cap< L4::Task > const & task) const [inline], [noexcept]
```

Detach and unmap all parts of the regions within the specified interval.

Parameters

	<i>start</i>	Start of area to detach, must be within region.
	<i>size</i>	Size of of area to detach (in bytes).
out	<i>mem</i>	Dataspace that is affected. Give 0 if not interested.
	<i>task</i>	This argument specifies the task where the pages are unmapped. Provide L4::Cap<L4::Task>::Invalid for none. The default is the current task.

Return values

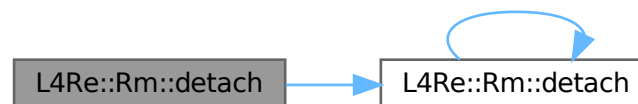
L4Re::Rm::Detach_result	On success.
-L4_ENOENT	No region found.
<0	IPC errors

Frees all regions within the interval given by start and size. If a region overlaps the start or the end of the interval this region is only detached partly. If the interval is within one region the original region is split up into two separate regions.

Definition at line [778](#) of file [rm](#).

References [detach\(\)](#), and [Detach_exact](#).

Here is the call graph for this function:



16.301.3.5 detach() [3/3]

```
int L4Re::Rm::detach (
    void * addr,
    L4::Cap< Dataspace > * mem,
    L4::Cap< L4::Task > const & task = This_task) const [inline], [noexcept]
```

Detach and unmap a region from the address space.

Parameters

	<i>addr</i>	Virtual address of region, any address within the region is valid.
out	<i>mem</i>	Dataspace that is affected. Give 0 if not interested.
	<i>task</i>	This argument specifies the task where the pages are unmapped. Provide L4::Cap<L4::Task>::Invalid for none. The default is the current task.

Return values

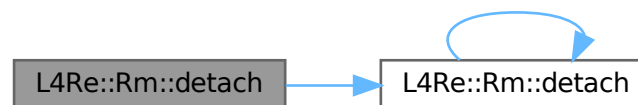
L4Re::Rm::Detach_result	On success.
-L4_ENOENT	No region found.
<0	IPC errors

Frees a region in the virtual address space given by *addr* (address type). The corresponding part of the address space is now available again.

Definition at line 770 of file [rm](#).

References [detach\(\)](#), and [Detach_overlap](#).

Here is the call graph for this function:



16.301.3.6 find()

```

long L4Re::Rm::find (
    l4\_addr\_t * addr,
    unsigned long * size,
    Offset * offset,
    L4Re::Rm::Flags * flags,
    L4::Cap< Dataspace > \* m) [inline], [noexcept]

```

Find a region given an address and size.

Parameters

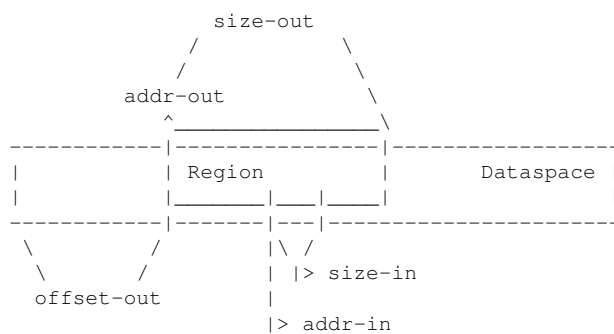
in, out	<i>addr</i>	Address to look for. Returns the start address of the found region.
in, out	<i>size</i>	Size of the area to look for (in bytes). Returns the size of the found region (in bytes).
out	<i>offset</i>	Offset at the beginning of the region within the associated dataspace.

out	<i>flags</i>	Region flags, see F::Region_flags (and F::In_area).
out	<i>m</i>	Associated dataspace or paging service.

Return values

0	Success
-L4_EPERM	Operation not allowed.
-L4_ENOENT	No region found.
<0	IPC errors

This function returns the properties of the region that contains the area described by the `addr` and `size` parameter. If no such region is found but a reserved area, the area is returned and [F::In_area](#) is set in `flags`. Note, in the case of an area the `offset` and `m` return values are invalid.



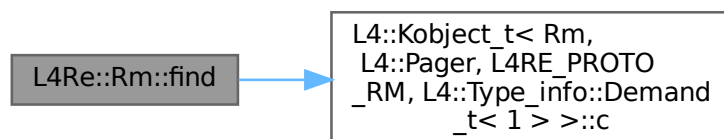
Note

The value of the `size` input parameter should be 1 to assure that a region can be determined unambiguously.

Definition at line 672 of file [rm](#).

References [L4::Kobject_t< Rm, L4::Pager, L4RE_PROTO_RM, L4::Type_info::Demand_t< 1 > >::c\(\)](#).

Here is the call graph for this function:



16.301.3.7 free_area()

```
long L4Re::Rm::free_area (  
    l4_addr_t addr)
```

Free an area from the region map.

Parameters

<i>addr</i>	An address within the area to free.
-------------	-------------------------------------

Return values

0	Success
-L4_ENOENT	No area found.
<0	IPC errors

Note

The data spaces that are attached to that area are not detached by this operation.

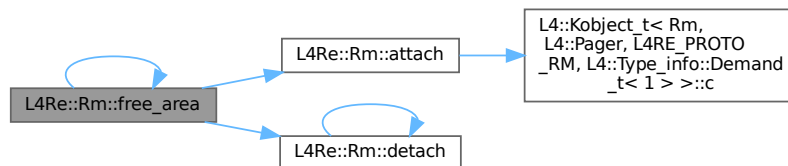
See also

[reserve_area\(\)](#) for more information about areas.

References [attach\(\)](#), [detach\(\)](#), [free_area\(\)](#), [L4::Cap_base::Invalid](#), and [L4_PAGESHIFT](#).

Referenced by [free_area\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.301.3.8 get_areas()

```
long L4Re::Rm::get_areas (
    l4_addr_t start,
    L4::Ipc::Ret_array< Area > areas)
```

Return the list of areas whose starting addresses are higher or equal to `start` in the address space managed by this region map.

Parameters

	<i>start</i>	Virtual address from where to start searching.
out	<i>areas</i>	List of areas found in this region map.

Return values

≥ 0	Number of returned areas in the <i>areas</i> array.
< 0	IPC errors

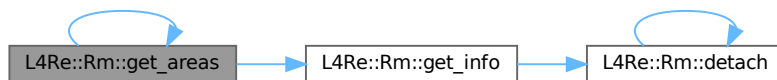
Note

The returned list of areas might not be complete and the caller shall use the function repeatedly with a start address one larger than the end address of the last area from the previous call.

References [get_areas\(\)](#), and [get_info\(\)](#).

Referenced by [get_areas\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.301.3.9 get_info()

```

long L4Re::Rm::get_info (
    l4_addr_t addr,
    L4::Ipc::String< char > & name,
    Offset & backing_offset)

```

Return auxiliary information of a region.

This is a debugging feature and might not be available.

Parameters

	<i>addr</i>	Virtual address of the region.
out	<i>name</i>	Name of the region.
out	<i>backing_offset</i>	Backing offset information.

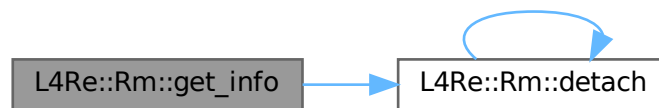
Return values

0	Success
-L4_ENOENT	Region not found.
-L4_ENOSYS	Function not available.
<0	IPC errors

References [detach\(\)](#).

Referenced by [get_areas\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.301.3.10 get_regions()

```

long L4Re::Rm::get_regions (
    l4_addr_t start,
    L4::Ipc::Ret_array< Region > regions)
  
```

Return the list of regions whose starting addresses are higher or equal to `start` in the address space managed by this region map.

Parameters

	<i>start</i>	Virtual address from where to start searching.
out	<i>regions</i>	List of regions found in this region map.

Return values

≥ 0	Number of returned regions in the <code>regions</code> array.
< 0	IPC errors

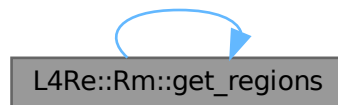
Note

The returned list of regions might not be complete and the caller shall use the function repeatedly with a start address one larger than the end address of the last region from the previous call.

References [get_regions\(\)](#).

Referenced by [get_regions\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.301.3.11 `reserve_area()` [1/2]

```
long L4Re::Rm::reserve_area (  
    14_addr_t * start,  
    unsigned long size,
```

```
Flags flags = Flags(0),  
unsigned char align = L4_PAGESHIFT) const [inline], [noexcept]
```

Reserve the given area in the region map.

Parameters

<i>in, out</i>	<i>start</i>	The virtual start address of the area to reserve. Returns the start address of the area.
	<i>size</i>	The size of the area to reserve (in bytes).
	<i>flags</i>	Flags for the reserved area (see L4Re::Rm::F::Region_flags and L4Re::Rm::F::Attach_flags).
	<i>align</i>	Alignment of area if searched as bits (log2 value).

Return values

0	Success
-L4_EADDRNOTAVAIL	The given area cannot be reserved.
<0	IPC errors

This function reserves an area within the virtual address space managed by the region map. There are two kinds of areas available:

- Reserved areas (*flags* = [L4Re::Rm::F::Reserved](#)), where no data spaces can be attached
- Special purpose areas (*flags* = 0), where data spaces can be attached to the area via the [L4Re::Rm::F::In_area](#) flag and a start address within the area itself.

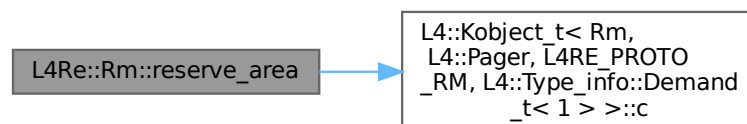
Note

When searching for a free place in the virtual address space (with *flags* = [L4Re::Rm::F::Search_addr](#)), the space between *start* and the end of the virtual address space is searched.

Definition at line 280 of file [rm](#).

References [L4::Kobject_t< Rm, L4::Pager, L4RE_PROTO_RM, L4::Type_info::Demand_t< 1 > >::c\(\)](#), and [L4_PAGESHIFT](#).

Here is the call graph for this function:



16.301.3.12 reserve_area() [2/2]

```
template<typename T>
long L4Re::Rm::reserve_area (
    T ** start,
    unsigned long size,
    Flags flags = Flags(0),
    unsigned char align = L4_PAGESHIFT) const [inline], [noexcept]
```

Reserve the given area in the region map.

Parameters

<i>in, out</i>	<i>start</i>	The virtual start address of the area to reserve. Returns the start address of the area.
	<i>size</i>	The size of the area to reserve (in bytes).
	<i>flags</i>	Flags for the reserved area (see F::Region_flags and F::Attach_flags).
	<i>align</i>	Alignment of area if searched as bits (log2 value).

Return values

0	Success
-L4_EADDRNOTAVAIL	The given area cannot be reserved.
<0	IPC errors

For more information, please refer to the analogous function

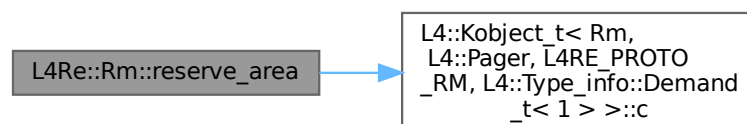
See also

[L4Re::Rm::reserve_area](#).

Definition at line 306 of file [rm](#).

References [L4::Kobject_t< Rm, L4::Pager, L4RE_PROTO_RM, L4::Type_info::Demand_t< 1 > >::c\(\)](#), [L4_PAGESHIFT](#), and

Here is the call graph for this function:



The documentation for this class was generated from the following files:

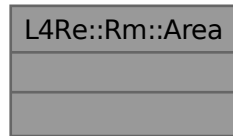
- [l4/re/rm](#)
- [l4/re/impl/rm_impl.h](#)

16.302 L4Re::Rm::Area Struct Reference

An area is a range of virtual addresses which is reserved, see [L4Re::Rm::reserve_area\(\)](#).

```
#include <rm>
```

Collaboration diagram for L4Re::Rm::Area:



16.302.1 Detailed Description

An area is a range of virtual addresses which is reserved, see [L4Re::Rm::reserve_area\(\)](#).

See also

[Region map API](#)

Definition at line [699](#) of file [rm](#).

The documentation for this struct was generated from the following file:

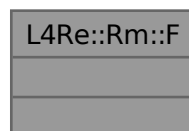
- [l4/re/rm](#)

16.303 L4Re::Rm::F Struct Reference

[Rm](#) flags definitions.

```
#include <rm>
```

Collaboration diagram for L4Re::Rm::F:



Public Types

- enum [Attach_flags](#) : l4_uint32_t {
[Search_addr](#) = 0x20000 , [In_area](#) = 0x40000 , [Eager_map](#) = 0x80000 , [No_eager_map](#) = 0x100000 ,
[Attach_mask](#) = 0x1f0000 }
Flags for attach operation.
- enum [Region_flags](#) : l4_uint16_t {
[Rights_mask](#) = 0x0f , [R](#) = Dataspace::F::R , [W](#) = Dataspace::F::W , [X](#) = Dataspace::F::X ,
[RW](#) = Dataspace::F::RW , [RX](#) = Dataspace::F::RX , [RWX](#) = Dataspace::F::RWX , [Kernel](#) = 0x100 ,
[Detach_free](#) = 0x200 , [Pager](#) = 0x400 , [Reserved](#) = 0x800 , [Caching_mask](#) = Dataspace::F::Caching_mask ,
[Cache_normal](#) = Dataspace::F::Normal , [Cache_buffered](#) = Dataspace::F::Bufferable , [Cache_uncached](#) =
Dataspace::F::Uncacheable , [Ds_map_mask](#) = 0xff ,
[Region_flags_mask](#) = 0xffff }
Region flags (permissions, cacheability, special).

16.303.1 Detailed Description

[Rm](#) flags definitions.

Definition at line [108](#) of file [rm](#).

16.303.2 Member Enumeration Documentation

16.303.2.1 Attach_flags

```
enum L4Re::Rm::F::Attach_flags : l4_uint32_t
```

Flags for attach operation.

Enumerator

Search_addr	Search for a suitable address range.
In_area	Search only in area, or map into area.
Eager_map	Eagerly map the attached data space in.
No_eager_map	Prevent eager mapping of the attached data space.
Attach_mask	Mask of all attach flags.

Definition at line [111](#) of file [rm](#).

16.303.2.2 Region_flags

```
enum L4Re::Rm::F::Region_flags : l4_uint16_t
```

[Region](#) flags (permissions, cacheability, special).

Enumerator

Rights_mask	Region rights.
R	Readable region.
W	Writable region.
X	Executable region.
RW	Readable and writable region.
RX	Readable and executable region.
RWX	Readable, writable and executable region.
Kernel	Kernel-provided memory (KUMEM).
Detach_free	Free the portion of the data space after detach.
Pager	Region has a pager.
Reserved	Region is reserved (blocked).
Caching_mask	Mask of all Rm cache bits.
Cache_normal	Cache bits for normal cacheable memory. This is the default if no other cache-related flag was specified.
Cache_buffered	Cache bits for buffered (write combining) memory.
Cache_uncached	Cache bits for uncached memory.
Ds_map_mask	Mask for all bits for cache options and rights.
Region_flags_mask	Mask of all region flags.

Definition at line 128 of file [rm](#).

The documentation for this struct was generated from the following file:

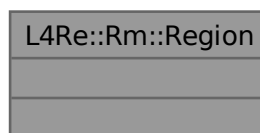
- [l4/re/rm](#)

16.304 L4Re::Rm::Region Struct Reference

A region is a range of virtual addresses which is backed by content.

```
#include <rm>
```

Collaboration diagram for L4Re::Rm::Region:



16.304.1 Detailed Description

A region is a range of virtual addresses which is backed by content.

See also

[Region map API](#)

Definition at line 686 of file [rm](#).

The documentation for this struct was generated from the following file:

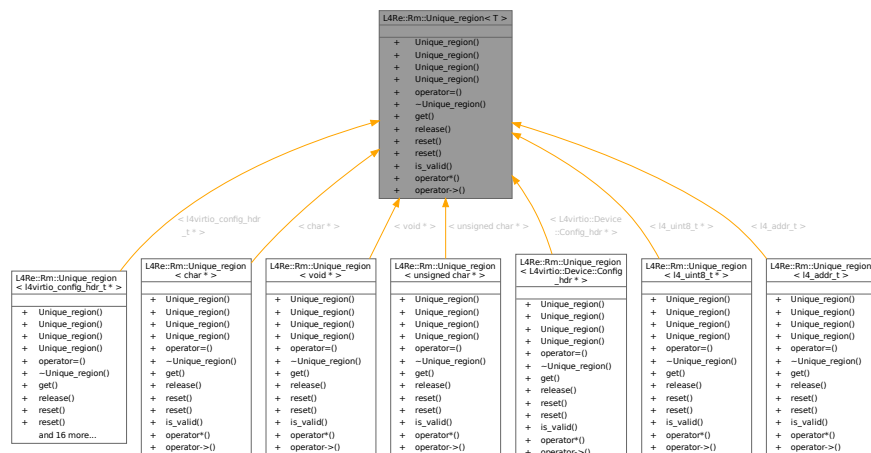
- [l4/re/rm](#)

16.305 L4Re::Rm::Unique_region< T > Class Template Reference

Unique region.

```
#include <rm>
```

Inheritance diagram for L4Re::Rm::Unique_region< T >:



Collaboration diagram for L4Re::Rm::Unique_region< T >:

L4Re::Rm::Unique_region< T >	
+	Unique_region()
+	Unique_region()
+	Unique_region()
+	Unique_region()
+	operator=()
+	~Unique_region()
+	get()
+	release()
+	reset()
+	reset()
+	is_valid()
+	operator*()
+	operator->()

Public Member Functions

- **Unique_region** () noexcept
Construct an invalid [Unique_region](#).
- **Unique_region** (T addr) noexcept
Construct a [Unique_region](#) from an address.
- **Unique_region** (T addr, L4::Cap< Rm > const &rm) noexcept
Construct a valid [Unique_region](#) from an address and a region manager.
- **Unique_region** (Unique_region &&o) noexcept
Move-Construct a [Unique_region](#).
- **Unique_region & operator=** (Unique_region &&o) noexcept
Move-assign a [Unique_region](#).
- **~Unique_region** () noexcept
Destructor.
- T **get** () const noexcept
Return the address.
- T **release** () noexcept
Return the address and invalidate the [Unique_region](#).
- void **reset** (T addr, L4::Cap< Rm > const &rm) noexcept
Set new address and region manager.
- void **reset** () noexcept
Make the [Unique_region](#) invalid.
- bool **is_valid** () const noexcept
Check if the [Unique_region](#) is valid.

- **T operator*** () const noexcept
Dereference the address.
- **T operator->** () const noexcept
Member access for the address.

16.305.1 Detailed Description

```
template<typename T>
class L4Re::Rm::Unique_region< T >
```

Unique region.

Capture a single region with automatic detach on destruction and unique ownership. Stores the start address and the region-mapper capability internally. A unique region is valid precisely if the internal region-mapper capability is valid. The features for unique ownership and automatic detach are only active for valid unique regions.

Definition at line [435](#) of file [rm](#).

16.305.2 Constructor & Destructor Documentation

16.305.2.1 Unique_region() [1/3]

```
template<typename T>
L4Re::Rm::Unique_region< T >::Unique_region (
    T addr) [inline], [explicit], [noexcept]
```

Construct a [Unique_region](#) from an address.

No region manager is set.

Parameters

<i>addr</i>	The new address
-------------	-----------------

Definition at line [456](#) of file [rm](#).

16.305.2.2 Unique_region() [2/3]

```
template<typename T>
L4Re::Rm::Unique_region< T >::Unique_region (
    T addr,
    L4::Cap< Rm > const & rm) [inline], [noexcept]
```

Construct a valid [Unique_region](#) from an address and a region manager.

Parameters

<i>addr</i>	The address
<i>rm</i>	The region manager

Definition at line 465 of file [rm](#).

16.305.2.3 Unique_region() [3/3]

```
template<typename T>
L4Re::Rm::Unique_region< T >::Unique_region (
    Unique_region< T > && o) [inline], [noexcept]
```

Move-Construct a [Unique_region](#).

Parameters

<i>o</i>	L-value reference to other region.
----------	------------------------------------

Definition at line 473 of file [rm](#).

16.305.2.4 ~Unique_region()

```
template<typename T>
L4Re::Rm::Unique_region< T >::~~Unique_region () [inline], [noexcept]
```

Destructor.

If the region is valid, call [detach](#).

Definition at line 498 of file [rm](#).

16.305.3 Member Function Documentation

16.305.3.1 get()

```
template<typename T>
T L4Re::Rm::Unique_region< T >::get () const [inline], [noexcept]
```

Return the address.

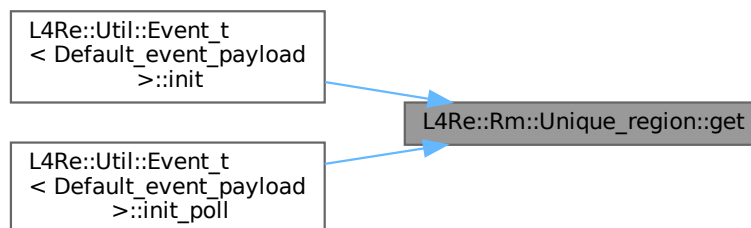
Returns

the address

Definition at line 509 of file [rm](#).

Referenced by [L4Re::Util::Event_t< Default_event_payload >::init\(\)](#), and [L4Re::Util::Event_t< Default_event_payload >::init_poll\(\)](#).

Here is the caller graph for this function:

**16.305.3.2 is_valid()**

```

template<typename T>
bool L4Re::Rm::Unique_region< T >::is_valid () const [inline], [noexcept]

```

Check if the [Unique_region](#) is valid.

Returns

true iff the [Unique_region](#) is valid

Definition at line 549 of file [rm](#).

16.305.3.3 operator=()

```

template<typename T>
Unique_region & L4Re::Rm::Unique_region< T >::operator= (
    Unique_region< T > && o) [inline], [noexcept]

```

Move-assign a [Unique_region](#).

Parameters

<i>o</i>	L-value reference to region to assign from
----------	--

Definition at line 481 of file [rm](#).

16.305.3.4 release()

```
template<typename T>
T L4Re::Rm::Unique_region< T >::release () [inline], [noexcept]
```

Return the address and invalidate the [Unique_region](#).

Returns

the address

Definition at line [517](#) of file [rm](#).

16.305.3.5 reset()

```
template<typename T>
void L4Re::Rm::Unique_region< T >::reset (
    T addr,
    L4::Cap< Rm > const & rm) [inline], [noexcept]
```

Set new address and region manager.

Parameters

<i>addr</i>	The new address
<i>rm</i>	The new region manager

Definition at line [529](#) of file [rm](#).

The documentation for this class was generated from the following file:

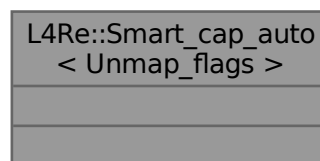
- [l4/re/rm](#)

16.306 L4Re::Smart_cap_auto< Unmap_flags > Class Template Reference

Helper for [Unique_cap](#) and [Unique_del_cap](#).

```
#include <cap_alloc>
```

Collaboration diagram for L4Re::Smart_cap_auto< Unmap_flags >:



16.306.1 Detailed Description

```
template<unsigned long Unmap_flags = L4_FP_ALL_SPACES>
class L4Re::Smart_cap_auto< Unmap_flags >
```

Helper for [Unique_cap](#) and [Unique_del_cap](#).

Definition at line 110 of file [cap_alloc](#).

The documentation for this class was generated from the following file:

- [l4/re/cap_alloc](#)

16.307 L4Re::Smart_count_cap< Unmap_flags > Class Template Reference

Helper for [Ref_cap](#) and [Ref_del_cap](#).

```
#include <cap_alloc>
```

Collaboration diagram for L4Re::Smart_count_cap< Unmap_flags >:

L4Re::Smart_count_cap < Unmap_flags >	
+	free()
+	copy()
+	invalidate()

Public Member Functions

- void **free** ([L4::Cap_base](#) &c) noexcept
Free operation for [L4::Smart_cap](#) (decrement ref count and delete if 0).
- [L4::Cap_base](#) **copy** ([L4::Cap_base](#) const &src)
Copy operation for [L4::Smart_cap](#) (increment ref count).

Static Public Member Functions

- static void **invalidate** ([L4::Cap_base](#) &c) noexcept
Invalidate operation for [L4::Smart_cap](#).

16.307.1 Detailed Description

```
template<unsigned long Unmap_flags = L4_FP_ALL_SPACES>
class L4Re::Smart_count_cap< Unmap_flags >
```

Helper for Ref_cap and Ref_del_cap.

Definition at line 139 of file [cap_alloc](#).

The documentation for this class was generated from the following file:

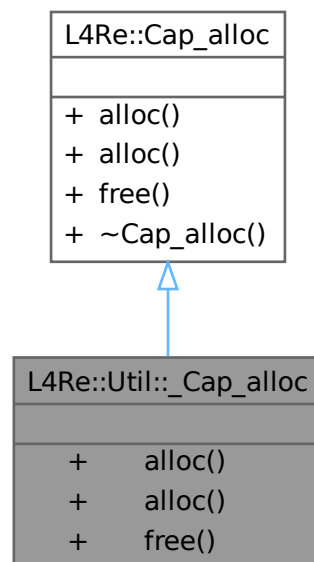
- [l4/re/cap_alloc](#)

16.308 L4Re::Util::_Cap_alloc Class Reference

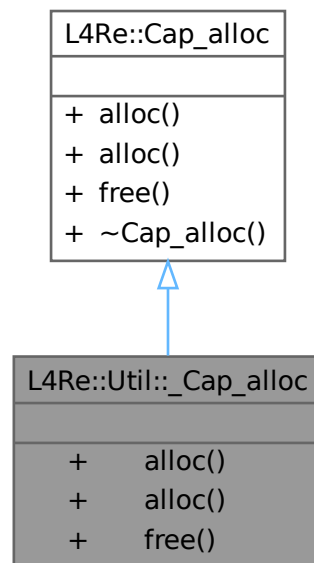
Adapter to expose the cap allocator implementation as [L4Re::Cap_alloc](#) compatible class.

```
#include <cap_alloc_impl.h>
```

Inheritance diagram for L4Re::Util::_Cap_alloc:



Collaboration diagram for L4Re::Util::_Cap_alloc:



Public Member Functions

- [L4::Cap](#)< void > [alloc](#) () noexcept override
Allocate a capability.
- template<typename T>
[L4::Cap](#)< T > [alloc](#) () noexcept
Allocate a capability.
- void [free](#) (L4::Cap< void > cap, [l4_cap_idx_t](#) task=[L4_INVALID_CAP](#), unsigned unmap_flags=[L4_FP_ALL_SPACES](#)) noexcept override
Free a capability.

Public Member Functions inherited from [L4Re::Cap_alloc](#)

- template<typename T>
[L4::Cap](#)< T > [alloc](#) () noexcept
Allocate a capability.
- virtual [~Cap_alloc](#) ()=0
Destructor.

16.308.1 Detailed Description

Adapter to expose the cap allocator implementation as [L4Re::Cap_alloc](#) compatible class.

Not intended to be used in application code.

Definition at line 66 of file [cap_alloc_impl.h](#).

16.308.2 Member Function Documentation

16.308.2.1 `alloc()` [1/2]

```
template<typename T>
L4::Cap< T > L4Re::Util::_Cap_alloc::alloc () [inline], [virtual], [noexcept]
```

Allocate a capability.

Returns

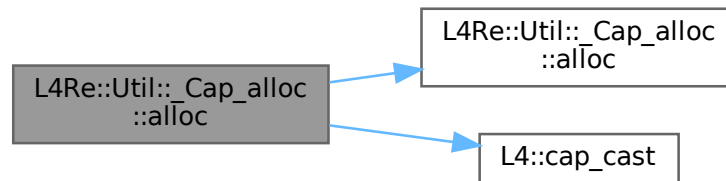
Capability of type void

Implements [L4Re::Cap_alloc](#).

Definition at line 79 of file [cap_alloc_impl.h](#).

References [alloc\(\)](#), and [L4::cap_cast\(\)](#).

Here is the call graph for this function:



16.308.2.2 `alloc()` [2/2]

```
L4::Cap< void > L4Re::Util::_Cap_alloc::alloc () [inline], [override], [virtual], [noexcept]
```

Allocate a capability.

Returns

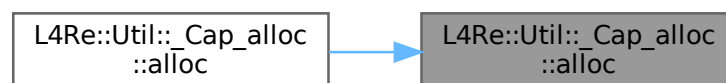
Capability of type void

Implements [L4Re::Cap_alloc](#).

Definition at line 75 of file [cap_alloc_impl.h](#).

Referenced by [alloc\(\)](#).

Here is the caller graph for this function:



16.308.2.3 free()

```
void L4Re::Util::_Cap_alloc::free (
    L4::Cap< void > cap,
    l4_cap_idx_t task = L4_INVALID_CAP,
    unsigned unmap_flags = L4_FP_ALL_SPACES) [inline], [override], [virtual], [noexcept]
```

Free a capability.

Parameters

<i>cap</i>	Capability to free.
<i>task</i>	If set, task to unmap the capability from.
<i>unmap_flags</i>	Flags for unmap, see l4_unmap_flags_t .

Implements [L4Re::Cap_alloc](#).

Definition at line 85 of file [cap_alloc_impl.h](#).

References [L4_FP_ALL_SPACES](#), and [L4_INVALID_CAP](#).

The documentation for this class was generated from the following file:

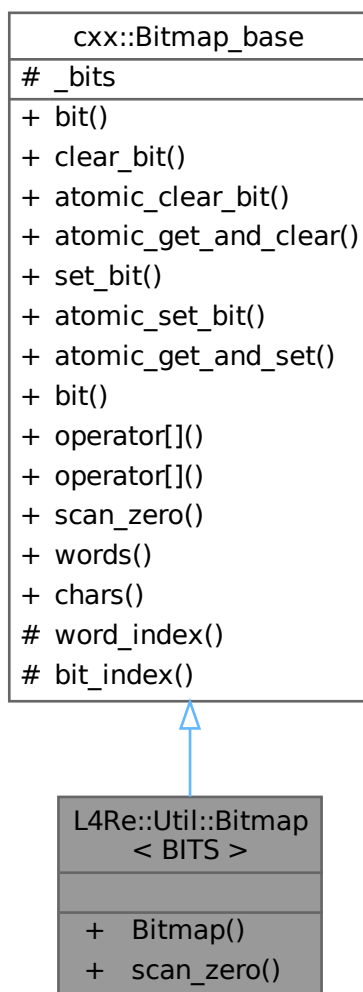
- [l4/re/util/cap_alloc_impl.h](#)

16.309 L4Re::Util::Bitmap< BITS > Class Template Reference

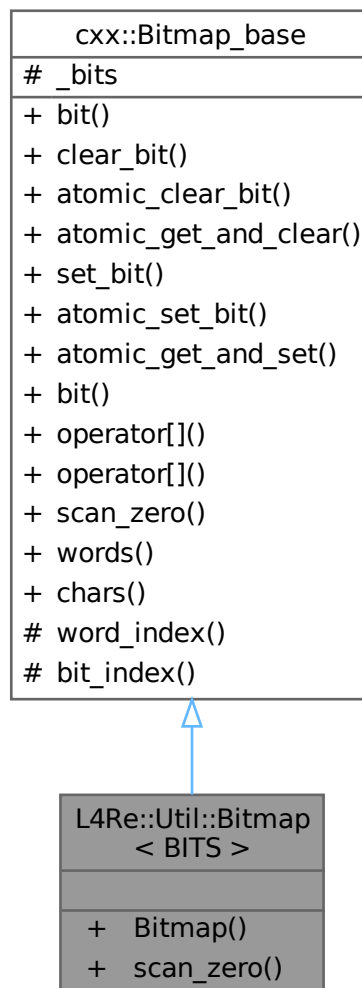
A static bitmap.

```
#include <bitmap>
```

Inheritance diagram for L4Re::Util::Bitmap< BITS >:



Collaboration diagram for L4Re::Util::Bitmap< BITS >:



Public Member Functions

- **Bitmap** () noexcept
Create a bitmap with `BITS` bits.
- long `scan_zero` (long start_bit=0) const noexcept
Scan for the first zero bit.

Public Member Functions inherited from `cxx::Bitmap_base`

- void `bit` (long bit, bool on) noexcept
Set the value of bit `bit` to on.
- void `clear_bit` (long bit) noexcept
Clear bit `bit`.

- void `atomic_clear_bit` (long `bit`) noexcept
Clear bit `bit` atomically.
- `word_type` `atomic_get_and_clear` (long `bit`) noexcept
Clear bit `bit` atomically and return old state.
- void `set_bit` (long `bit`) noexcept
Set bit `bit`.
- void `atomic_set_bit` (long `bit`) noexcept
Set bit `bit` atomically.
- `word_type` `atomic_get_and_set` (long `bit`) noexcept
Set bit `bit` atomically and return old state.
- `word_type` `bit` (long `bit`) const noexcept
Get the truth value of a bit.
- `word_type` `operator[]` (long `bit`) const noexcept
Get the bit at index `bit`.
- `Bit operator[]` (long `bit`) noexcept
Get the lvalue for the bit at index `bit`.
- long `scan_zero` (long `max_bit`, long `start_bit`=0) const noexcept
Scan for the first zero bit.

Additional Inherited Members

Static Public Member Functions inherited from `cxx::Bitmap_base`

- static long `words` (long bits) noexcept
Get the number of `Words` that are used for the bitmap.
- static long `chars` (long bits) noexcept
Get the number of chars that are used for the bitmap.

Protected Types inherited from `cxx::Bitmap_base`

- enum { `W_bits` = sizeof(word_type) * 8 , `C_bits` = 8 }
- typedef unsigned long `word_type`
Data type for each element of the bit buffer.

Static Protected Member Functions inherited from `cxx::Bitmap_base`

- static unsigned `word_index` (unsigned `bit`)
Get the word index for the given bit.
- static unsigned `bit_index` (unsigned `bit`)
Get the bit index within `word_type` for the given bit.

Protected Attributes inherited from `cxx::Bitmap_base`

- `word_type` * `_bits`
Pointer to the buffer storing the bits.

16.309.1 Detailed Description

```
template<int BITS>
class L4Re::Util::Bitmap< BITS >
```

A static bitmap.

Template Parameters

<i>BITS</i>	The number of bits that shall be in the bitmap.
-------------	---

Definition at line 220 of file [bitmap](#).

16.309.2 Member Function Documentation

16.309.2.1 scan_zero()

```
template<int BITS>
long cxx::Bitmap< BITS >::scan_zero (
    long start_bit = 0) const [inline], [noexcept]
```

Scan for the first zero bit.

Parameters

<i>start_bit</i>	Hint at the number of the first bit to look at. Zero bits below <i>start_bit</i> may or may not be taken into account by the implementation.
------------------	--

Return values

<i>>=</i>	0 Number of first zero bit found.
<i>-1</i>	All bits at <i>start_bit</i> or higher are set.

Compared to [Bitmap_base::scan_zero\(\)](#), the upper bound is set to BITS.

Definition at line 365 of file [bitmap](#).

The documentation for this class was generated from the following file:

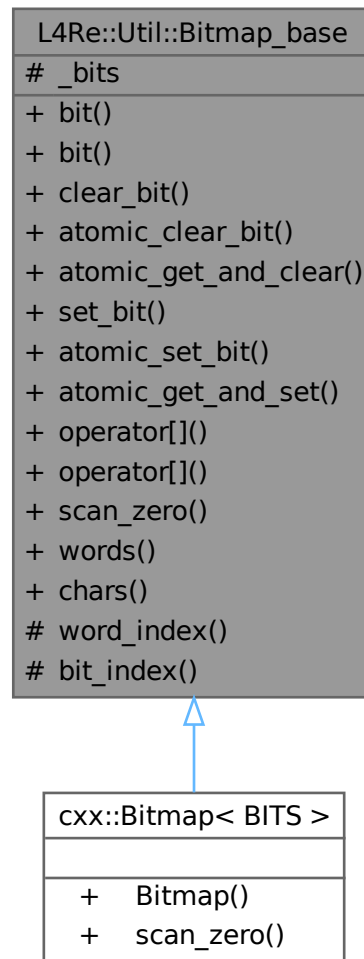
- [l4/cxx/bitmap](#)

16.310 L4Re::Util::Bitmap_base Class Reference

Basic bitmap abstraction.

```
#include <bitmap>
```

Inheritance diagram for L4Re::Util::Bitmap_base:



Collaboration diagram for L4Re::Util::Bitmap_base:

L4Re::Util::Bitmap_base
_bits
+ bit()
+ bit()
+ clear_bit()
+ atomic_clear_bit()
+ atomic_get_and_clear()
+ set_bit()
+ atomic_set_bit()
+ atomic_get_and_set()
+ operator[]()
+ operator[]()
+ scan_zero()
+ words()
+ chars()
word_index()
bit_index()

Data Structures

- class [Bit](#)
A writable bit in a bitmap.
- class [Word](#)
Helper abstraction for a word contained in the bitmap.
- class [Char](#)
Helper abstraction for a byte contained in the bitmap.

Public Member Functions

- void [bit](#) (long bit, bool on) noexcept
Set the value of bit [bit](#) to on.
- [word_type](#) [bit](#) (long bit) const noexcept
Get the truth value of a bit.
- void [clear_bit](#) (long [bit](#)) noexcept
Clear bit [bit](#).
- void [atomic_clear_bit](#) (long [bit](#)) noexcept
Clear bit [bit](#) atomically.
- [word_type](#) [atomic_get_and_clear](#) (long [bit](#)) noexcept
Clear bit [bit](#) atomically and return old state.

- void `set_bit` (long `bit`) noexcept
Set bit `bit`.
- void `atomic_set_bit` (long `bit`) noexcept
Set bit `bit` atomically.
- `word_type atomic_get_and_set` (long `bit`) noexcept
Set bit `bit` atomically and return old state.
- `word_type operator[]` (long `bit`) const noexcept
Get the bit at index `bit`.
- `Bit operator[]` (long `bit`) noexcept
Get the lvalue for the bit at index `bit`.
- long `scan_zero` (long `max_bit`, long `start_bit=0`) const noexcept
Scan for the first zero bit.

Static Public Member Functions

- static long `words` (long bits) noexcept
Get the number of `Words` that are used for the bitmap.
- static long `chars` (long bits) noexcept
Get the number of chars that are used for the bitmap.

Protected Types

- enum { `W_bits` = sizeof(word_type) * 8 , `C_bits` = 8 }
- typedef unsigned long `word_type`
Data type for each element of the bit buffer.

Static Protected Member Functions

- static unsigned `word_index` (unsigned `bit`)
Get the word index for the given bit.
- static unsigned `bit_index` (unsigned `bit`)
Get the bit index within `word_type` for the given bit.

Protected Attributes

- `word_type * _bits`
Pointer to the buffer storing the bits.

16.310.1 Detailed Description

Basic bitmap abstraction.

This abstraction keeps a pointer to a memory area that is used as bitmap.

Definition at line 18 of file `bitmap`.

16.310.2 Member Enumeration Documentation

16.310.2.1 anonymous enum

```
anonymous enum [protected]
```

Enumerator

W_bits	number of bits in word_type
C_bits	number of bits in char

Definition at line 26 of file [bitmap](#).

16.310.3 Member Function Documentation

16.310.3.1 `atomic_clear_bit()`

```
void cxx::Bitmap_base::atomic_clear_bit (  
    long bit) [inline], [noexcept]
```

Clear bit [bit](#) atomically.

Use this function for multi-threaded access to the bitmap.

Parameters

<i>bit</i>	The number of the bit to clear.
------------	---------------------------------

Definition at line 269 of file [bitmap](#).

16.310.3.2 `atomic_get_and_clear()`

```
Bitmap_base::word_type cxx::Bitmap_base::atomic_get_and_clear (  
    long bit) [inline], [noexcept]
```

Clear bit [bit](#) atomically and return old state.

Use this function for multi-threaded access to the bitmap.

Parameters

<i>bit</i>	The number of the bit to clear.
------------	---------------------------------

Definition at line 279 of file [bitmap](#).

16.310.3.3 `atomic_get_and_set()`

```
Bitmap_base::word_type cxx::Bitmap_base::atomic_get_and_set (  
    long bit) [inline], [noexcept]
```

Set bit [bit](#) atomically and return old state.

Use this function for multi-threaded access to the bitmap.

Parameters

<i>bit</i>	The number of the bit to set.
------------	-------------------------------

Definition at line 308 of file [bitmap](#).

16.310.3.4 atomic_set_bit()

```
void cxx::Bitmap_base::atomic_set_bit (
    long bit) [inline], [noexcept]
```

Set bit [bit](#) atomically.

Use this function for multi-threaded access to the bitmap.

Parameters

<i>bit</i>	The number of the bit to set.
------------	-------------------------------

Definition at line 298 of file [bitmap](#).

16.310.3.5 bit() [1/2]

```
Bitmap_base::word_type cxx::Bitmap_base::bit (
    long bit) const [inline], [noexcept]
```

Get the truth value of a bit.

Parameters

<i>bit</i>	The number of the bit to read.
------------	--------------------------------

Return values

0	Bit is not set.
!= 0	0 Bit is set.

Definition at line 318 of file [bitmap](#).

16.310.3.6 bit() [2/2]

```
void cxx::Bitmap_base::bit (
    long bit,
    bool on) [inline], [noexcept]
```

Set the value of bit [bit](#) to on.

Parameters

<i>bit</i>	The number of the bit.
<i>on</i>	The boolean value that shall be assigned to the bit.

Definition at line 251 of file [bitmap](#).

16.310.3.7 bit_index()

```
unsigned cxx::Bitmap_base::bit_index (  
    unsigned bit) [inline], [static], [protected]
```

Get the bit index within [word_type](#) for the given bit.

Parameters

<i>bit</i>	The bit index in the bitmap.
------------	------------------------------

Returns

the bit index within [word_type](#) (bit % W_bits).

Definition at line 53 of file [bitmap](#).

16.310.3.8 clear_bit()

```
void cxx::Bitmap_base::clear_bit (  
    long bit) [inline], [noexcept]
```

Clear bit [bit](#).

Parameters

<i>bit</i>	The number of the bit to clear.
------------	---------------------------------

Definition at line 260 of file [bitmap](#).

16.310.3.9 operator[]() [1/2]

```
word\_type cxx::Bitmap_base::operator[] (  
    long bit) const [inline], [noexcept]
```

Get the bit at index [bit](#).

Parameters

<i>bit</i>	The number of the bit to read.
------------	--------------------------------

Return values

0	Bit is not set.
!= 0	Bit is set.

Definition at line 181 of file [bitmap](#).

16.310.3.10 operator[]() [2/2]

```
Bit cxx::Bitmap_base::operator[] (
    long bit) [inline], [noexcept]
```

Get the lvalue for the bit at index [bit](#).

Parameters

<i>bit</i>	The number.
------------	-------------

Returns

lvalue for [bit](#)

Definition at line 191 of file [bitmap](#).

16.310.3.11 scan_zero()

```
long cxx::Bitmap_base::scan_zero (
    long max_bit,
    long start_bit = 0) const [inline], [noexcept]
```

Scan for the first zero bit.

Parameters

<i>max_bit</i>	Upper bound (exclusive) for the scanning operation.
<i>start_bit</i>	Hint at the number of the first bit to look at. Zero bits below <i>start_bit</i> may or may not be taken into account by the implementation.

Return values

>=	0 Number of first zero bit found.
----	-----------------------------------

-1	All bits between <code>start_bit</code> and <code>max_bit</code> are set.
----	---

Definition at line 339 of file [bitmap](#).

16.310.3.12 `set_bit()`

```
void cxx::Bitmap_base::set_bit (  
    long bit) [inline], [noexcept]
```

Set bit [bit](#).

Parameters

<i>bit</i>	The number of the bit to set.
------------	-------------------------------

Definition at line 289 of file [bitmap](#).

16.310.3.13 `word_index()`

```
unsigned cxx::Bitmap_base::word_index (  
    unsigned bit) [inline], [static], [protected]
```

Get the word index for the given bit.

Parameters

<i>bit</i>	The index of the bit in question.
------------	-----------------------------------

Returns

the index in [Bitmap_base::_bits](#) for the given bit (bit / W_bits).

Definition at line 44 of file [bitmap](#).

The documentation for this class was generated from the following file:

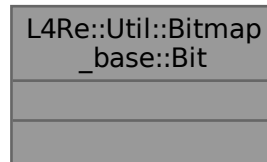
- I4/cxx/bitmap

16.311 L4Re::Util::Bitmap_base::Bit Class Reference

A writable bit in a bitmap.

```
#include <bitmap>
```

Collaboration diagram for L4Re::Util::Bitmap_base::Bit:



16.311.1 Detailed Description

A writable bit in a bitmap.

Definition at line 58 of file [bitmap](#).

The documentation for this class was generated from the following file:

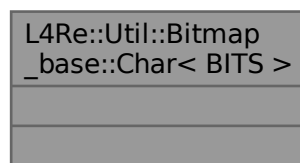
- [l4/cxx/bitmap](#)

16.312 L4Re::Util::Bitmap_base::Char< BITS > Class Template Reference

Helper abstraction for a byte contained in the bitmap.

```
#include <bitmap>
```

Collaboration diagram for L4Re::Util::Bitmap_base::Char< BITS >:



16.312.1 Detailed Description

```
template<long BITS>
class L4Re::Util::Bitmap_base::Char< BITS >
```

Helper abstraction for a byte contained in the bitmap.

Definition at line 95 of file [bitmap](#).

The documentation for this class was generated from the following file:

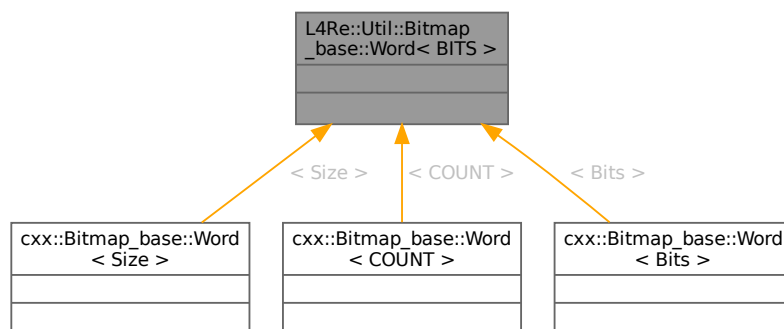
- I4/cxx/bitmap

16.313 L4Re::Util::Bitmap_base::Word< BITS > Class Template Reference

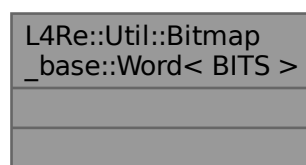
Helper abstraction for a word contained in the bitmap.

```
#include <bitmap>
```

Inheritance diagram for L4Re::Util::Bitmap_base::Word< BITS >:



Collaboration diagram for L4Re::Util::Bitmap_base::Word< BITS >:



16.313.1 Detailed Description

```
template<long BITS>  
class L4Re::Util::Bitmap_base::Word< BITS >
```

Helper abstraction for a word contained in the bitmap.

Definition at line 79 of file [bitmap](#).

The documentation for this class was generated from the following file:

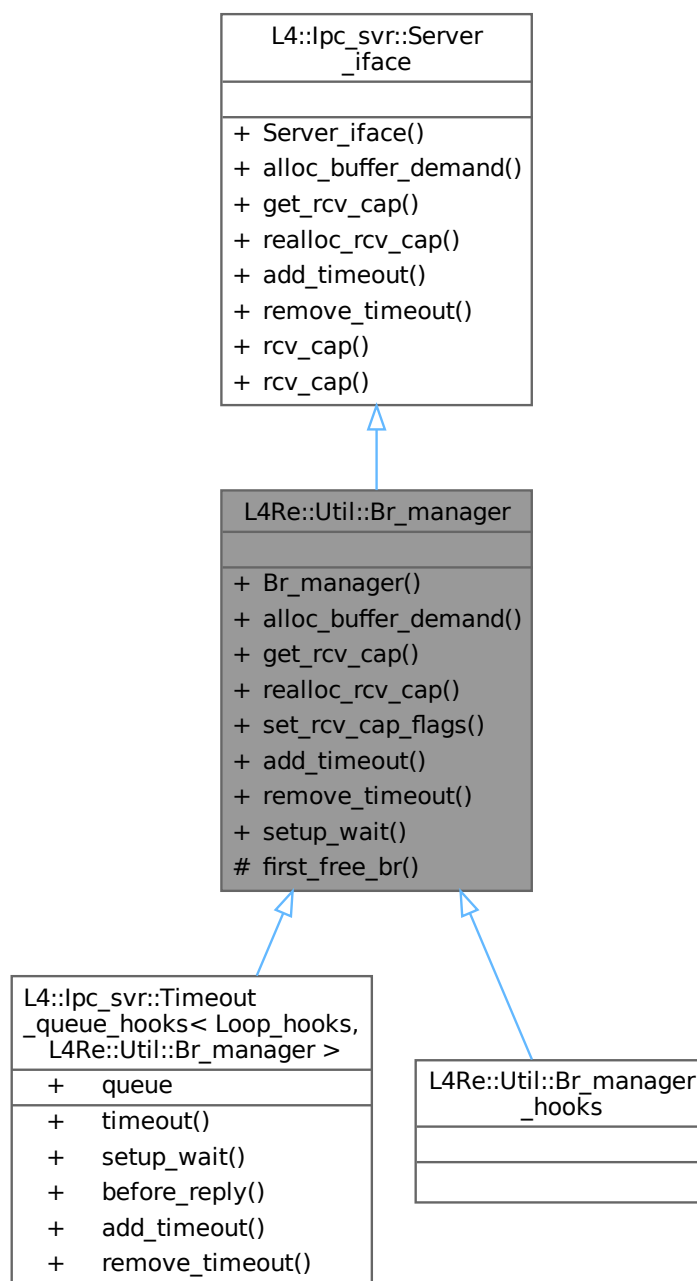
- I4/cxx/bitmap

16.314 L4Re::Util::Br_manager Class Reference

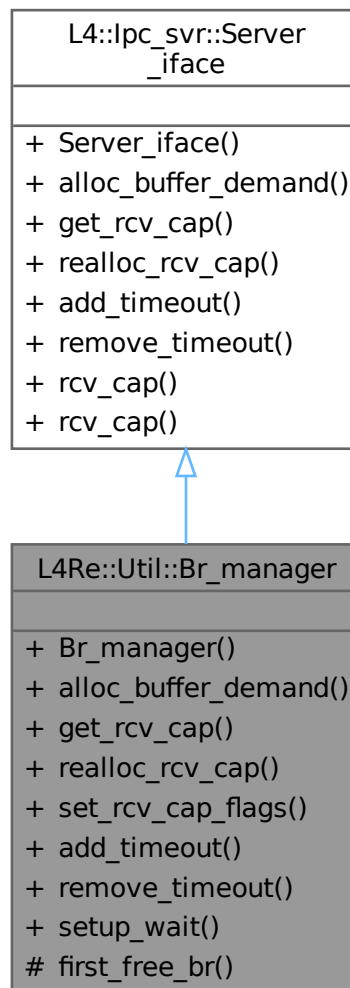
Buffer-register (BR) manager for [L4::Server](#).

```
#include <br_manager>
```


Inheritance diagram for L4Re::Util::Br_manager:



Collaboration diagram for L4Re::Util::Br_manager:



Public Member Functions

- **Br_manager ()**
Make a buffer-register (BR) manager.
- int **alloc_buffer_demand** (**Demand** const &d) override
Tells the server to allocate buffers for the given demand.
- **L4::Cap**< void > **get_rcv_cap** (int i) const override
Get capability slot allocated to the given receive buffer.
- int **realloc_rcv_cap** (int i) override
Allocate a new capability for the given receive buffer.
- void **set_rcv_cap_flags** (unsigned long flags)
Set the receive flags for the buffers.
- int **add_timeout** (**L4::lpc_svr::Timeout** *, **l4_kernel_clock_t**) override
No timeouts handled by us.

- int **remove_timeout** ([L4::lpc_svr::Timeout](#) *) override
No timeouts handled by us.
- void **setup_wait** ([l4_utcb_t](#) *utcb, [L4::lpc_svr::Reply_mode](#))
setup_wait() used the server loop ([L4::Server](#))

Public Member Functions inherited from [L4::lpc_svr::Server_iface](#)

- **Server_iface** ()
Make a server interface.
- template<typename T>
[L4::Cap](#)< T > **rcv_cap** (int index) const
Get given receive buffer as typed capability.
- [L4::Cap](#)< void > **rcv_cap** (int index) const
Get receive cap with the given index as generic (void) type.

Protected Member Functions

- unsigned **first_free_br** () const
Used for assigning BRs for a timeout.

Additional Inherited Members

Public Types inherited from [L4::lpc_svr::Server_iface](#)

- typedef [L4::Type_info::Demand](#) **Demand**
Data type expressing server-side demand for receive buffers.

16.314.1 Detailed Description

Buffer-register (BR) manager for [L4::Server](#).

Implementation of the [L4::lpc_svr::Server_iface](#) API for managing the server-side receive buffers needed for a set of server objects running within a server.

Definition at line 25 of file [br_manager](#).

16.314.2 Member Function Documentation

16.314.2.1 **alloc_buffer_demand()**

```
int L4Re::Util::Br_manager::alloc_buffer_demand (
    Demand const & demand) [inline], [override], [virtual]
```

Tells the server to allocate buffers for the given demand.

Parameters

<i>demand</i>	The total server-side demand of receive buffers needed for a given interface, see Demand .
---------------	--

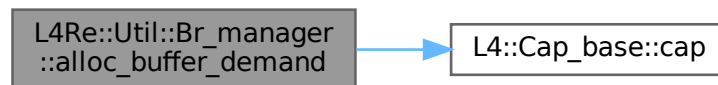
This function is not called by user applications directly. Usually the server implementation or the registry implementation calls this function whenever a new object is registered at the server.

Implements [L4::lpc_svr::Server_iface](#).

Definition at line 54 of file [br_manager](#).

References [L4::Cap_base::cap\(\)](#), [L4Re::Util::cap_alloc](#), [L4::Type_info::Demand::caps](#), [L4_EINVAL](#), [L4_ENOMEM](#), [L4_EOK](#), [L4_ERANGE](#), [L4::Type_info::Demand::mem](#), and [L4::Type_info::Demand::ports](#).

Here is the call graph for this function:



16.314.2.2 get_rcv_cap()

```
L4::Cap< void > L4Re::Util::Br_manager::get_rcv_cap (
    int index) const [inline], [override], [virtual]
```

Get capability slot allocated to the given receive buffer.

Parameters

<i>index</i>	The receive buffer index of the expected capability argument ($0 \leq \text{index} < \text{caps}$ registered with alloc_buffer_demand()).
--------------	---

Precondition

$0 \leq \text{index} < \text{caps}$ registered with [alloc_buffer_demand\(\)](#)

Returns

Capability slot currently allocated to the given receive buffer.

Implements [L4::lpc_svr::Server_iface](#).

Definition at line 86 of file [br_manager](#).

References [L4::Cap_base::Invalid](#), and [L4_CAP_MASK](#).

16.314.2.3 realloc_rcv_cap()

```
int L4Re::Util::Br_manager::realloc_rcv_cap (
    int index) [inline], [override], [virtual]
```

Allocate a new capability for the given receive buffer.

Parameters

<i>index</i>	The receive buffer index of the expected capability argument ($0 \leq \text{index} < \text{caps}$ registered with alloc_buffer_demand()).
--------------	---

Precondition

$0 \leq \text{index} < \text{caps}$ registered with [alloc_buffer_demand\(\)](#)

Returns

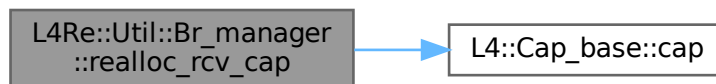
0 on success, < 0 on error.

Implements [L4::lpc_svr::Server_iface](#).

Definition at line 94 of file [br_manager](#).

References [L4::Cap_base::cap\(\)](#), [L4Re::Util::cap_alloc](#), [L4_EINVAL](#), [L4_ENOMEM](#), and [L4_EOK](#).

Here is the call graph for this function:

**16.314.2.4 set_rcv_cap_flags()**

```
void L4Re::Util::Br_manager::set_rcv_cap_flags (
    unsigned long flags) [inline]
```

Set the receive flags for the buffers.

Precondition

Must be called before any handlers are registered.

Parameters

<i>flags</i>	New receive capability flags, see l4_msg_item_consts_t .
--------------	--

Definition at line 118 of file [br_manager](#).

References [l4_assert](#).

The documentation for this class was generated from the following file:

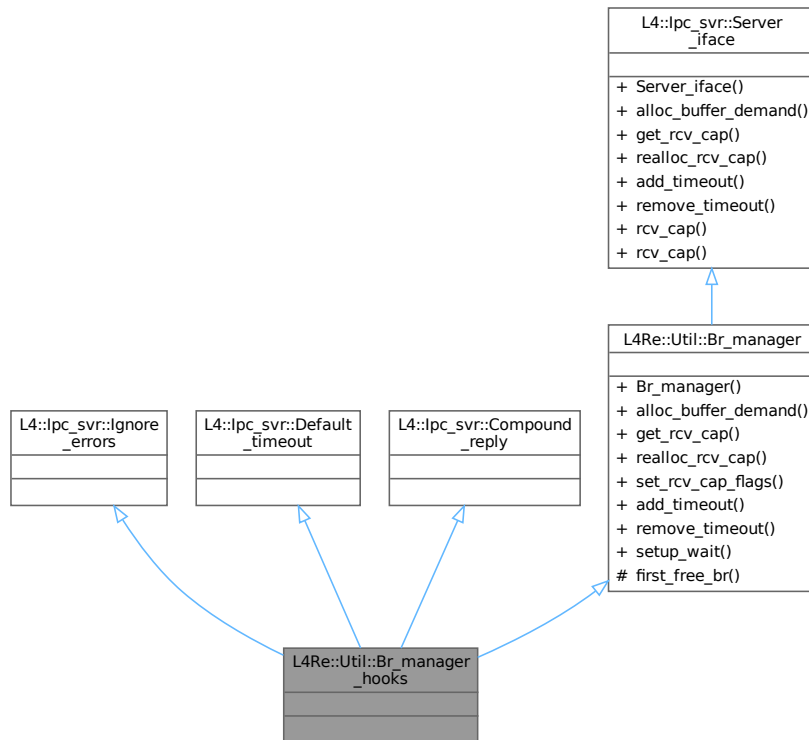
- `l4/re/util/br_manager`

16.315 L4Re::Util::Br_manager_hooks Struct Reference

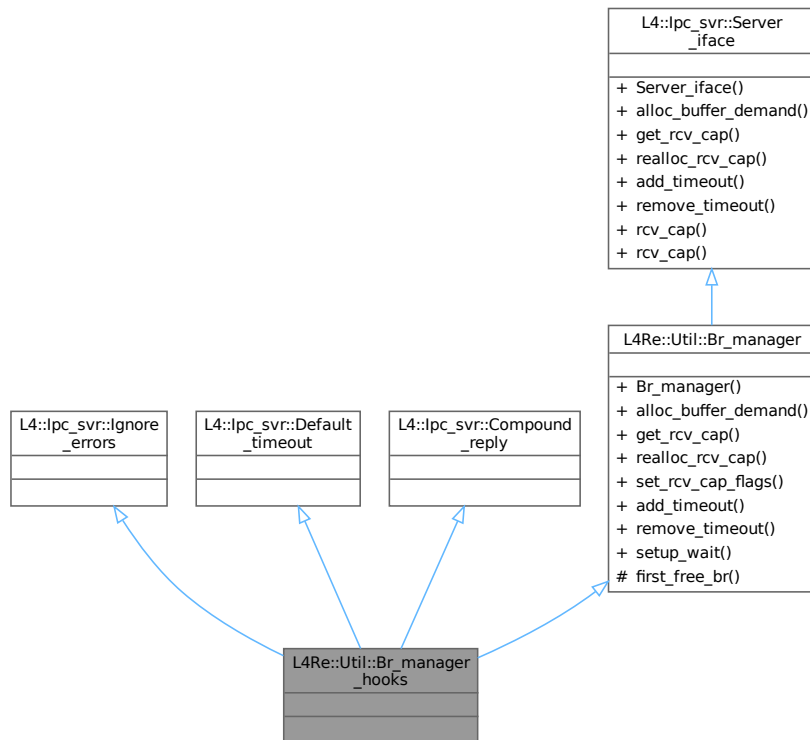
Predefined server-loop hooks for a server loop using the [Br_manager](#).

```
#include <br_manager>
```

Inheritance diagram for L4Re::Util::Br_manager_hooks:



Collaboration diagram for L4Re::Util::Br_manager_hooks:



Additional Inherited Members

Public Types inherited from L4::lpc_svr::Server_iface

- typedef L4::Type_info::Demand Demand
Data type expressing server-side demand for receive buffers.

Public Member Functions inherited from L4Re::Util::Br_manager

- **Br_manager** ()
Make a buffer-register (BR) manager.
- int **alloc_buffer_demand** (Demand const &d) override
Tells the server to allocate buffers for the given demand.
- L4::Cap< void > **get_rcv_cap** (int i) const override
Get capability slot allocated to the given receive buffer.
- int **realloc_rcv_cap** (int i) override
Allocate a new capability for the given receive buffer.
- void **set_rcv_cap_flags** (unsigned long flags)
Set the receive flags for the buffers.
- int **add_timeout** (L4::lpc_svr::Timeout *, l4_kernel_clock_t) override
No timeouts handled by us.
- int **remove_timeout** (L4::lpc_svr::Timeout *) override
No timeouts handled by us.
- void **setup_wait** (l4_utcb_t *utcb, L4::lpc_svr::Reply_mode)
setup_wait() used the server loop (L4::Server)

Public Member Functions inherited from [L4::lpc_svr::Server_iface](#)

- **Server_iface** ()
Make a server interface.
- `template<typename T>`
[L4::Cap](#)< T > [rcv_cap](#) (int index) const
Get given receive buffer as typed capability.
- [L4::Cap](#)< void > [rcv_cap](#) (int index) const
Get receive cap with the given index as generic (void) type.

Protected Member Functions inherited from [L4Re::Util::Br_manager](#)

- `unsigned first_free_br` () const
Used for assigning BRs for a timeout.

16.315.1 Detailed Description

Predefined server-loop hooks for a server loop using the [Br_manager](#).

This class can be used whenever a server loop including full management of receive buffer resources is needed.

Definition at line 165 of file [br_manager](#).

The documentation for this struct was generated from the following file:

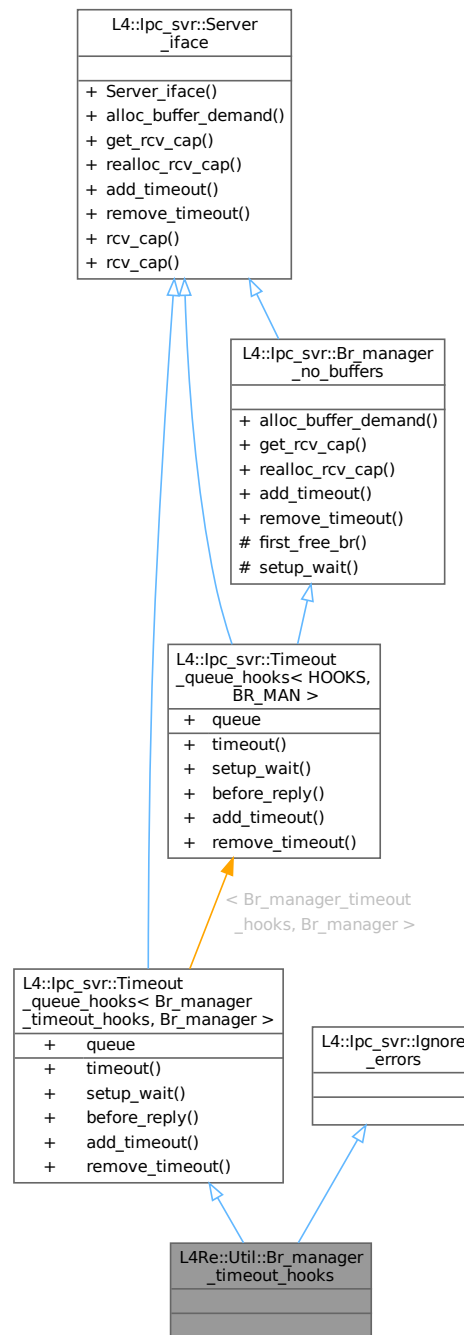
- `l4/re/util/br_manager`

16.316 L4Re::Util::Br_manager_timeout_hooks Struct Reference

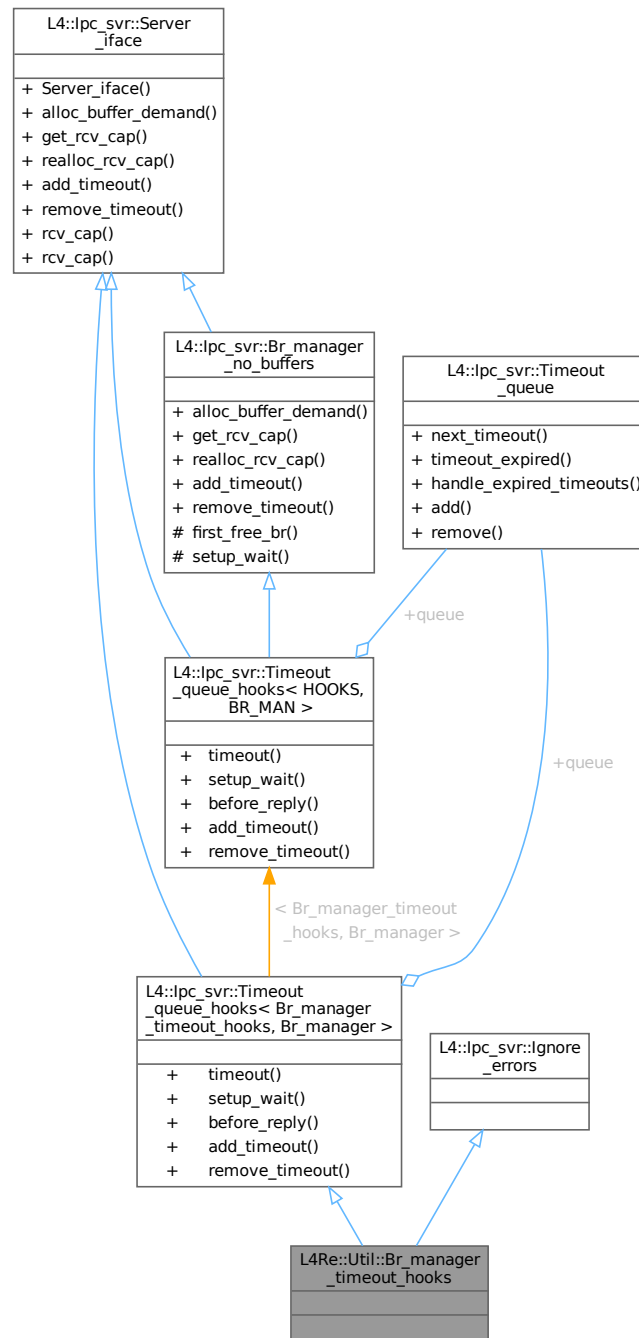
Predefined server-loop hooks for a server with using the [Br_manager](#) and a timeout queue.

```
#include <br_manager>
```


Inheritance diagram for L4Re::Util::Br_manager_timeout_hooks:



Collaboration diagram for L4Re::Util::Br_manager_timeout_hooks:



Additional Inherited Members

Public Types inherited from L4::lpc_svr::Server_iface

- typedef L4::Type_info::Demand Demand
Data type expressing server-side demand for receive buffers.

Public Member Functions inherited from**L4::lpc_svr::Timeout_queue_hooks< Br_manager_timeout_hooks, Br_manager >**

- **l4_timeout_t** **timeout** ()
get the time for the next timeout
- void **setup_wait** (l4_utcb_t *utcb, L4::lpc_svr::Reply_mode mode)
setup_wait() for the server loop
- L4::lpc_svr::Reply_mode **before_reply** (l4_msgtag_t, l4_utcb_t *)
server loop hook
- int **add_timeout** (Timeout *timeout, l4_kernel_clock_t time) override
Add a timeout to the queue for time time.
- int **remove_timeout** (Timeout *timeout) override
Remove timeout from the queue.

Public Member Functions inherited from L4::lpc_svr::Server_iface

- **Server_iface** ()
Make a server interface.
- virtual int **alloc_buffer_demand** (Demand const &demand)=0
Tells the server to allocate buffers for the given demand.
- virtual L4::Cap< void > **get_rcv_cap** (int index) const =0
Get capability slot allocated to the given receive buffer.
- virtual int **realloc_rcv_cap** (int index)=0
Allocate a new capability for the given receive buffer.
- template<typename T>
L4::Cap< T > **rcv_cap** (int index) const
Get given receive buffer as typed capability.
- L4::Cap< void > **rcv_cap** (int index) const
Get receive cap with the given index as generic (void) type.

Data Fields inherited from**L4::lpc_svr::Timeout_queue_hooks< Br_manager_timeout_hooks, Br_manager >**

- **Timeout_queue** **queue**
Use this timeout queue.

16.316.1 Detailed Description

Predefined server-loop hooks for a server with using the [Br_manager](#) and a timeout queue.

This class can be used for server loops that need the full package of buffer-register management and a timeout queue.

Definition at line 179 of file [br_manager](#).

The documentation for this struct was generated from the following file:

- l4/re/util/br_manager

16.317 L4Re::Util::Cap_alloc_base Class Reference

Capability allocator.

```
#include <bitmap_cap_alloc>
```

Collaboration diagram for L4Re::Util::Cap_alloc_base:

L4Re::Util::Cap_alloc_base	
+	alloc()
+	free()

Public Member Functions

- template<typename T>
[L4::Cap](#)< T > **alloc** () noexcept
Allocate a capability slot.
- template<typename T>
void **free** ([L4::Cap](#)< T > const &cap, [l4_cap_idx_t](#) task=[L4_INVALID_CAP](#), [l4_umword_t](#) unmap_↵
flags=[L4_FP_ALL_SPACES](#)) noexcept
Free a capability slot.

16.317.1 Detailed Description

Capability allocator.

Definition at line 28 of file [bitmap_cap_alloc](#).

The documentation for this class was generated from the following file:

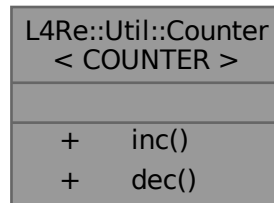
- [l4/re/util/bitmap_cap_alloc](#)

16.318 L4Re::Util::Counter< COUNTER > Struct Template Reference

[Counter](#) for [Counting_cap_alloc](#) with variable data width.

```
#include <counting_cap_alloc>
```

Collaboration diagram for L4Re::Util::Counter< COUNTER >:



Public Member Functions

- bool [inc](#) ()
Increment counter if not yet saturated.
- Type [dec](#) ()
Decrement counter if not saturated.

16.318.1 Detailed Description

```
template<typename COUNTER = unsigned char>
struct L4Re::Util::Counter< COUNTER >
```

[Counter](#) for [Counting_cap_alloc](#) with variable data width.

This version is not thread safe.

Definition at line 27 of file [counting_cap_alloc](#).

16.318.2 Member Function Documentation

16.318.2.1 dec()

```
template<typename COUNTER = unsigned char>
Type L4Re::Util::Counter< COUNTER >::dec () [inline]
```

Decrement counter if not saturated.

Once the counter reached the saturated state, the counter value isn't changed.

Definition at line 67 of file [counting_cap_alloc](#).

16.318.2.2 inc()

```
template<typename COUNTER = unsigned char>  
bool L4Re::Util::Counter< COUNTER >::inc () [inline]
```

Increment counter if not yet saturated.

Once the counter reached the saturated state, the counter value isn't changed.

Return values

<i>false</i>	The counter just went saturated after it was increased.
<i>true</i>	Either the counter was already saturated – in that case the counter value was not changed, or the counter was not saturated – in that case the counter was increased.

Definition at line 50 of file [counting_cap_alloc](#).

The documentation for this struct was generated from the following file:

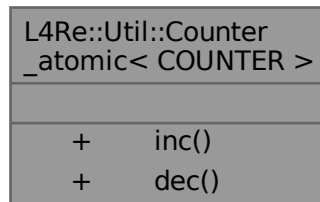
- [l4/re/util/counting_cap_alloc](#)

16.319 L4Re::Util::Counter_atomic< COUNTER > Struct Template Reference

Thread safe version of counter for [Counting_cap_alloc](#).

```
#include <counting_cap_alloc>
```

Collaboration diagram for L4Re::Util::Counter_atomic< COUNTER >:



Public Member Functions

- bool [inc](#) ()
Increment counter if not yet saturated.
- Type [dec](#) ()
Decrement counter if not saturated.

16.319.1 Detailed Description

```
template<typename COUNTER = unsigned char>
struct L4Re::Util::Counter_atomic< COUNTER >
```

Thread safe version of counter for [Counting_cap_alloc](#).

Despite using atomic instructions, this version has to make sure that capability slots are not reused too early. If the last reference is gone, the capability slot has to be unmapped. The slot must only be allocated again when the unmap has completed. This is accomplished by starting with an initial count of 2. The last reference will decrease the counter to 1. Only then, after the slot was unmapped, will the counter be set to 0. This will allow other threads to reallocate the slot.

Definition at line 98 of file [counting_cap_alloc](#).

16.319.2 Member Function Documentation

16.319.2.1 dec()

```
template<typename COUNTER = unsigned char>
Type L4Re::Util::Counter_atomic< COUNTER >::dec () [inline]
```

Decrement counter if not saturated.

Once the counter reached the saturated state, the counter value isn't changed.

Definition at line 142 of file [counting_cap_alloc](#).

16.319.2.2 inc()

```
template<typename COUNTER = unsigned char>
bool L4Re::Util::Counter_atomic< COUNTER >::inc () [inline]
```

Increment counter if not yet saturated.

Once the counter reached the saturated state, the counter value isn't changed.

Return values

<i>false</i>	The counter just went saturated after it was increased.
<i>true</i>	Either the counter was already saturated – in that case the counter value was not changed, or the counter was not saturated – in that case the counter was increased.

Definition at line 121 of file [counting_cap_alloc](#).

The documentation for this struct was generated from the following file:

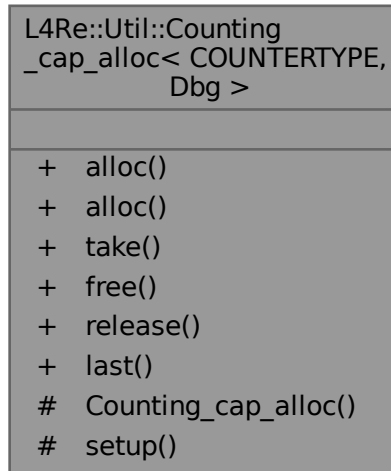
- [l4/re/util/counting_cap_alloc](#)

16.320 L4Re::Util::Counting_cap_alloc< COUNTERTYPE, Dbg > Class Template Reference

Internal reference-counting cap allocator.

```
#include <counting_cap_alloc>
```


Collaboration diagram for L4Re::Util::Counting_cap_alloc< COUNTERTYPE, Dbg >:



Public Member Functions

- `L4::Cap< void > alloc ()` noexcept
Allocate a new capability slot.
- `template<typename T>`
`L4::Cap< T > alloc ()` noexcept
Allocate a new capability slot.
- `void take (L4::Cap< void > cap)` noexcept
Increase the reference counter for the capability.
- `bool free (L4::Cap< void > cap, l4_cap_idx_t task=L4_INVALID_CAP, unsigned unmap_flags=L4_FP_ALL_SPACES)` noexcept
Free the capability.
- `bool release (L4::Cap< void > cap, l4_cap_idx_t task=L4_INVALID_CAP, unsigned unmap_flags=L4_FP_ALL_SPACES)` noexcept
Decrease the reference counter for a capability.
- `long last ()` noexcept
Return highest capability id managed by this allocator.

Protected Member Functions

- `Counting_cap_alloc ()` noexcept
Create a new, empty allocator.
- `void setup (void *m, long capacity, long bias, Dbg *dbg)` noexcept
Set up the backing memory for the allocator and the area of managed capability slots.

16.320.1 Detailed Description

```
template<typename COUNTERTYPE, typename Dbg>
class L4Re::Util::Counting_cap_alloc< COUNTERTYPE, Dbg >
```

Internal reference-counting cap allocator.

This is intended for internal use only. [L4Re](#) applications should use [L4Re::Util::cap_alloc\(\)](#).

Allocator for capability slots that automatically frees the slot and optionally unmaps the capability when the reference count goes down to zero. Reference counting must be done manually via [take\(\)](#) and [release\(\)](#). The backing store for the reference counters must be provided in the [setup\(\)](#) method. The allocator can recognize capability slots that are not managed by itself and does nothing on such slots.

Note

The user must ensure that the backing store is zero-initialized.

The user must ensure that the capability slots managed by this allocator are not used by a different allocator, see [setup\(\)](#).

The operations in this class are not thread-safe.

Definition at line 191 of file [counting_cap_alloc](#).

16.320.2 Constructor & Destructor Documentation

16.320.2.1 Counting_cap_alloc()

```
template<typename COUNTERTYPE, typename Dbg>
L4Re::Util::Counting\_cap\_alloc< COUNTERTYPE, Dbg >::Counting_cap_alloc () [inline], [protected],
[noexcept]
```

Create a new, empty allocator.

Needs to be initialized with [setup\(\)](#) before it can be used.

Definition at line 224 of file [counting_cap_alloc](#).

16.320.3 Member Function Documentation

16.320.3.1 alloc() [1/2]

```
template<typename COUNTERTYPE, typename Dbg>
template<typename T>
L4::Cap< T > L4Re::Util::Counting\_cap\_alloc< COUNTERTYPE, Dbg >::alloc () [inline], [noexcept]
```

Allocate a new capability slot.

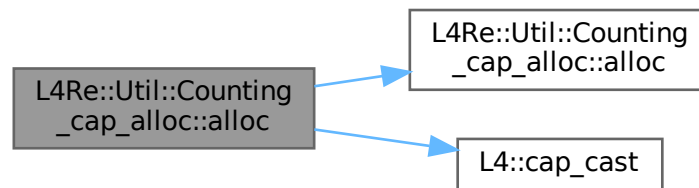
Returns

The newly allocated capability slot, invalid if the allocator was exhausted.

Definition at line 282 of file [counting_cap_alloc](#).

References [alloc\(\)](#), and [L4::cap_cast\(\)](#).

Here is the call graph for this function:

**16.320.3.2 alloc() [2/2]**

```
template<typename COUNTERTYPE, typename Dbg>
L4::Cap< void > L4Re::Util::Counting_cap_alloc< COUNTERTYPE, Dbg >::alloc () [inline], [noexcept]
```

Allocate a new capability slot.

Returns

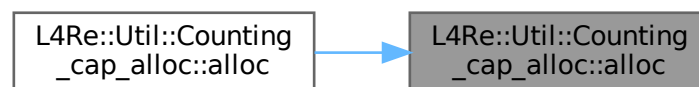
The newly allocated capability slot, invalid if the allocator was exhausted.

Definition at line 257 of file [counting_cap_alloc](#).

References [L4::Cap_base::Invalid](#), and [L4_CAP_SHIFT](#).

Referenced by [alloc\(\)](#).

Here is the caller graph for this function:



16.320.3.3 free()

```
template<typename COUNTERTYPE, typename Dbg>
bool L4Re::Util::Counting_cap_alloc< COUNTERTYPE, Dbg >::free (
    L4::Cap< void > cap,
    l4_cap_idx_t task = L4_INVALID_CAP,
    unsigned unmap_flags = L4_FP_ALL_SPACES) [inline], [noexcept]
```

Free the capability.

Parameters

<i>cap</i>	Capability to free.
<i>task</i>	If set, task to unmap the capability from.
<i>unmap_flags</i>	Flags for unmap, see l4_unmap_flags_t .

Precondition

The capability has been allocated. Calling free twice on a capability managed by this allocator results in undefined behaviour.

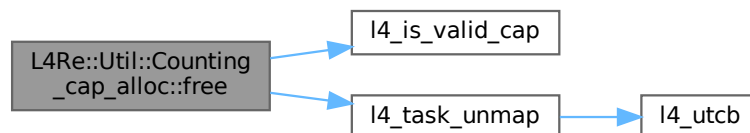
Returns

True, if the capability was managed by this allocator.

Definition at line 321 of file [counting_cap_alloc](#).

References [l4_assert](#), [L4_FP_ALL_SPACES](#), [L4_INVALID_CAP](#), [l4_is_valid_cap\(\)](#), and [l4_task_unmap\(\)](#).

Here is the call graph for this function:



16.320.3.4 release()

```
template<typename COUNTERTYPE, typename Dbg>
bool L4Re::Util::Counting_cap_alloc< COUNTERTYPE, Dbg >::release (
    L4::Cap< void > cap,
    l4_cap_idx_t task = L4_INVALID_CAP,
    unsigned unmap_flags = L4_FP_ALL_SPACES) [inline], [noexcept]
```

Decrease the reference counter for a capability.

Parameters

<i>cap</i>	Capability to release.
<i>task</i>	If set, task to unmap the capability from.
<i>unmap_flags</i>	Flags for unmap, see l4_unmap_flags_t .

Precondition

The capability has been allocated. Calling release on a free capability results in undefined behaviour.

Returns

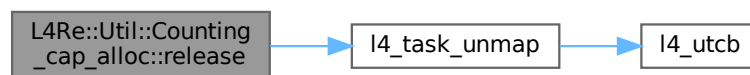
True, if the capability was freed as a result of this operation. If false is returned the capability is either still in use or is not managed by this allocator.

Does nothing apart from returning false if the capability is not managed by this allocator.

Definition at line 359 of file [counting_cap_alloc](#).

References [l4_assert](#), [L4_FP_ALL_SPACES](#), [L4_INVALID_CAP](#), and [l4_task_unmap\(\)](#).

Here is the call graph for this function:

**16.320.3.5 setup()**

```

template<typename COUNTERTYPE, typename Dbg>
void L4Re::Util::Counting_cap_alloc< COUNTERTYPE, Dbg >::setup (
    void * m,
    long capacity,
    long bias,
    Dbg * dbg) [inline], [protected], [noexcept]
  
```

Set up the backing memory for the allocator and the area of managed capability slots.

Parameters

<i>m</i>	Pointer to backing memory.
<i>capacity</i>	Number of capabilities that can be stored.
<i>bias</i>	First capability id to use by this allocator.
<i>dbg</i>	Logger for warnings if counter got saturated.

The allocator will manage the capability slots between `bias` and `bias + capacity - 1` (inclusive). It is the responsibility of the user to ensure that these slots are not used otherwise.

Definition at line 242 of file [counting_cap_alloc](#).

16.320.3.6 take()

```
template<typename COUNTERTYPE, typename Dbg>
void L4Re::Util::Counting_cap_alloc< COUNTERTYPE, Dbg >::take (
    L4::Cap< void > cap) [inline], [noexcept]
```

Increase the reference counter for the capability.

Parameters

<i>cap</i>	Capability, whose reference counter should be increased.
------------	--

If the capability was still free, it will be automatically allocated. Silently does nothing if the capability is not managed by this allocator.

Definition at line 296 of file [counting_cap_alloc](#).

References [L4_CAP_SHIFT](#), and [L4_UNLIKELY](#).

The documentation for this class was generated from the following file:

- [l4/re/util/counting_cap_alloc](#)

16.321 L4Re::Util::Dataspace_svr Class Reference

[Dataspace](#) server class.

```
#include <dataspace_svr>
```

Collaboration diagram for L4Re::Util::Dataspace_svr:

L4Re::Util::Dataspace_svr	
+	map()
+	map_hook()
+	take()
+	release()
+	copy()
+	clear()
+	allocate()
+	page_shift()
+	is_static()
+	map_info()

Public Member Functions

- int [map](#) (Dataspace::Offset offset, Dataspace::Map_addr local_addr, Dataspace::Flags flags, Dataspace::↔ Map_addr min_addr, Dataspace::Map_addr max_addr, [L4::lpc::Snd_fpage](#) &memory)
Map a region of the dataspace.
- virtual int [map_hook](#) (Dataspace::Offset offs, unsigned order, Dataspace::Flags flags, Dataspace::Map_addr *base, unsigned *send_order)
A hook that is called for acquiring the data to be mapped.
- virtual void [take](#) () noexcept
Take a reference to this dataspace.
- virtual unsigned long [release](#) () noexcept
Release a reference to this dataspace.
- virtual long [copy](#) ([l4_addr_t](#) dst_offs, [l4_umword_t](#) src_id, [l4_addr_t](#) src_offs, unsigned long size) noexcept
Copy from src dataspace to this destination dataspace.
- virtual long [clear](#) (unsigned long offs, unsigned long size) const noexcept
Clear a region in the dataspace.
- virtual long [allocate](#) ([l4_addr_t](#) offset, [l4_size_t](#) size, unsigned access) noexcept
Allocate a region within a dataspace.
- virtual unsigned long [page_shift](#) () const noexcept
Define the size of the flexpage to map.
- virtual bool [is_static](#) () const noexcept
Return whether the dataspace is static.
- virtual long [map_info](#) ([l4_addr_t](#) &start_addr, [l4_addr_t](#) &end_addr) noexcept
Return mapping information for no-MMU systems.

16.321.1 Detailed Description

[Dataspace](#) server class.

The default implementation of the interface provides a continuous dataspace with contiguous pages.

Definition at line 29 of file [dataspace_svr](#).

16.321.2 Member Function Documentation

16.321.2.1 [allocate\(\)](#)

```
virtual long L4Re::Util::Dataspace_svr::allocate (
    l4\_addr\_t offset,
    l4\_size\_t size,
    unsigned access) [inline], [virtual], [noexcept]
```

Allocate a region within a dataspace.

Parameters

<i>offset</i>	Offset in the dataspace, in bytes.
<i>size</i>	Size of the range, in bytes.

<i>access</i>	Access mode with which the memory backing the dataspace region should be allocated.
---------------	---

Return values

0	Success
<0	Error

Definition at line 224 of file [dataspace_svr](#).

References [L4_ENODEV](#).

16.321.2.2 clear()

```
virtual long L4Re::Util::Dataspace_svr::clear (  
    unsigned long offs,  
    unsigned long size) const [inline], [virtual], [noexcept]
```

Clear a region in the dataspace.

Parameters

<i>offs</i>	Start of the region
<i>size</i>	Size of the region

Return values

0	Success
<0	Error

Definition at line 192 of file [dataspace_svr](#).

References [L4_ERANGE](#).

16.321.2.3 copy()

```
virtual long L4Re::Util::Dataspace_svr::copy (  
    l4_addr_t dst_offs,  
    l4_umword_t src_id,  
    l4_addr_t src_offs,  
    unsigned long size) [inline], [virtual], [noexcept]
```

Copy from src dataspace to this destination dataspace.

Parameters

<i>dst_offs</i>	Offset into the destination dataspace
<i>src_id</i>	Local id of the source dataspace
<i>src_offs</i>	Offset into the source dataspace
<i>size</i>	Number of bytes to copy

Return values

≥ 0	Number of bytes copied
< 0	An error occurred. The error code may depend on the implementation.

Definition at line 175 of file [dataspace_svr](#).

References [L4_ENODEV](#).

16.321.2.4 is_static()

```
virtual bool L4Re::Util::Dataspace_svr::is_static () const [inline], [virtual], [noexcept]
```

Return whether the dataspace is static.

Returns

True if dataspace is static

Definition at line 244 of file [dataspace_svr](#).

16.321.2.5 map()

```
int L4Re::Util::Dataspace_svr::map (
    Dataspace::Offset offset,
    Dataspace::Map_addr local_addr,
    Dataspace::Flags flags,
    Dataspace::Map_addr min_addr,
    Dataspace::Map_addr max_addr,
    L4::Ipc::Snd_fpage & memory) [inline]
```

Map a region of the dataspace.

Parameters

	<i>offset</i>	Offset to start within data space
	<i>local_addr</i>	Local address to map to.
	<i>flags</i>	Dataspace flags, see L4Re::Dataspace::F::Flags .
	<i>min_addr</i>	Defines start of receive window.

	<i>max_addr</i>	Defines end of receive window.
out	<i>memory</i>	Send fpage to map

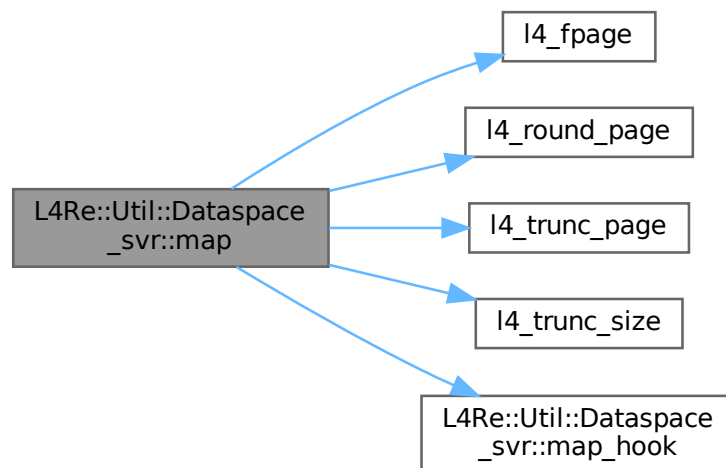
Return values

0	Success
<0	Error

Definition at line 57 of file [dataspace_svr](#).

References [L4_EOK](#), [L4_ERANGE](#), [l4_fpage\(\)](#), [L4_PAGESHIFT](#), [l4_round_page\(\)](#), [l4_trunc_page\(\)](#), [l4_trunc_size\(\)](#), and [map_hook\(\)](#).

Here is the call graph for this function:



16.321.2.6 map_hook()

```

virtual int L4Re::Util::Dataspace_svr::map_hook (
    Dataspace::Offset offs,
    unsigned order,
    Dataspace::Flags flags,
    Dataspace::Map_addr * base,
    unsigned * send_order) [inline], [virtual]

```

A hook that is called for acquiring the data to be mapped.

Parameters

<i>offs</i>	Offset in dataspace to supply
<i>order</i>	Log2-size of data to supply
<i>flags</i>	Flags for the mapping
<i>base</i>	Start address of the flexpage to be mapped
<i>send_order</i>	Order (log2 of size) of the flexpage to be mapped

Return values

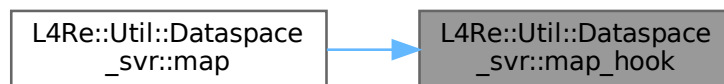
< 0	Error and the map request will be aborted with that error.
≥ 0	Success

See also[map](#)

Definition at line 136 of file [dataspace_svr](#).

Referenced by [map\(\)](#).

Here is the caller graph for this function:

**16.321.2.7 map_info()**

```
virtual long L4Re::Util::Dataspace_svr::map_info (
    l4_addr_t & start_addr,
    l4_addr_t & end_addr) [inline], [virtual], [noexcept]
```

Return mapping information for no-MMU systems.

The method is only called on no-MMU systems. It should return the address of the underlying backing buffer so that the caller might map the dataspace.

The default implementation always returns an error because the derived class must provide the required information.

See also[L4Re::Dataspace::map_info\(\)](#)

Definition at line 259 of file [dataspace_svr](#).

References [L4_EPERM](#).

16.321.2.8 `page_shift()`

```
virtual unsigned long L4Re::Util::Dataspace_svr::page_shift () const [inline], [virtual],  
[noexcept]
```

Define the size of the flexpage to map.

Returns

flexpage size

Definition at line 236 of file [dataspace_svr](#).

References [L4_LOG2_PAGESIZE](#).

16.321.2.9 `release()`

```
virtual unsigned long L4Re::Util::Dataspace_svr::release () [inline], [virtual], [noexcept]
```

Release a reference to this dataspace.

Returns

Number of references to the dataspace

Default does nothing and returns always zero.

Definition at line 160 of file [dataspace_svr](#).

16.321.2.10 `take()`

```
virtual void L4Re::Util::Dataspace_svr::take () [inline], [virtual], [noexcept]
```

Take a reference to this dataspace.

Default does nothing.

Definition at line 150 of file [dataspace_svr](#).

The documentation for this class was generated from the following file:

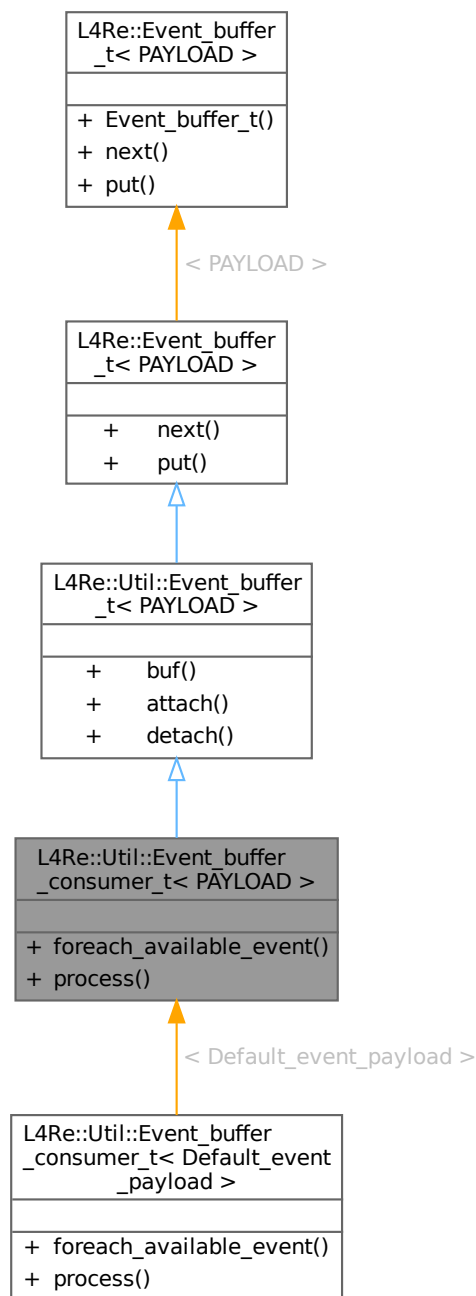
- `l4/re/util/dataspace_svr`

16.322 L4Re::Util::Event_buffer_consumer_t< PAYLOAD > Class Template Reference

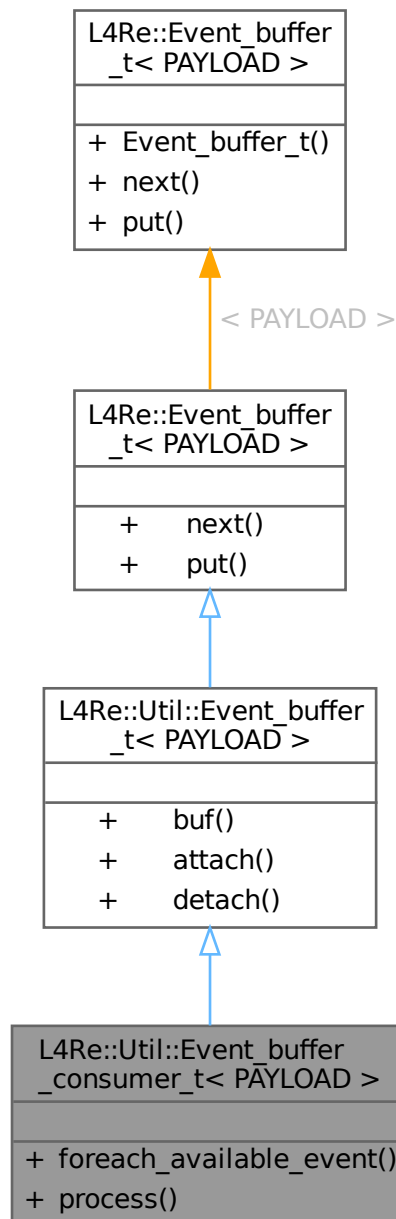
An event buffer consumer.

```
#include <event_buffer>
```

Inheritance diagram for L4Re::Util::Event_buffer_consumer_t< PAYLOAD >:



Collaboration diagram for L4Re::Util::Event_buffer_consumer_t< PAYLOAD >:



Public Member Functions

- template<typename CB, typename D>
void **foreach_available_event** (CB const &cb, D data=D())
Call function on every available event.
- template<typename CB, typename D>
void **process** (L4::Cap< L4::Irq > irq, L4::Cap< L4::Thread > thread, CB const &cb, D data=D())
Continuously wait for events and process them.

Public Member Functions inherited from [L4Re::Util::Event_buffer_t< PAYLOAD >](#)

- void * [buf](#) () const noexcept
Return the buffer.
- long [attach](#) ([L4::Cap](#)< [L4Re::Dataspace](#) > ds, [L4::Cap](#)< [L4Re::Rm](#) > rm) noexcept
Attach event buffer from address space.
- long [detach](#) ([L4::Cap](#)< [L4Re::Rm](#) > rm) noexcept
Detach event buffer from address space.

Public Member Functions inherited from [L4Re::Event_buffer_t< PAYLOAD >](#)

- Event * [next](#) () noexcept
Next event in buffer.
- bool [put](#) (Event const &ev) noexcept
Put event into buffer at current position.

16.322.1 Detailed Description

template<typename PAYLOAD>
class L4Re::Util::Event_buffer_consumer_t< PAYLOAD >

An event buffer consumer.

Definition at line 83 of file [event_buffer](#).

16.322.2 Member Function Documentation**16.322.2.1 foreach_available_event()**

```
template<typename PAYLOAD>
template<typename CB, typename D>
void L4Re::Util::Event_buffer_consumer_t< PAYLOAD >::foreach_available_event (
    CB const & cb,
    D data = D()) [inline]
```

Call function on every available event.

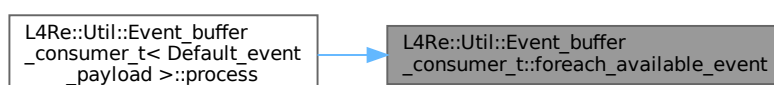
Parameters

<i>cb</i>	Function callback.
<i>data</i>	Data to pass as an argument to the callback.

Definition at line 94 of file [event_buffer](#).

Referenced by [L4Re::Util::Event_buffer_consumer_t< Default_event_payload >::process\(\)](#).

Here is the caller graph for this function:



16.322.2.2 process()

```
template<typename PAYLOAD>
template<typename CB, typename D>
void L4Re::Util::Event_buffer_consumer_t< PAYLOAD >::process (
    L4::Cap< L4::Irq > irq,
    L4::Cap< L4::Thread > thread,
    CB const & cb,
    D data = D()) [inline]
```

Continuously wait for events and process them.

Parameters

<i>irq</i>	Event signal to wait for.
<i>thread</i>	Thread capability of the thread calling this function.
<i>cb</i>	Callback function that is called for each received event.
<i>data</i>	Data to pass as an argument to the processing callback.

Note

This function never returns.

Definition at line 115 of file [event_buffer](#).

The documentation for this class was generated from the following file:

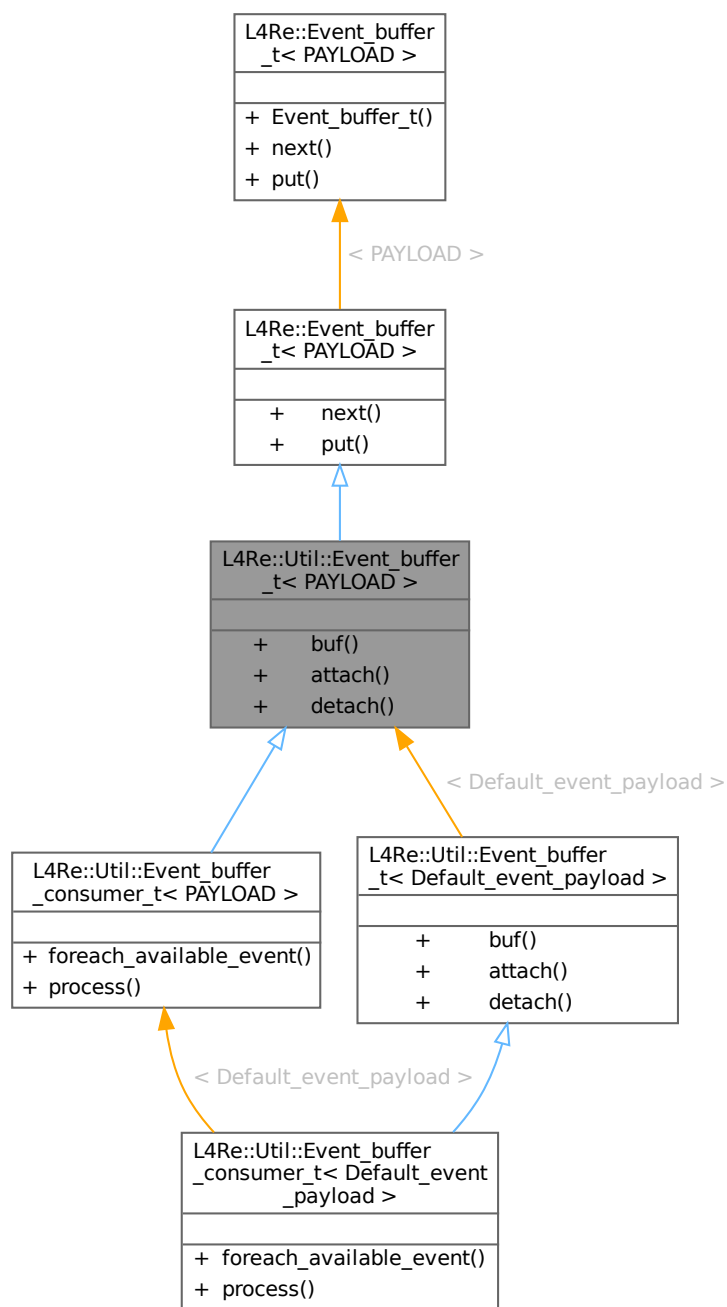
- l4/re/util/event_buffer

16.323 L4Re::Util::Event_buffer_t< PAYLOAD > Class Template Reference

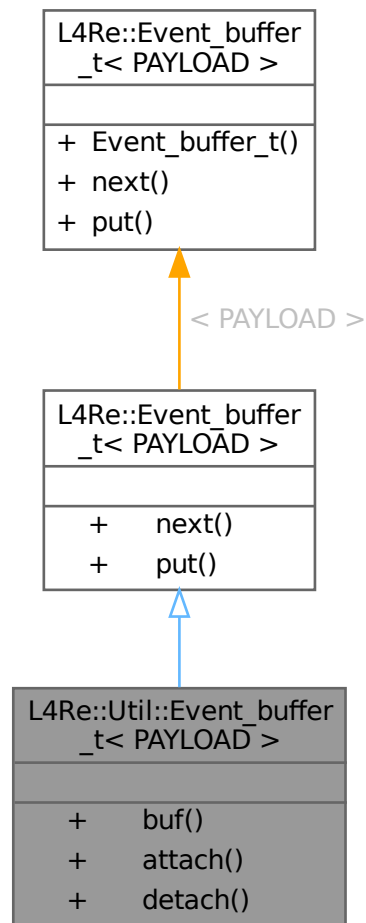
Event_buffer utility class.

```
#include <event_buffer>
```


Inheritance diagram for L4Re::Util::Event_buffer_t< PAYLOAD >:



Collaboration diagram for L4Re::Util::Event_buffer_t< PAYLOAD >:



Public Member Functions

- void * `buf()` const noexcept
Return the buffer.
- long `attach` (L4::Cap< L4Re::Dataspace > ds, L4::Cap< L4Re::Rm > rm) noexcept
Attach event buffer from address space.
- long `detach` (L4::Cap< L4Re::Rm > rm) noexcept
Detach event buffer from address space.

Public Member Functions inherited from L4Re::Event_buffer_t< PAYLOAD >

- Event * `next()` noexcept
Next event in buffer.
- bool `put` (Event const &ev) noexcept
Put event into buffer at current position.

16.323.1 Detailed Description

```
template<typename PAYLOAD>
class L4Re::Util::Event_buffer_t< PAYLOAD >
```

Event_buffer utility class.

Definition at line 25 of file [event_buffer](#).

16.323.2 Member Function Documentation

16.323.2.1 attach()

```
template<typename PAYLOAD>
long L4Re::Util::Event_buffer_t< PAYLOAD >::attach (
    L4::Cap< L4Re::Dataspace > ds,
    L4::Cap< L4Re::Rm > rm) [inline], [noexcept]
```

Attach event buffer from address space.

Parameters

<i>ds</i>	Dataspace of the event buffer.
<i>rm</i>	Region manager to attach buffer to.

Returns

0 on success, negative error code otherwise.

Definition at line 45 of file [event_buffer](#).

16.323.2.2 buf()

```
template<typename PAYLOAD>
void * L4Re::Util::Event_buffer_t< PAYLOAD >::buf () const [inline], [noexcept]
```

Return the buffer.

Returns

Pointer to the event buffer.

Definition at line 35 of file [event_buffer](#).

16.323.2.3 detach()

```
template<typename PAYLOAD>
long L4Re::Util::Event_buffer_t< PAYLOAD >::detach (
    L4::Cap< L4Re::Rm > rm) [inline], [noexcept]
```

Detach event buffer from address space.

Parameters

<i>rm</i>	Region manager to detach buffer from.
-----------	---------------------------------------

Returns

0 on success, negative error code otherwise.

Definition at line 68 of file [event_buffer](#).

The documentation for this class was generated from the following file:

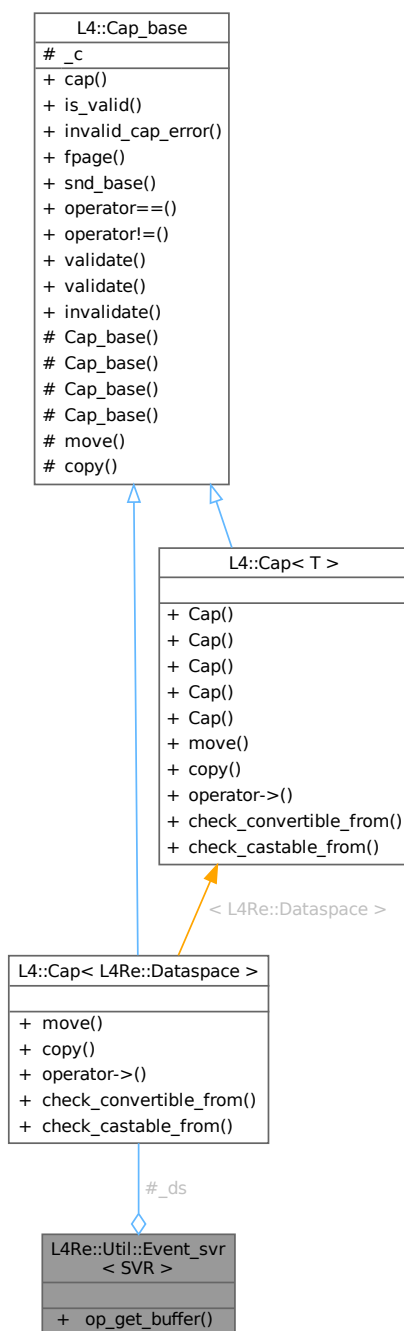
- l4/re/util/event_buffer

16.324 L4Re::Util::Event_svr< SVR > Class Template Reference

Convenience wrapper for implementing an event server.

```
#include <event_svr>
```

Collaboration diagram for L4Re::Util::Event_svr< SVR >:



Public Member Functions

- long **op_get_buffer** (L4Re::Event::Rights, L4::ipc::Cap< L4Re::Dataspace > &ds)
Handle L4Re::Event protocol.

16.324.1 Detailed Description

```
template<typename SVR>
class L4Re::Util::Event_svr< SVR >
```

Convenience wrapper for implementing an event server.

See also

[L4Re::Event](#), [L4Re::Util::Event_t](#)

Definition at line 28 of file [event_svr](#).

The documentation for this class was generated from the following file:

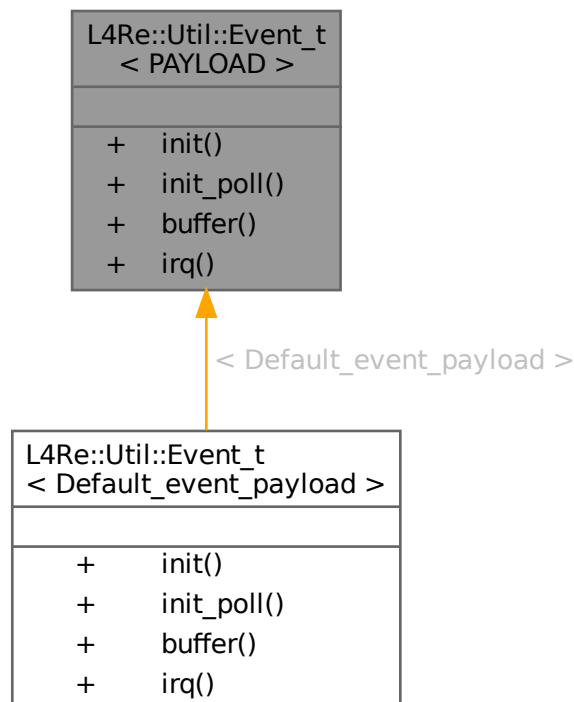
- [l4/re/util/event_svr](#)

16.325 L4Re::Util::Event_t< PAYLOAD > Class Template Reference

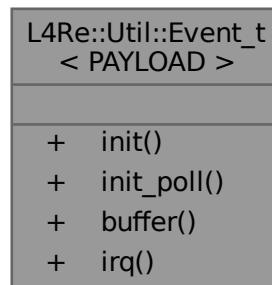
Convenience wrapper for getting access to an event object.

```
#include <event>
```

Inheritance diagram for L4Re::Util::Event_t< PAYLOAD >:



Collaboration diagram for L4Re::Util::Event_t< PAYLOAD >:



Public Types

- enum [Mode](#) { [Mode_irq](#) , [Mode_polling](#) }
Modes of operation.

Public Member Functions

- template<typename IRQ_TYPE>
int [init](#) (L4::Cap< [L4Re::Event](#) > event, [L4Re::Env](#) const *env=[L4Re::Env::env\(\)](#), [L4Re::Cap_alloc](#) *ca=&[L4Re::Util::cap_alloc](#))
Initialise an event object.
- int [init_poll](#) (L4::Cap< [L4Re::Event](#) > event, [L4Re::Env](#) const *env=[L4Re::Env::env\(\)](#), [L4Re::Cap_alloc](#) *ca=&[L4Re::Util::cap_alloc](#))
Initialise an event object in polling mode.
- [L4Re::Event_buffer_t](#)< PAYLOAD > & [buffer](#) ()
Get event buffer.
- [L4::Cap](#)< [L4::Triggerable](#) > [irq](#) () const
Get event IRQ.

16.325.1 Detailed Description

```
template<typename PAYLOAD>
class L4Re::Util::Event_t< PAYLOAD >
```

Convenience wrapper for getting access to an event object.

After calling [init\(\)](#) the class supplies the event-buffer and the associated IRQ object.

Definition at line [32](#) of file [event](#).

16.325.2 Member Enumeration Documentation

16.325.2.1 Mode

```
template<typename PAYLOAD>  
enum L4Re::Util::Event_t::Mode
```

Modes of operation.

Enumerator

Mode_irq	Create an IRQ and attach, to get notifications.
Mode_polling	Do not use an IRQ.

Definition at line 38 of file [event](#).

16.325.3 Member Function Documentation

16.325.3.1 buffer()

```
template<typename PAYLOAD>
L4Re::Event_buffer_t< PAYLOAD > & L4Re::Util::Event_t< PAYLOAD >::buffer () [inline]
```

Get event buffer.

Returns

[Event](#) buffer object.

Definition at line 148 of file [event](#).

16.325.3.2 init()

```
template<typename PAYLOAD>
template<typename IRQ_TYPE>
int L4Re::Util::Event_t< PAYLOAD >::init (
    L4::Cap< L4Re::Event > event,
    L4Re::Env const * env = L4Re::Env::env(),
    L4Re::Cap_alloc * ca = &L4Re::Util::cap_alloc) [inline]
```

Initialise an event object.

Template Parameters

<i>IRQ_TYPE</i>	Type used for handling notifications from the event provider. This must be derived from L4::Triggerable .
-----------------	---

Parameters

<i>event</i>	Capability to event.
<i>env</i>	Pointer to L4Re-Environment
<i>ca</i>	Pointer to capability allocator.

Return values

<i>0</i>	Success
<i>-L4_ENOMEM</i>	No memory to allocate required capabilities.
<i><0</i>	Other IPC errors.

Definition at line 59 of file [event](#).

16.325.3.3 `init_poll()`

```
template<typename PAYLOAD>
int L4Re::Util::Event_t< PAYLOAD >::init_poll (
    L4::Cap< L4Re::Event > event,
    L4Re::Env const * env = L4Re::Env::env(),
    L4Re::Cap_alloc * ca = &L4Re::Util::cap_alloc) [inline]
```

Initialise an event object in polling mode.

Parameters

<i>event</i>	Capability to event.
<i>env</i>	Pointer to L4Re-Environment
<i>ca</i>	Pointer to capability allocator.

Return values

<i>0</i>	Success
<i>-L4_ENOMEM</i>	No memory to allocate required capabilities.
<i><0</i>	Other IPC errors.

Definition at line 112 of file [event](#).

16.325.3.4 `irq()`

```
template<typename PAYLOAD>
L4::Cap< L4::Triggerable > L4Re::Util::Event_t< PAYLOAD >::irq () const [inline]
```

Get event IRQ.

Returns

[Event](#) IRQ.

Definition at line 155 of file [event](#).

The documentation for this class was generated from the following file:

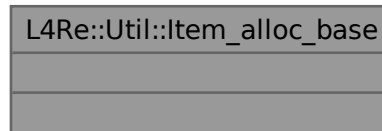
- [l4/re/util/event](#)

16.326 L4Re::Util::Item_alloc_base Class Reference

Item allocator.

```
#include <item_alloc>
```

Collaboration diagram for L4Re::Util::Item_alloc_base:



16.326.1 Detailed Description

Item allocator.

Definition at line 27 of file [item_alloc](#).

The documentation for this class was generated from the following file:

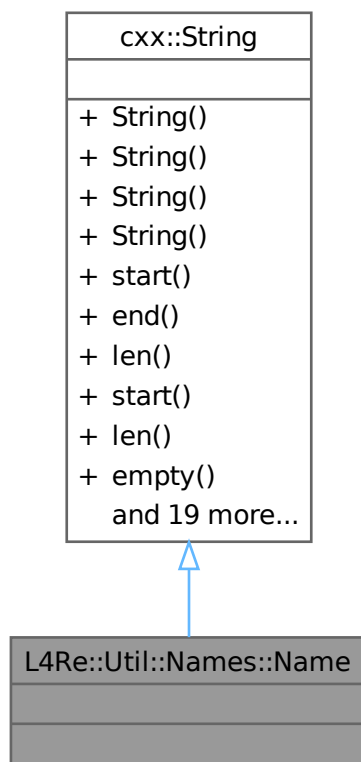
- [l4/re/util/item_alloc](#)

16.327 L4Re::Util::Names::Name Class Reference

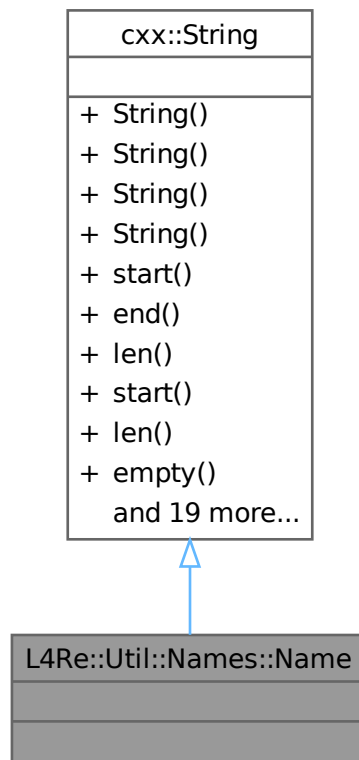
[Name](#) class.

```
#include <name_space_svr>
```

Inheritance diagram for L4Re::Util::Names::Name:



Collaboration diagram for L4Re::Util::Names::Name:



Additional Inherited Members

Public Types inherited from `cxx::String`

- `typedef char const * Index`
Character index type.

Public Member Functions inherited from `cxx::String`

- **String** (char const *s) noexcept
Initialize from a zero-terminated string.
- **String** (char const *s, unsigned long len) noexcept
Initialize from a pointer to first character and a length.
- **String** (char const *s, char const *e) noexcept
Initialize with start and end pointer.
- **String** ()
Zero-initialize. Create an invalid string.
- **Index start** () const
Pointer to first character.

- **Index end** () const
Pointer to first byte behind the string.
- int **len** () const
Length.
- void **start** (char const *s)
Set start.
- void **len** (unsigned long len)
Set length.
- bool **empty** () const
Check if the string has length zero.
- **String head** (**Index end**) const
Return prefix up to index.
- **String head** (unsigned long **end**) const
*Prefix of length **end**.*
- **String substr** (unsigned long idx, unsigned long **len**=~0UL) const
*Substring of length **len** starting at **idx**.*
- **String substr** (char const ***start**, unsigned long **len**=0) const
*Substring of length **len** starting at **start**.*
- template<typename F>
char const * **find_match** (F &&match) const
*Find matching character. **match** should be a function such as **isspace**.*
- char const * **find** (char const *c) const
*Find character. Return **end()** if not found.*
- char const * **find** (int c) const
*Find character. Return **end()** if not found.*
- char const * **rfind** (char const *c) const
*Find right-most character. Return **end()** if not found.*
- **Index starts_with** (cxx::String const &c) const
*Check if **c** is a prefix of string.*
- char const * **find** (int c, char const *s) const
*Find character **c** starting at position **s**. Return **end()** if not found.*
- char const * **find** (char const *c, char const *s) const
Find character set at position.
- char const & **operator[]** (unsigned long idx) const
*Get character at **idx**.*
- char const & **operator[]** (int idx) const
*Get character at **idx**.*
- char const & **operator[]** (**Index** idx) const
*Get character at **idx**.*
- bool **eof** (char const *s) const
*Check if pointer **s** points behind string.*
- template<typename INT>
int **from_dec** (INT *v) const
Convert decimal string to integer.
- template<typename INT>
int **from_hex** (INT *v) const
Convert hex string to integer.
- bool **operator==** (**String** const &o) const
Equality.
- bool **operator!=** (**String** const &o) const
Inequality.

16.327.1 Detailed Description

[Name](#) class.

Definition at line 28 of file [name_space_svr](#).

The documentation for this class was generated from the following file:

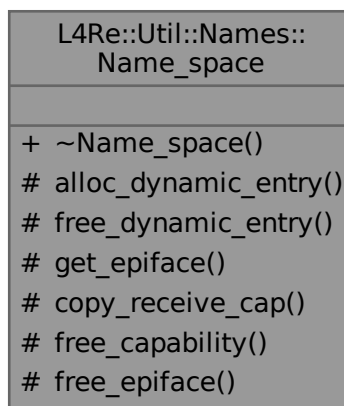
- [l4/re/util/name_space_svr](#)

16.328 L4Re::Util::Names::Name_space Class Reference

Abstract server-side implementation of the L4::Namespace interface.

```
#include <name_space_svr>
```

Collaboration diagram for L4Re::Util::Names::Name_space:



Public Member Functions

- virtual `~Name_space()`
The destructor of the derived class is responsible for freeing resources.

Protected Member Functions

- virtual `Entry * alloc_dynamic_entry (Name const &n, unsigned flags)=0`
Allocate a new entry for the given name.
- virtual `void free_dynamic_entry (Entry *e)=0`
Free an entry previously allocated with [alloc_dynamic_entry\(\)](#).
- virtual `int get_epiface (l4_umword_t data, bool is_local, L4::Epiface **lo)=0`
Return a pointer to the epiface assigned to a given label.
- virtual `int copy_receive_cap (L4::Cap< void > *cap)=0`
Return the receive capability for permanent use.
- virtual `void free_capability (L4::Cap< void > cap)=0`
Free a capability previously acquired with [copy_receive_cap\(\)](#).
- virtual `void free_epiface (L4::Epiface *epiface)=0`
Free epiface previously acquired with [get_epiface\(\)](#).

16.328.1 Detailed Description

Abstract server-side implementation of the L4::Namespace interface.

Note

The derived class is responsible for resource management through the provided interfaces. This includes freeing all resources on destruction!

Definition at line 180 of file [name_space_svr](#).

16.328.2 Member Function Documentation

16.328.2.1 alloc_dynamic_entry()

```
virtual Entry * L4Re::Util::Names::Name_space::alloc_dynamic_entry (
    Name const & n,
    unsigned flags) [protected], [pure virtual]
```

Allocate a new entry for the given name.

Parameters

<i>n</i>	Name of the entry, must be copied.
<i>flags</i>	Entry flags, see Obj::Flags .

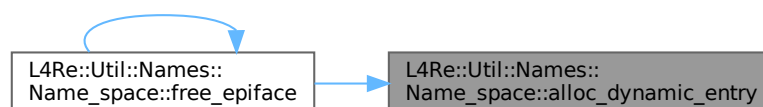
Returns

A pointer to the newly allocated entry or NULL on error.

This method is called when a new entry was received. It must allocate memory, copy *n* out of the receive buffer and wrap everything in an Entry.

Referenced by [free_epiface\(\)](#).

Here is the caller graph for this function:



16.328.2.2 copy_receive_cap()

```
virtual int L4Re::Util::Names::Name_space::copy_receive_cap (
    L4::Cap< void > * cap) [protected], [pure virtual]
```

Return the receive capability for permanent use.

Parameters

out	cap	Capability slot with the received capability. Must be permanently available until free_capability() is called.
-----	-----	--

This method is called when a new entry is registered together with a capability mapping. It must decide whether and where to store the capability and return the final capability slot. Typical implementations return the capability slot in the receive window and allocate a new receive window.

16.328.2.3 free_capability()

```
virtual void L4Re::Util::Names::Name_space::free_capability (
    L4::Cap< void > cap) [protected], [pure virtual]
```

Free a capability previously acquired with [copy_receive_cap\(\)](#).

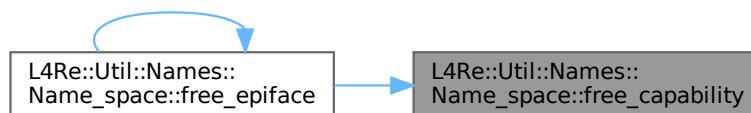
Parameters

in	cap	Capability to free.
----	-----	---------------------

Counterpart of [copy_receive_cap](#). Free the capability slot when the entry is deleted or changed.

Referenced by [free_epiface\(\)](#).

Here is the caller graph for this function:



16.328.2.4 free_dynamic_entry()

```
virtual void L4Re::Util::Names::Name_space::free_dynamic_entry (
    Entry * e) [protected], [pure virtual]
```

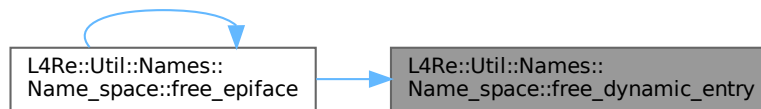
Free an entry previously allocated with [alloc_dynamic_entry\(\)](#).

Parameters

e	Entry to free.
---	----------------

Referenced by [free_epiface\(\)](#).

Here is the caller graph for this function:



16.328.2.5 free_epiface()

```
virtual void L4Re::Util::Names::Name_space::free_epiface (
    L4::Epiface * epiface) [protected], [pure virtual]
```

Free epiface previously acquired with [get_epiface\(\)](#).

Parameters

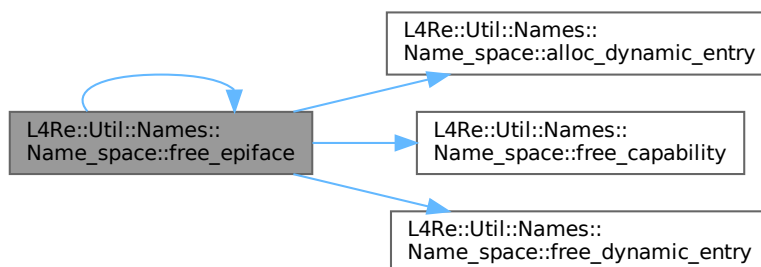
in	<i>epiface</i>	Epiface to free.
----	----------------	------------------

Called when an entry that points to an epiface is deleted allowing implementations that hold resources to free them.

References [alloc_dynamic_entry\(\)](#), [free_capability\(\)](#), [free_dynamic_entry\(\)](#), [free_epiface\(\)](#), [L4_BASE_TASK_CAP](#), [L4_EEXIST](#), [L4_ENOMEM](#), and [L4Re::Namespace::Overwrite](#).

Referenced by [free_epiface\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.328.2.6 get_epiface()

```
virtual int L4Re::Util::Names::Name_space::get_epiface (
    l4_umword_t data,
    bool is_local,
    L4::Epiface ** lo) [protected], [pure virtual]
```

Return a pointer to the epiface assigned to a given label.

Parameters

in	<i>data</i>	Label or in the local case the capability slot of the receiving capability.
in	<i>is_local</i>	If true, a local capability slot was supplied, if false the label of a locally bound IPC gate.
out	<i>lo</i>	Pointer to epiface responsible for the capability.

Returns

[L4_EOK](#) if a valid interface could be found or an error message otherwise.

This method is called when a new entry is registered and some local ID was received for the capability. In this case, the generic implementation needs to get the epiface in order to get the capability.

The callee must make sure that the epiface remains valid until `free_epiface` is called. In particular, the capability slot must not be reallocated as long as the namespace server holds a reference to the epiface.

The documentation for this class was generated from the following file:

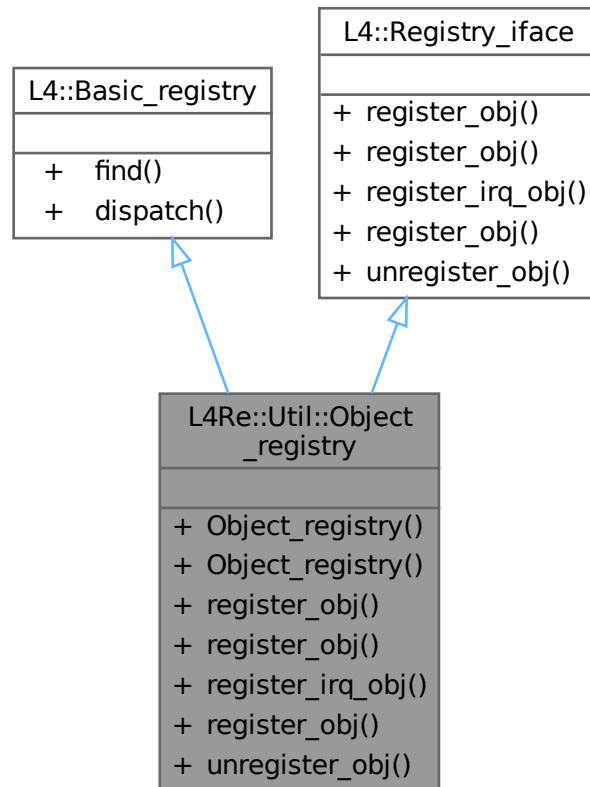
- `l4/re/util/name_space_svr`

16.329 L4Re::Util::Object_registry Class Reference

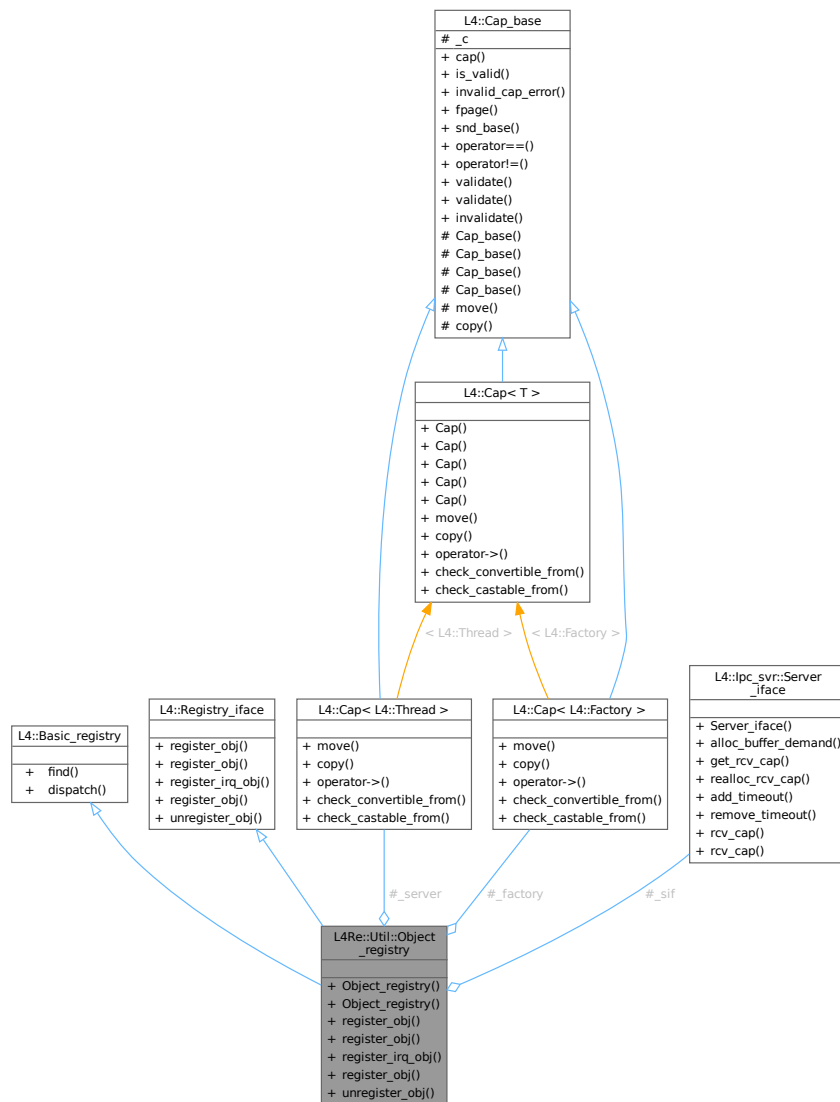
A registry that manages server objects and their attached IPC gates for a single server loop for a specific thread.

```
#include <object_registry>
```

Inheritance diagram for L4Re::Util::Object_registry:



Collaboration diagram for L4Re::Util::Object_registry:



Public Member Functions

- [Object_registry](#) ([L4::lpc_svr::Server_iface](#) *sif)
Create a registry for the main thread of the task using the default factory.
- [Object_registry](#) ([L4::lpc_svr::Server_iface](#) *sif, [L4::Cap< L4::Thread >](#) server, [L4::Cap< L4::Factory >](#) factory)
Create a registry for arbitrary threads.
- [L4::Cap< void >](#) [register_obj](#) ([L4::Epiface](#) *o, char const *service) override
Register a new server object to a pre-allocated receive endpoint.
- [L4::Cap< void >](#) [register_obj](#) ([L4::Epiface](#) *o) override
Register a new server object on a newly allocated capability.
- [L4::Cap< L4::Irq >](#) [register_irq_obj](#) ([L4::Epiface](#) *o) override
Register a handler for an interrupt.
- [L4::Cap< L4::Rcv_endpoint >](#) [register_obj](#) ([L4::Epiface](#) *o, [L4::Cap< L4::Rcv_endpoint >](#) ep) override

Register a handler for an already existing interrupt.

- void `unregister_obj` (`L4::Epiface *o`, `bool unmap=true`) override

Remove a server object from the handler list.

Additional Inherited Members

Static Public Member Functions inherited from `L4::Basic_registry`

- static `Value * find` (`l4_umword_t label`)

Get the server object for an `lpc_gate` label.

- static `l4_msgtag_t dispatch` (`l4_msgtag_t tag`, `l4_umword_t label`, `l4_utcb_t *utcb`)

The dispatch function called by the server loop.

16.329.1 Detailed Description

A registry that manages server objects and their attached IPC gates for a single server loop for a specific thread.

This class manages most of the setup of a server object. If necessary, an IPC gate is created, the specified thread is bound to the IPC gate. Incoming IPC is dispatched to the server object based on the label of the IPC gate.

The object registry is also able to manage IRQ endpoints. They require a different method for the object creation. Otherwise they are handled in the same way as IPC gates: a server object is responsible to process the incoming interrupts.

Definition at line 41 of file `object_registry`.

16.329.2 Constructor & Destructor Documentation

16.329.2.1 `Object_registry()` [1/2]

```
L4Re::Util::Object_registry::Object_registry (
    L4::Ipc_svr::Server_iface * sif) [inline], [explicit]
```

Create a registry for the main thread of the task using the default factory.

Parameters

<code>sif</code>	Server loop interface.
------------------	------------------------

Definition at line 67 of file `object_registry`.

16.329.2.2 `Object_registry()` [2/2]

```
L4Re::Util::Object_registry::Object_registry (
    L4::Ipc_svr::Server_iface * sif,
    L4::Cap< L4::Thread > server,
    L4::Cap< L4::Factory > factory) [inline]
```

Create a registry for arbitrary threads.

Parameters

<i>sif</i>	Server loop interface.
<i>server</i>	Capability to the thread that executes the server objects.
<i>factory</i>	Capability to a factory object capable of creating new IPC gates.

Definition at line 81 of file [object_registry](#).

16.329.3 Member Function Documentation

16.329.3.1 register_irq_obj()

```
L4::Cap< L4::Irq > L4Re::Util::Object_registry::register_irq_obj (
    L4::Epiface * o) [inline], [override], [virtual]
```

Register a handler for an interrupt.

Parameters

<i>o</i>	Server object that handles IRQs.
----------	----------------------------------

Return values

L4::Cap<L4::Irq>	Capability to a new IRQ object on success.
L4::Cap<L4::Irq>::Invalid	The allocation of the IRQ has failed.

The IRQ will be newly allocated using the registry's factory object. The caller must call [unregister_obj\(\)](#) to free all resources.

Implements [L4::Registry_iface](#).

Definition at line 227 of file [object_registry](#).

16.329.3.2 register_obj() [1/3]

```
L4::Cap< void > L4Re::Util::Object_registry::register_obj (
    L4::Epiface * o) [inline], [override], [virtual]
```

Register a new server object on a newly allocated capability.

Parameters

<i>o</i>	Server object that handles IPC requests.
----------	--

Return values

<i>L4::Cap<void></i>	A valid capability to a new IPC gate.
<i>L4::Cap<void>::Invalid</i>	The allocation of the IPC gate has failed.

The IPC gate will be allocated using the registry's factory. The caller must call [unregister_obj\(\)](#) to free all resources.

Implements [L4::Registry_iface](#).

Definition at line 211 of file [object_registry](#).

16.329.3.3 register_obj() [2/3]

```
L4::Cap< void > L4Re::Util::Object_registry::register_obj (
    L4::Epiface * o,
    char const * service) [inline], [override], [virtual]
```

Register a new server object to a pre-allocated receive endpoint.

Parameters

<i>o</i>	Server object that handles IPC requests.
<i>service</i>	Name of a pre-allocated receive endpoint.

Return values

<i>L4::Cap<void></i>	The capability known as <i>service</i> on success.
<i>L4::Cap<void>::Invalid</i>	No capability with the given name found.

The interface must be freed with [unregister_obj\(\)](#) by the caller to unbind the thread from the capability.

Implements [L4::Registry_iface](#).

Definition at line 194 of file [object_registry](#).

16.329.3.4 register_obj() [3/3]

```
L4::Cap< L4::Rcv_endpoint > L4Re::Util::Object_registry::register_obj (
    L4::Epiface * o,
    L4::Cap< L4::Rcv_endpoint > ep) [inline], [override], [virtual]
```

Register a handler for an already existing interrupt.

Parameters

<i>o</i>	Server object that handles the IPC.
<i>ep</i>	Capability to a receive endpoint, may be a hardware or software interrupt or an IPC gate.

Return values

L4::Cap<L4::Rcv_endpoint>	Capability ep on success.
L4::Cap<L4::Rcv_endpoint>::Invalid	The IRQ attach operation has failed.

The interface must be freed with [unregister_obj\(\)](#) by the caller to unbind the thread from the capability.

Implements [L4::Registry_iface](#).

Definition at line 246 of file [object_registry](#).

16.329.3.5 unregister_obj()

```
void L4Re::Util::Object_registry::unregister_obj (
    L4::Epiface * o,
    bool unmap = true) [inline], [override], [virtual]
```

Remove a server object from the handler list.

Parameters

<i>o</i>	Server object to unbind.
<i>unmap</i>	Specifies if the object capability shall be unmapped (true) or not. The default (true) is to unmap the capability.

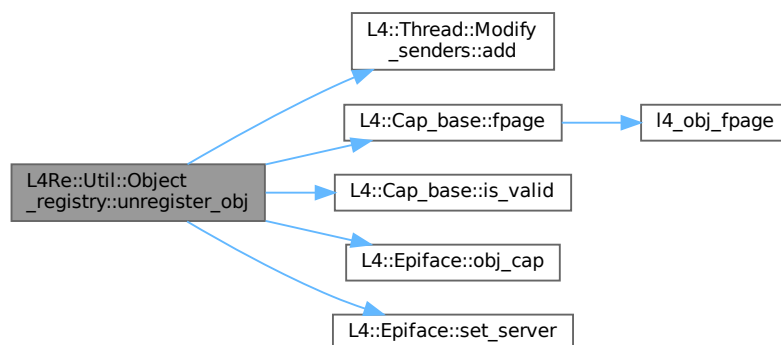
The capability used by the server object will be unmapped if `unmap` is true.

Implements [L4::Registry_iface](#).

Definition at line 262 of file [object_registry](#).

References [L4::Thread::Modify_senders::add\(\)](#), [L4Re::Util::cap_alloc](#), [L4::Cap_base::fpage\(\)](#), [L4::Cap_base::Invalid](#), [L4::Cap_base::is_valid\(\)](#), [L4_FP_ALL_SPACES](#), [L4::Epiface::obj_cap\(\)](#), and [L4::Epiface::set_server\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

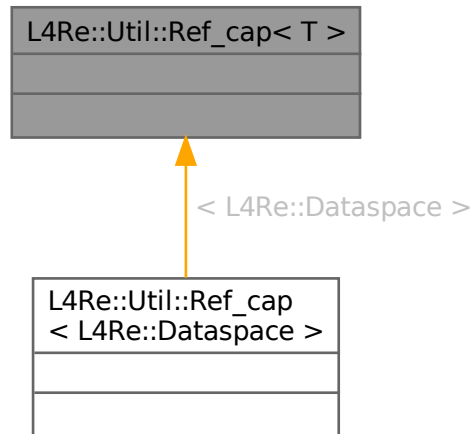
- `I4/re/util/object_registry`

16.330 L4Re::Util::Ref_cap< T > Struct Template Reference

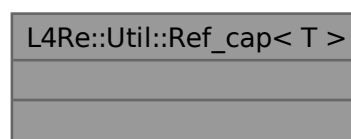
Automatic capability that implements automatic free and unmap of the capability selector.

```
#include <cap_alloc>
```

Inheritance diagram for L4Re::Util::Ref_cap< T >:



Collaboration diagram for L4Re::Util::Ref_cap< T >:



16.330.1 Detailed Description

```
template<typename T>
struct L4Re::Util::Ref_cap< T >
```

Automatic capability that implements automatic free and unmap of the capability selector.

Template Parameters

<i>T</i>	Type of the object that is referred by the capability.
----------	--

This kind of automatic capability implements a counted reference to a capability selector. The capability shall be unmapped and freed when the reference count in the allocator goes to zero.

Usage:

```
L4Re::Util::Ref_cap<L4Re::Dataspace>::Cap global_ds_cap;

{
    L4Re::Util::Ref_cap<L4Re::Dataspace>::Cap
        ds_cap(L4Re::Util::cap_alloc.alloc<L4Re::Dataspace>());
    // reference count for the allocated cap selector is now 1

    // use the dataspace cap
    L4Re::chksys(mem_alloc->alloc(4096, ds_cap.get()));

    global_ds_cap = ds_cap;
    // reference count is now 2
    ...
}
// reference count dropped to 1 (ds_cap is no longer existing).
```

Definition at line 142 of file [cap_alloc](#).

The documentation for this struct was generated from the following file:

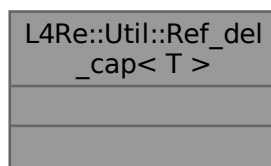
- [l4/re/util/cap_alloc](#)

16.331 L4Re::Util::Ref_del_cap< T > Struct Template Reference

Automatic capability that implements automatic free and unmap+delete of the capability selector.

```
#include <cap_alloc>
```

Collaboration diagram for L4Re::Util::Ref_del_cap< T >:



16.331.1 Detailed Description

```
template<typename T>
struct L4Re::Util::Ref_del_cap< T >
```

Automatic capability that implements automatic free and unmap+delete of the capability selector.

Template Parameters

<i>T</i>	Type of the object that is referred by the capability.
----------	--

This kind of automatic capability implements a counted reference to a capability selector. The capability shall be unmapped and freed when the reference count in the allocator goes to zero. The main difference to [Ref_cap](#) is that the unmap is done with the deletion flag enabled and this leads to the deletion of the object if the current task holds appropriate deletion rights.

Usage:

```
L4Re::Util::Ref_del_cap<L4Re::Dataspace>::Cap global_ds_cap;

{
    L4Re::Util::Ref_del_cap<L4Re::Dataspace>::Cap
        ds_cap(L4Re::Util::cap_alloc.alloc<L4Re::Dataspace>());
    // reference count for the allocated cap selector is now 1

    // use the dataspace cap
    L4Re::chksys(mem_alloc->alloc(4096, ds_cap.get()));

    global_ds_cap = ds_cap;
    // reference count is now 2
    ...
}
// reference count dropped to 1 (ds_cap is no longer existing).
...
global_ds_cap = L4_INVALID_CAP;
// reference count dropped to 0 (data space shall be deleted).
```

Definition at line 183 of file [cap_alloc](#).

The documentation for this struct was generated from the following file:

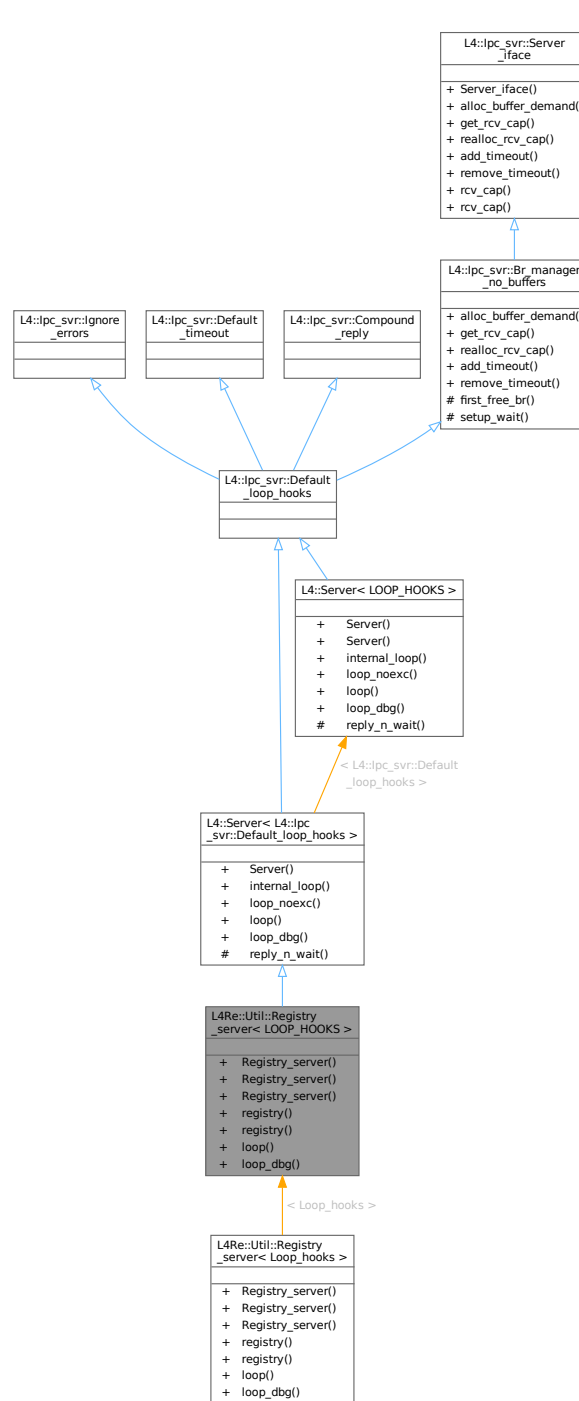
- [l4/re/util/cap_alloc](#)

16.332 L4Re::Util::Registry_server< LOOP_HOOKS > Class Template Reference

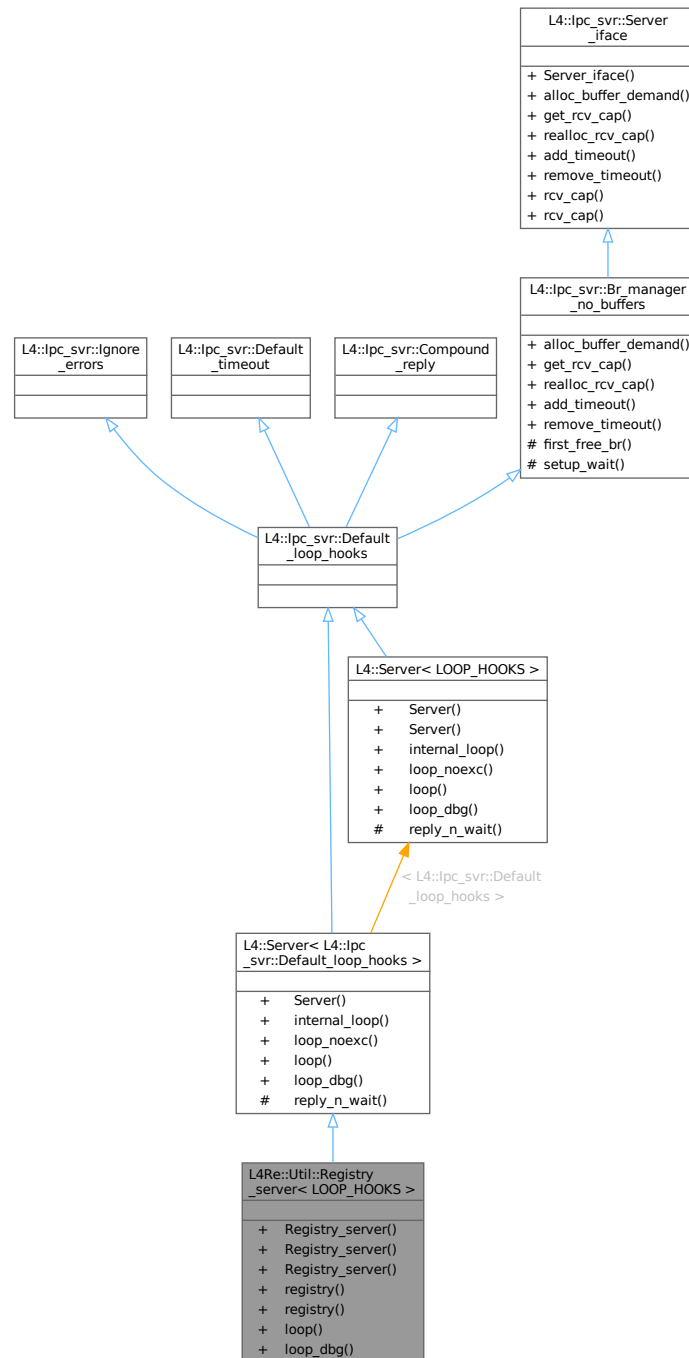
A server loop object which has a [Object_registry](#) included.

```
#include <object_registry>
```

Inheritance diagram for L4Re::Util::Registry_server< LOOP_HOOKS >:



Collaboration diagram for L4Re::Util::Registry_server< LOOP_HOOKS >:



Public Member Functions

- `Registry_server()`
Create a new server loop object for the main thread of the task.
- `Registry_server(l4_utcb_t *, L4::Cap< L4::Thread > server, L4::Cap< L4::Factory > factory)`
Create a new server loop object for an arbitrary thread and factory.
- `Registry_server(L4::Cap< L4::Thread > server, L4::Cap< L4::Factory > factory)`

- *Create a new server loop object for an arbitrary thread and factory.*
- `Object_registry` const * **registry** () const
Return registry of this server loop.
- `Object_registry` * **registry** ()
Return registry of this server loop.
- void `L4_NORETURN` **loop** (`l4_utcb_t` *utcb=`l4_utcb`())
Start the server loop.
- template<typename Printer>
void `L4_NORETURN` **loop_dbg** (Printer printer, `l4_utcb_t` *utcb=`l4_utcb`())
Start the server loop with error printing.

Public Member Functions inherited from `L4::Server< L4::lpc_svr::Default_loop_hooks >`

- `Server` (`l4_utcb_t` *)
Initializes the server loop.
- `L4_NORETURN` void **internal_loop** (DISPATCH dispatch, `l4_utcb_t` *)
The server loop.
- `L4_NORETURN` void **loop_noexc** (R r, `l4_utcb_t` *u=`l4_utcb`())
Server loop without exception handling.
- `L4_NORETURN` void **loop** (R r, `l4_utcb_t` *u=`l4_utcb`())
Server loop with internal exception handling.
- `L4_NORETURN` void **loop_dbg** (R r, Printer p, `l4_utcb_t` *u=`l4_utcb`())
Server loop with internal exception handling including message printing.

Public Member Functions inherited from `L4::lpc_svr::Br_manager_no_buffers`

- int **alloc_buffer_demand** (Demand const &demand) override
Tells the server to allocate buffers for the given demand.
- `L4::Cap< void >` **get_rcv_cap** (int) const override
Returns `L4::Cap<void>::Invalid`, we have no buffer management.
- int **realloc_rcv_cap** (int) override
Returns `-L4_ENOMEM`, we have no buffer management.
- int **add_timeout** (Timeout *, `l4_kernel_clock_t`) override
Returns `-L4_ENOSYS`, we have no timeout queue.
- int **remove_timeout** (Timeout *) override
Returns `-L4_ENOSYS`, we have no timeout queue.

Public Member Functions inherited from `L4::lpc_svr::Server_iface`

- `Server_iface` ()
Make a server interface.
- template<typename T>
`L4::Cap< T >` **rcv_cap** (int index) const
Get given receive buffer as typed capability.
- `L4::Cap< void >` **rcv_cap** (int index) const
Get receive cap with the given index as generic (void) type.

Additional Inherited Members

Public Types inherited from [L4::lpc_svr::Server_iface](#)

- typedef [L4::Type_info::Demand](#) **Demand**
Data type expressing server-side demand for receive buffers.

Protected Member Functions inherited from [L4::Server< L4::lpc_svr::Default_loop_hooks >](#)

- [l4_msgtag_t](#) **reply_n_wait** ([l4_msgtag_t](#) reply, [l4_umword_t](#) *p, [l4_utcb_t](#) *)
Internal implementation for reply and wait.

Protected Member Functions inherited from [L4::lpc_svr::Br_manager_no_buffers](#)

- unsigned **first_free_br** () const
Returns 1 as first free buffer.
- void **setup_wait** ([l4_utcb_t](#) *utcb, [L4::lpc_svr::Reply_mode](#))
Setup wait function for the server loop ([Server<>](#)).

16.332.1 Detailed Description

```
template<typename LOOP_HOOKS = L4::lpc_svr::Default_loop_hooks>
class L4Re::Util::Registry_server< LOOP_HOOKS >
```

A server loop object which has a [Object_registry](#) included.

Examples

[examples/clntsrv/src/server.cc](#), [examples/libs/l4re/c++/shared_ds/ds_srv.cc](#), and [examples/libs/l4re/streammap/server.cc](#).

Definition at line 293 of file [object_registry](#).

16.332.2 Constructor & Destructor Documentation

16.332.2.1 Registry_server() [1/3]

```
template<typename LOOP_HOOKS = L4::lpc_svr::Default_loop_hooks>
L4Re::Util::Registry_server< LOOP_HOOKS >::Registry_server () [inline]
```

Create a new server loop object for the main thread of the task.

Precondition

Must be called from the main thread or behaviour is undefined.

Definition at line 305 of file [object_registry](#).

16.332.2.2 Registry_server() [2/3]

```
template<typename LOOP_HOOKS = L4::lpc_svr::Default_loop_hooks>
L4Re::Util::Registry_server< LOOP_HOOKS >::Registry_server (
    l4_utcb_t * ,
    L4::Cap< L4::Thread > server,
    L4::Cap< L4::Factory > factory) [inline]
```

Create a new server loop object for an arbitrary thread and factory.

Parameters

<i>server</i>	Capability to thread running the server loop.
<i>factory</i>	Capability to factory object used to create new IPC gates.

Deprecated Note that this variant of the constructor is deprecated, please do not supply the UTCB pointer, it's not used.

Definition at line 317 of file [object_registry](#).

16.332.2.3 Registry_server() [3/3]

```
template<typename LOOP_HOOKS = L4::Ipc_svr::Default_loop_hooks>
L4Re::Util::Registry_server< LOOP_HOOKS >::Registry_server (
    L4::Cap< L4::Thread > server,
    L4::Cap< L4::Factory > factory) [inline]
```

Create a new server loop object for an arbitrary thread and factory.

Parameters

<i>server</i>	Capability to thread running the server loop.
<i>factory</i>	Capability to factory object used to create new IPC gates.

Definition at line 328 of file [object_registry](#).

16.332.3 Member Function Documentation

16.332.3.1 loop()

```
template<typename LOOP_HOOKS = L4::Ipc_svr::Default_loop_hooks>
void L4_NORETURN L4Re::Util::Registry_server< LOOP_HOOKS >::loop (
    l4_utcb_t * utcb = l4_utcb()) [inline]
```

Start the server loop.

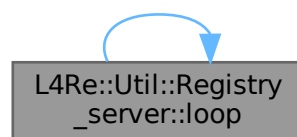
Parameters

<i>utcb</i>	The UTCB of the thread running the server loop, defaults to l4_utcb() .
-------------	---

Definition at line 344 of file [object_registry](#).

Referenced by [L4Re::Util::Registry_server< Loop_hooks >::loop\(\)](#).

Here is the caller graph for this function:



16.332.3.2 loop_dbg()

```
template<typename LOOP_HOOKS = L4::Ipc_svr::Default_loop_hooks>
template<typename Printer>
void L4_NORETURN L4Re::Util::Registry_server< LOOP_HOOKS >::loop_dbg (
    Printer printer,
    l4_utcb_t * utcb = l4_utcb()) [inline]
```

Start the server loop with error printing.

Template Parameters

<i>Printer</i>	The printer type.
----------------	-------------------

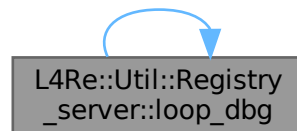
Parameters

<i>printer</i>	The printer object on which printf() is called.
<i>utcb</i>	The UTCB of the thread running the server loop, defaults to l4_utcb() .

Definition at line [356](#) of file [object_registry](#).

Referenced by [L4Re::Util::Registry_server< Loop_hooks >::loop_dbg\(\)](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

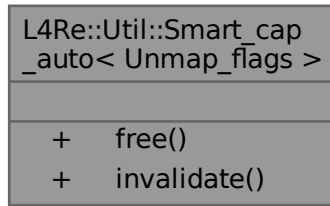
- `l4/re/util/object_registry`

16.333 L4Re::Util::Smart_cap_auto< Unmap_flags > Class Template Reference

Helper for [Unique_cap](#) and [Unique_del_cap](#).

```
#include <cap_alloc>
```

Collaboration diagram for L4Re::Util::Smart_cap_auto< Unmap_flags >:



Static Public Member Functions

- static void **free** ([L4::Cap_base](#) &c)
Free the provided capability.
- static void **invalidate** ([L4::Cap_base](#) &c)
Invalidate the provided capability.

16.333.1 Detailed Description

```
template<unsigned long Unmap_flags = L4_FP_ALL_SPACES>
class L4Re::Util::Smart_cap_auto< Unmap_flags >
```

Helper for [Unique_cap](#) and [Unique_del_cap](#).

Definition at line 45 of file [cap_alloc](#).

The documentation for this class was generated from the following file:

- [l4/re/util/cap_alloc](#)

16.334 L4Re::Util::Smart_count_cap< Unmap_flags > Class Template Reference

Helper for [Ref_cap](#) and [Ref_del_cap](#).

```
#include <cap_alloc>
```

Collaboration diagram for L4Re::Util::Smart_count_cap< Unmap_flags >:

L4Re::Util::Smart_count_cap< Unmap_flags >	
+	free()
+	invalidate()
+	copy()

Static Public Member Functions

- static void **free** ([L4::Cap_base](#) &c) noexcept
Free operation for [L4::Smart_cap](#) (decrement ref count and delete if 0).
- static void **invalidate** ([L4::Cap_base](#) &c) noexcept
Invalidate operation for [L4::Smart_cap](#).
- static [L4::Cap_base](#) **copy** ([L4::Cap_base](#) const &src)
Copy operation for [L4::Smart_cap](#) (increment ref count).

16.334.1 Detailed Description

```
template<unsigned long Unmap_flags = L4_FP_ALL_SPACES>
class L4Re::Util::Smart_count_cap< Unmap_flags >
```

Helper for [Ref_cap](#) and [Ref_del_cap](#).

Definition at line 76 of file [cap_alloc](#).

The documentation for this class was generated from the following file:

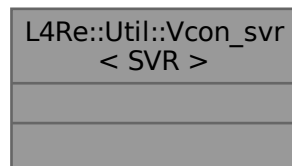
- [l4/re/util/cap_alloc](#)

16.335 L4Re::Util::Vcon_svr< SVR > Class Template Reference

[Console](#) server template class.

```
#include <vcon_svr>
```

Collaboration diagram for L4Re::Util::Vcon_svr< SVR >:



16.335.1 Detailed Description

```
template<typename SVR>
class L4Re::Util::Vcon_svr< SVR >
```

[Console](#) server template class.

This template uses `vcon_write()` and `vcon_read()` to get and deliver data from the implementor.

`vcon_read()` needs to update the status argument with the `L4_vcon_read_stat` flags, especially the `L4_VCON_READ_STAT_DONE` flag to indicate that there's nothing more to read for the other end.

`vcon_write()` gets the live data from the UTCB. Make sure to copy out the data before using the UTCB again.

The size parameter of both functions is given in bytes.

Definition at line [36](#) of file [vcon_svr](#).

The documentation for this class was generated from the following file:

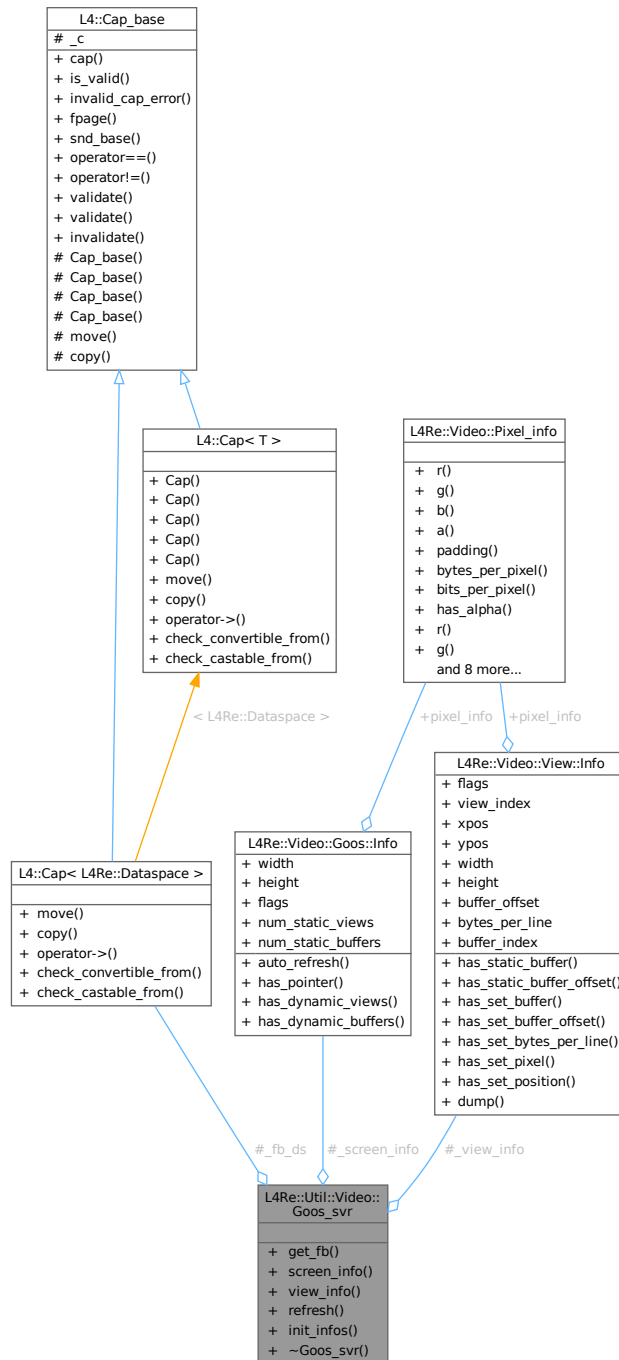
- `l4/re/util/vcon_svr`

16.336 L4Re::Util::Video::Goos_svr Class Reference

Goos server class.

```
#include <goos_svr>
```

Collaboration diagram for L4Re::Util::Video::Goos_svr:



Public Member Functions

- **L4::Cap< L4Re::Dataspace >** **get_fb ()** const
Return framebuffer memory dataspace.
- **L4Re::Video::Goos::Info** const * **screen_info ()** const
Goos information structure.
- **L4Re::Video::View::Info** const * **view_info ()** const

View information structure.

- virtual int [refresh](#) (int x, int y, int w, int h)

Refresh area of the framebuffer.

- void [init_infos](#) ()

Initialize the view information structure of this object.

- virtual \sim [Goos_svr](#) ()

Destructor.

Protected Attributes

- [L4::Cap](#)< [L4Re::Dataspace](#) > [_fb_ds](#)

Goos memory dataspace.

- [L4Re::Video::Goos::Info](#) [_screen_info](#)

Goos information.

- [L4Re::Video::View::Info](#) [_view_info](#)

View information.

16.336.1 Detailed Description

Goos server class.

Definition at line 25 of file [goos_svr](#).

16.336.2 Member Function Documentation

16.336.2.1 [get_fb\(\)](#)

```
L4::Cap< L4Re::Dataspace > L4Re::Util::Video::Goos\_svr::get\_fb () const [inline]
```

Return framebuffer memory dataspace.

Returns

Goos memory dataspace

Definition at line 42 of file [goos_svr](#).

References [_fb_ds](#).

16.336.2.2 [init_infos\(\)](#)

```
void L4Re::Util::Video::Goos\_svr::init\_infos () [inline]
```

Initialize the view information structure of this object.

This function initializes the view info structure of this goos object based on the information in the goos information, i.e. the width, height and pixel_info of the goos information has to contain valid values before calling [init_info\(\)](#).

Definition at line 78 of file [goos_svr](#).

References [_screen_info](#), and [_view_info](#).

16.336.2.3 refresh()

```
virtual int L4Re::Util::Video::Goos_svr::refresh (  
    int x,  
    int y,  
    int w,  
    int h) [inline], [virtual]
```

Refresh area of the framebuffer.

Parameters

<i>x</i>	X coordinate (pixels)
<i>y</i>	Y coordinate (pixels)
<i>w</i>	Width of area in pixels
<i>h</i>	Height of area in pixels

Returns

0 on success, negative error code otherwise

Definition at line 66 of file [goos_svr](#).

References [L4_ENOSYS](#).

16.336.2.4 screen_info()

```
L4Re::Video::Goos::Info const * L4Re::Util::Video::Goos_svr::screen_info () const [inline]
```

Goos information structure.

Returns

Return goos information structure.

Definition at line 48 of file [goos_svr](#).

References [_screen_info](#).

16.336.2.5 view_info()

```
L4Re::Video::View::Info const * L4Re::Util::Video::Goos_svr::view_info () const [inline]
```

View information structure.

Returns

Return view information structure.

Definition at line 54 of file [goos_svr](#).

References [_view_info](#).

The documentation for this class was generated from the following file:

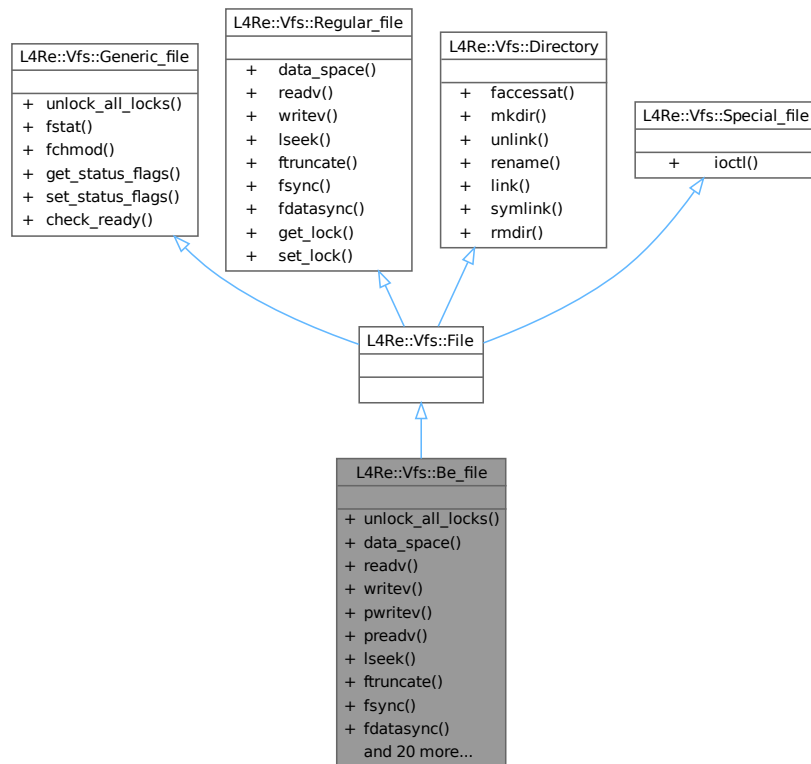
- [l4/re/util/video/goos_svr](#)

16.337 L4Re::Vfs::Be_file Class Reference

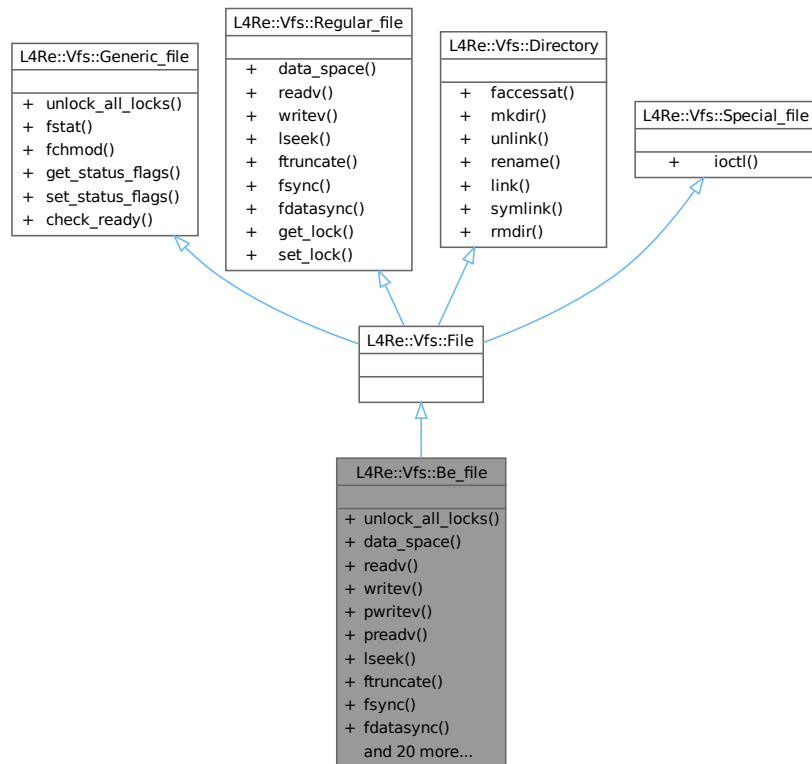
Boiler plate class for implementing an open file for [L4Re::Vfs](#).

```
#include <backend>
```

Inheritance diagram for L4Re::Vfs::Be_file:



Collaboration diagram for L4Re::Vfs::Be_file:



Public Member Functions

- `int unlock_all_locks ()` noexcept override
Unlock all locks on the file.
- `L4::Cap< L4Re::Dataspace > data_space ()` noexcept override
Get an [L4Re::Dataspace](#) object for the file.
- `ssize_t readv (const struct iovec *, int)` noexcept override
Default backend for POSIX read and readv functions.
- `ssize_t writev (const struct iovec *, int)` noexcept override
Default backend for POSIX write and writev functions.
- `ssize_t pwritev (const struct iovec *, int, off64_t)` noexcept override
Default backend for POSIX pwrite and pwritev functions.
- `ssize_t preadv (const struct iovec *, int, off64_t)` noexcept override
Default backend for POSIX pread and preadv functions.
- `off64_t lseek (off64_t, int)` noexcept override
Default backend for POSIX seek and lseek functions.
- `int ftruncate (off64_t)` noexcept override
Default backend for the POSIX truncate, ftruncate and similar functions.
- `int fsync ()` const noexcept override
Default backend for POSIX fsync.
- `int fdatasync ()` const noexcept override
Default backend for POSIX fdatasync.

- int **ioctl** (unsigned long, va_list) noexcept override
Default backend for POSIX ioctl.
- int **fstat** (struct stat64 *) const noexcept override
Get status information for the file.
- int **fchmod** (mode_t) noexcept override
Default backend for POSIX chmod and fchmod.
- int **get_status_flags** () const noexcept override
Default backend for POSIX fcntl subfunctions.
- int **set_status_flags** (long) noexcept override
Default backend for POSIX fcntl subfunctions.
- int **get_lock** (struct flock64 *) noexcept override
Default backend for POSIX fcntl subfunctions.
- int **set_lock** (struct flock64 *, bool) noexcept override
Default backend for POSIX fcntl subfunctions.
- int **faccessat** (const char *, int, int) noexcept override
Default backend for POSIX access and faccessat functions.
- int **fchmodat** (const char *, mode_t, int) noexcept override
Default backend for POSIX fchmodat function.
- int **utime** (const struct utimbuf *) noexcept override
Default backend for POSIX utime.
- int **utimes** (const struct timeval[2]) noexcept override
Default backend for POSIX utimes.
- int **utimensat** (const char *, const struct timespec[2], int) noexcept override
Default backend for POSIX utimensat.
- int **mkdir** (const char *, mode_t) noexcept override
Default backend for POSIX mkdir and mkdirat.
- int **unlink** (const char *) noexcept override
Default backend for POSIX unlink, unlinkat.
- int **rename** (const char *, const char *) noexcept override
Default backend for POSIX rename, renameat.
- int **link** (const char *, const char *) noexcept override
Default backend for POSIX link, linkat.
- int **symlink** (const char *, const char *) noexcept override
Default backend for POSIX symlink, symlinkat.
- int **rmdir** (const char *) noexcept override
Default backend for POSIX rmdir, rmdirat.
- ssize_t **readlink** (char *, size_t) override
Default backend for POSIX readlink, readlinkat.
- bool **check_ready** ([Ready_type](#)) noexcept override
Default implementation of a readiness check.

Additional Inherited Members

Public Types inherited from [L4Re::Vfs::Generic_file](#)

- enum [Ready_type](#) : unsigned
Type of I/O operation/condition a file can indicate readiness.

16.337.1 Detailed Description

Boiler plate class for implementing an open file for [L4Re::Vfs](#).

This class may be used as a base class for everything that a POSIX file descriptor may point to. This are things such as regular files, directories, special device files, streams, pipes, and so on.

Examples

[tmpfs/lib/src/fs.cc](#).

Definition at line 30 of file [backend](#).

16.337.2 Member Function Documentation

16.337.2.1 `check_ready()`

```
bool L4Re::Vfs::Be_file::check_ready (
    Ready_type ) [inline], [override], [virtual], [noexcept]
```

Default implementation of a readiness check.

By default, we assume a file is not ready for an I/O operation/condition since the proper semantics of that relies on the backend.

Returns

Always false.

Implements [L4Re::Vfs::Generic_file](#).

Definition at line 217 of file [backend](#).

16.337.2.2 `data_space()`

```
L4::Cap< L4Re::Dataspace > L4Re::Vfs::Be_file::data_space () [inline], [override], [virtual],
[noexcept]
```

Get an [L4Re::Dataspace](#) object for the file.

This is used as a backend for POSIX mmap and mmap2 functions.

Note

mmap is not possible if the function returns an invalid capability.

Returns

A capability to an [L4Re::Dataspace](#) that represents the file contents in an [L4Re](#) way.

Implements [L4Re::Vfs::Regular_file](#).

Definition at line 47 of file [backend](#).

References [L4::Cap_base::Invalid](#).

16.337.2.3 `fstat()`

```
int L4Re::Vfs::Be_file::fstat (
    struct stat64 * buf) const [inline], [override], [virtual], [noexcept]
```

Get status information for the file.

This is the backend for POSIX fstat, stat, fstat64 and friends.

Parameters

out	buf	This buffer is filled with the status information.
-----	-----	--

Returns

0 on success, or <0 on error.

Implements [L4Re::Vfs::Generic_file](#).

Definition at line 86 of file [backend](#).

16.337.2.4 unlock_all_locks()

```
int L4Re::Vfs::Be_file::unlock_all_locks () [inline], [override], [virtual], [noexcept]
```

Unlock all locks on the file.

Note

All locks means all locks independent of which file the locks were taken by.

This method is called by the POSIX close implementation to get the POSIX semantics of releasing all locks taken by this application on a close for any fd referencing the real file.

Returns

0 on success, or <0 on error.

Implements [L4Re::Vfs::Generic_file](#).

Definition at line 43 of file [backend](#).

The documentation for this class was generated from the following file:

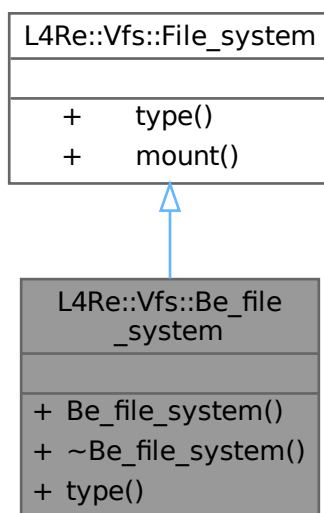
- [l4/l4re_vfs/backend](#)

16.338 L4Re::Vfs::Be_file_system Class Reference

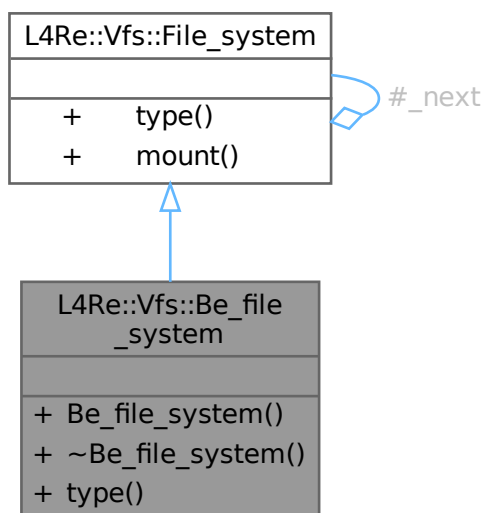
Boilerplate class for implementing a [L4Re::Vfs::File_system](#).

```
#include <backend>
```

Inheritance diagram for L4Re::Vfs::Be_file_system:



Collaboration diagram for L4Re::Vfs::Be_file_system:



Public Member Functions

- [Be_file_system](#) (char const *fstype) noexcept
Create a file-system object for the given fstype.
- [~Be_file_system](#) () noexcept
Destroy a file-system object.
- char const * [type](#) () const noexcept override
Return the file-system type.

Public Member Functions inherited from [L4Re::Vfs::File_system](#)

- virtual int [mount](#) (char const *source, unsigned long mountflags, void const *data, [cxx::Ref_ptr](#)< [File](#) > *dir) noexcept=0
Create a directory object dir representing source mounted with this file system.

16.338.1 Detailed Description

Boilerplate class for implementing a [L4Re::Vfs::File_system](#).

This class already takes care of registering and unregistering the file system in the global registry and implements the [type\(\)](#) method.

Examples

[tmpfs/lib/src/fs.cc](#).

Definition at line 310 of file [backend](#).

16.338.2 Constructor & Destructor Documentation

16.338.2.1 [Be_file_system](#)()

```
L4Re::Vfs::Be_file_system::Be_file_system (
    char const * fstype) [inline], [explicit], [noexcept]
```

Create a file-system object for the given *fstype*.

Parameters

<i>fstype</i>	The type that type() shall return.
---------------	--

This constructor takes care of registering the file system in the registry of [L4Re::Vfs::vfs_ops](#).

Definition at line 324 of file [backend](#).

16.338.2.2 ~Be_file_system()

```
L4Re::Vfs::Be_file_system::~Be_file_system () [inline], [noexcept]
```

Destroy a file-system object.

This destructor takes care of removing this file system from the registry of L4Re::Vfs::vfs_ops.

Definition at line 336 of file [backend](#).

16.338.3 Member Function Documentation

16.338.3.1 type()

```
char const * L4Re::Vfs::Be_file_system::type () const [inline], [override], [virtual], [noexcept]
```

Return the file-system type.

Returns the file-system type given as *fstype* in the constructor.

Implements [L4Re::Vfs::File_system](#).

Definition at line 346 of file [backend](#).

The documentation for this class was generated from the following file:

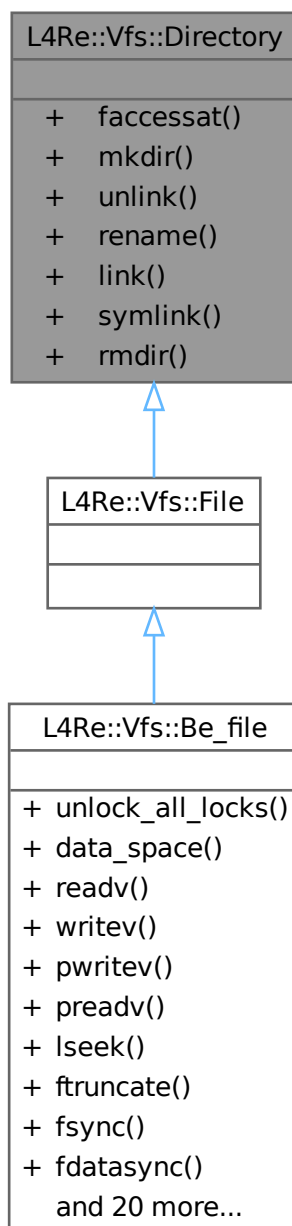
- [l4/l4re_vfs/backend](#)

16.339 L4Re::Vfs::Directory Class Reference

Interface for a POSIX file that is a directory.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::Directory:



Collaboration diagram for L4Re::Vfs::Directory:

L4Re::Vfs::Directory	
+	faccessat()
+	mkdir()
+	unlink()
+	rename()
+	link()
+	symlink()
+	rmdir()

Public Member Functions

- virtual int [faccessat](#) (const char *path, int mode, int flags) noexcept=0
Check access permissions on the given file.
- virtual int [mkdir](#) (const char *path, mode_t mode) noexcept=0
Create a new subdirectory.
- virtual int [unlink](#) (const char *path) noexcept=0
Unlink the given file from that directory.
- virtual int [rename](#) (const char *src_path, const char *dst_path) noexcept=0
Rename the given file.
- virtual int [link](#) (const char *src_path, const char *dst_path) noexcept=0
Create a hard link (second name) for the given file.
- virtual int [symlink](#) (const char *src_path, const char *dst_path) noexcept=0
Create a symbolic link for the given file.
- virtual int [rmdir](#) (const char *path) noexcept=0
Delete an empty directory.

16.339.1 Detailed Description

Interface for a POSIX file that is a directory.

This interface provides functionality for directory files in the [L4Re::Vfs](#). However, real objects always use the combined [L4Re::Vfs::File](#) interface.

Definition at line 160 of file [vfs.h](#).

16.339.2 Member Function Documentation

16.339.2.1 faccessat()

```
virtual int L4Re::Vfs::Directory::faccessat (
    const char * path,
    int mode,
    int flags) [pure virtual], [noexcept]
```

Check access permissions on the given file.

Backend function for POSIX access and faccessat functions.

Parameters

<i>path</i>	The path relative to this directory. Note: <i>path</i> is relative to this directory and may contain subdirectories.
<i>mode</i>	The access mode to check.
<i>flags</i>	The flags as in POSIX faccessat (AT_EACCESS, AT_SYMLINK_NOFOLLOW).

Returns

0 on success, or <0 on error.

Implemented in [L4Re::Vfs::Be_file](#).

16.339.2.2 link()

```
virtual int L4Re::Vfs::Directory::link (
    const char * src_path,
    const char * dst_path) [pure virtual], [noexcept]
```

Create a hard link (second name) for the given file.

Backend for the POSIX link and linkat functions.

Parameters

<i>src_path</i>	The old name of the file. Note: <i>src_path</i> is relative to this directory and may contain subdirectories.
<i>dst_path</i>	The new (second) name for the file. Note: <i>dst_path</i> is relative to this directory and may contain subdirectories.

Returns

0 on success, or <0 on error.

Implemented in [L4Re::Vfs::Be_file](#).

16.339.2.3 mkdir()

```
virtual int L4Re::Vfs::Directory::mkdir (
    const char * path,
    mode_t mode) [pure virtual], [noexcept]
```

Create a new subdirectory.

Backend for POSIX mkdir and mkdirat function calls.

Parameters

<i>path</i>	The name of the subdirectory to create. Note: <i>path</i> is relative to this directory and may contain subdirectories.
<i>mode</i>	The file mode to use for the new directory.

Returns

0 on success, or <0 on error. -ENOTDIR if this or some component in path is not a directory.

Implemented in [L4Re::Vfs::Be_file](#).

16.339.2.4 rename()

```
virtual int L4Re::Vfs::Directory::rename (  
    const char * src_path,  
    const char * dst_path) [pure virtual], [noexcept]
```

Rename the given file.

Backend for the POSIX rename, renameat functions.

Parameters

<i>src_path</i>	The old name of the file to rename. Note: <i>src_path</i> is relative to this directory and may contain subdirectories.
<i>dst_path</i>	The new name for the file. Note: <i>dst_path</i> is relative to this directory and may contain subdirectories.

Returns

0 on success, or <0 on error.

Implemented in [L4Re::Vfs::Be_file](#).

16.339.2.5 rmdir()

```
virtual int L4Re::Vfs::Directory::rmdir (  
    const char * path) [pure virtual], [noexcept]
```

Delete an empty directory.

Backend for POSIX rmdir, rmdirat functions.

Parameters

<i>path</i>	The name of the directory to remove. Note: <i>path</i> is relative to this directory and may contain subdirectories.
-------------	--

Returns

0 on success, or <0 on error.

Implemented in [L4Re::Vfs::Be_file](#).

16.339.2.6 symlink()

```
virtual int L4Re::Vfs::Directory::symlink (
    const char * src_path,
    const char * dst_path) [pure virtual], [noexcept]
```

Create a symbolic link for the given file.

Backend for the POSIX symlink and symlinkat functions.

Parameters

<i>src_path</i>	The old name of the file. Note: <i>src_path</i> shall be an absolute path.
<i>dst_path</i>	The name for symlink. Note: <i>dst_path</i> is relative to this directory and may contain subdirectories.

Returns

0 on success, or <0 on error.

Implemented in [L4Re::Vfs::Be_file](#).

16.339.2.7 unlink()

```
virtual int L4Re::Vfs::Directory::unlink (
    const char * path) [pure virtual], [noexcept]
```

Unlink the given file from that directory.

Backend for the POSIX unlink and unlinkat functions.

Parameters

<i>path</i>	The name of the file to unlink. Note: <i>path</i> is relative to this directory and may contain subdirectories.
-------------	---

Returns

0 on success, or <0 on error.

Implemented in [L4Re::Vfs::Be_file](#).

The documentation for this class was generated from the following file:

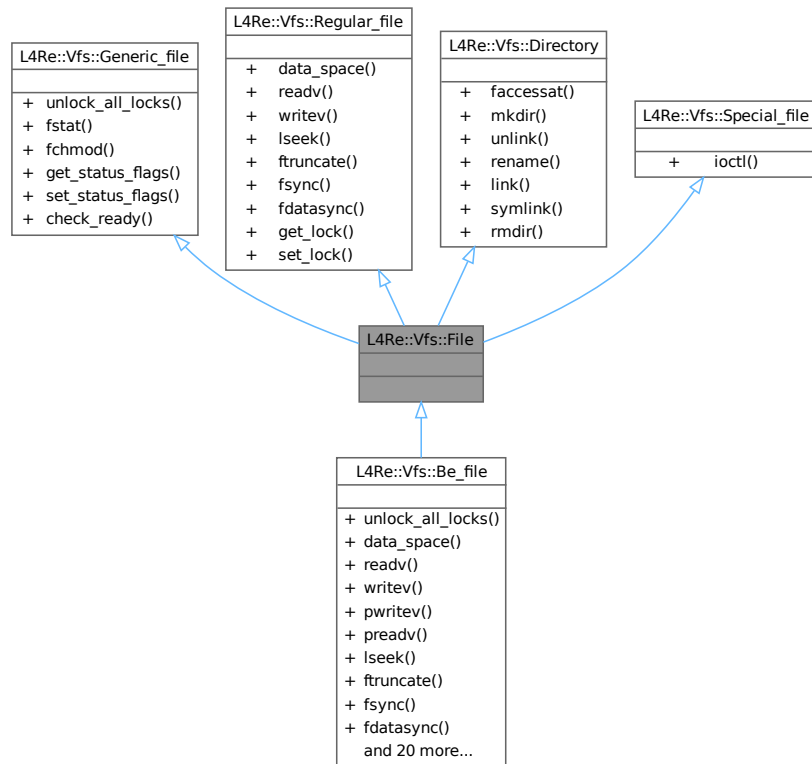
- l4/l4re_vfs/vfs.h

16.340 L4Re::Vfs::File Class Reference

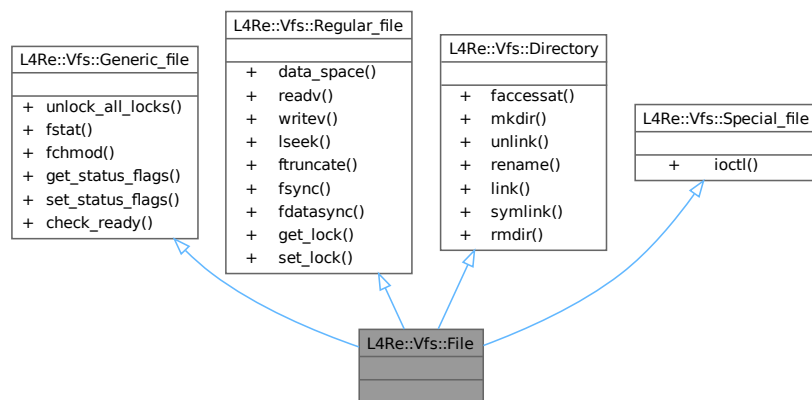
The basic interface for an open POSIX file.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::File:



Collaboration diagram for L4Re::Vfs::File:



Additional Inherited Members

Public Types inherited from [L4Re::Vfs::Generic_file](#)

- enum [Ready_type](#) : unsigned
Type of I/O operation/condition a file can indicate readiness.

Public Member Functions inherited from [L4Re::Vfs::Generic_file](#)

- virtual int [unlock_all_locks](#) () noexcept=0
Unlock all locks on the file.
- virtual int [fstat](#) (struct stat64 *buf) const noexcept=0
Get status information for the file.
- virtual int [fchmod](#) (mode_t) noexcept=0
Change POSIX access rights on the file.
- virtual int [get_status_flags](#) () const noexcept=0
Get file status flags (fcntl F_GETFL).
- virtual int [set_status_flags](#) (long flags) noexcept=0
Set file status flags (fcntl F_SETFL).
- virtual bool [check_ready](#) ([Ready_type](#) rt) noexcept=0
Check whether the file is ready for an I/O operation/condition.

Public Member Functions inherited from [L4Re::Vfs::Regular_file](#)

- virtual [L4::Cap](#)< [L4Re::Dataspace](#) > [data_space](#) () noexcept=0
Get an [L4Re::Dataspace](#) object for the file.
- virtual ssize_t [readv](#) (const struct iovec *, int iovcnt) noexcept=0
Read one or more blocks of data from the file.
- virtual ssize_t [writev](#) (const struct iovec *, int iovcnt) noexcept=0
Write one or more blocks of data to the file.
- virtual off64_t [lseek](#) (off64_t, int) noexcept=0
Change the file pointer.
- virtual int [ftruncate](#) (off64_t pos) noexcept=0
Truncate the file at the given position.
- virtual int [fsync](#) () const noexcept=0
Sync the data and meta data to persistent storage.
- virtual int [fdatsync](#) () const noexcept=0
Sync the data to persistent storage.
- virtual int [get_lock](#) (struct flock64 *lock) noexcept=0
Test if the given lock can be placed in the file.
- virtual int [set_lock](#) (struct flock64 *lock, bool wait) noexcept=0
Acquire or release the given lock on the file.

Public Member Functions inherited from L4Re::Vfs::Directory

- virtual int [faccessat](#) (const char *path, int mode, int flags) noexcept=0
Check access permissions on the given file.
- virtual int [mkdir](#) (const char *path, mode_t mode) noexcept=0
Create a new subdirectory.
- virtual int [unlink](#) (const char *path) noexcept=0
Unlink the given file from that directory.
- virtual int [rename](#) (const char *src_path, const char *dst_path) noexcept=0
Rename the given file.
- virtual int [link](#) (const char *src_path, const char *dst_path) noexcept=0
Create a hard link (second name) for the given file.
- virtual int [symlink](#) (const char *src_path, const char *dst_path) noexcept=0
Create a symbolic link for the given file.
- virtual int [rmdir](#) (const char *path) noexcept=0
Delete an empty directory.

Public Member Functions inherited from L4Re::Vfs::Special_file

- virtual int [ioctl](#) (unsigned long cmd, va_list args) noexcept=0
The famous IO control.

16.340.1 Detailed Description

The basic interface for an open POSIX file.

An open POSIX file can be anything that hides behind a POSIX file descriptor. This means that even directories are files. An open file can be anything from a directory to a special device file so see [Generic_file](#), [Regular_file](#), [Directory](#), and [Special_file](#) for more information.

Note

For implementing a backend for the [L4Re::Vfs](#) [L4Re::Vfs::Be_file](#) may be used as a base class.

Definition at line 455 of file [vfs.h](#).

The documentation for this class was generated from the following file:

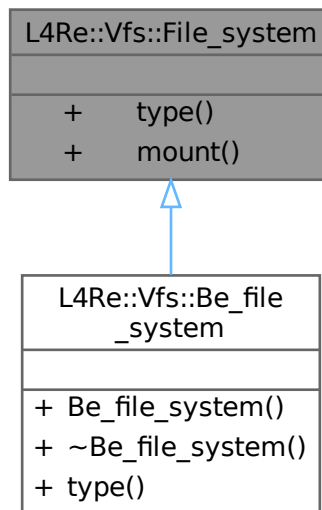
- [l4/l4re_vfs/vfs.h](#)

16.341 L4Re::Vfs::File_system Class Reference

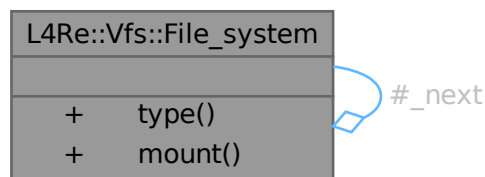
Basic interface for an [L4Re::Vfs](#) file system.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::File_system:



Collaboration diagram for L4Re::Vfs::File_system:



Public Member Functions

- virtual char const * `type` () const noexcept=0
Returns the type of the file system used in mount as fstype argument.
- virtual int `mount` (char const *source, unsigned long mountflags, void const *data, `cxx::Ref_ptr`< `File` > *dir) noexcept=0
Create a directory object dir representing source mounted with this file system.

16.341.1 Detailed Description

Basic interface for an [L4Re::Vfs](#) file system.

Note

For implementing a special file system [L4Re::Vfs::Be_file_system](#) may be used as a base class.

The main purpose of this interface is to have a single object for each supported file-system type (e.g., ext2, vfat) that exists in the application and is registered at the [L4Re::Vfs::Fs](#) singleton available via [L4Re::Vfs::vfs_ops](#). Ultimately, the POSIX mount function calls the [File_system::mount](#) method matching the file-system type given in mount.

Definition at line 857 of file [vfs.h](#).

16.341.2 Member Function Documentation

16.341.2.1 mount()

```
virtual int L4Re::Vfs::File_system::mount (
    char const * source,
    unsigned long mountflags,
    void const * data,
    cxx::Ref_ptr< File > * dir) [pure virtual], [noexcept]
```

Create a directory object *dir* representing *source* mounted with this file system.

Parameters

	<i>source</i>	The path to the source device to mount. This may also be some URL or anything file-system specific.
	<i>mountflags</i>	The mount flags as specified in the POSIX mount call.
	<i>data</i>	The data as specified in the POSIX mount call. The contents are file-system specific.
out	<i>dir</i>	A new directory object representing the file-system root directory.

Returns

0 on success, and <0 on error (e.g. -EINVAL).

References [mount\(\)](#).

Referenced by [mount\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.341.2.2 type()

```
virtual char const * L4Re::Vfs::File_system::type () const [pure virtual], [noexcept]
```

Returns the type of the file system used in mount as fstype argument.

Note

This method is already provided by [Be_file_system](#).

Implemented in [L4Re::Vfs::Be_file_system](#).

The documentation for this class was generated from the following file:

- `l4/l4re_vfs/vfs.h`

16.342 L4Re::Vfs::Fs Class Reference

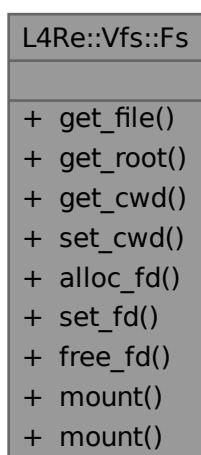
POSIX File-system related functionality.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::Fs:



Collaboration diagram for L4Re::Vfs::Fs:



Public Member Functions

- virtual `cxx::Ref_ptr< File > get_file` (int fd) noexcept=0
Get the `L4Re::Vfs::File` for the file descriptor fd.
- virtual `cxx::Ref_ptr< File > get_root` () noexcept=0
Get the directory object for the application's root directory.
- virtual `cxx::Ref_ptr< File > get_cwd` () noexcept
Get the directory object for the application's current working directory.
- virtual void `set_cwd` (cxx::Ref_ptr< File > const &) noexcept
Set the current working directory for the application.
- virtual int `alloc_fd` (cxx::Ref_ptr< File > const &f=cxx::Ref_ptr<>::Nil) noexcept=0
Allocate the next free file descriptor.
- virtual `cxx::Pair< cxx::Ref_ptr< File >, int > set_fd` (int fd, cxx::Ref_ptr< File > const &f=cxx::Ref_ptr<>::Nil) noexcept=0
Set the file object referenced by the file descriptor fd.
- virtual `cxx::Ref_ptr< File > free_fd` (int fd) noexcept=0
Free the file descriptor fd.
- virtual int `mount` (char const *path, cxx::Ref_ptr< File > const &dir) noexcept=0
Mount a given file object at the given global path in the VFS.
- int `mount` (char const *source, char const *target, char const *fstype, unsigned long mountflags, void const *data) noexcept
Backend for the POSIX mount call.

16.342.1 Detailed Description

POSIX File-system related functionality.

Note

This class usually exists as a singleton and as a superclass of `L4Re::Vfs::Ops` (

See also

`L4Re::Vfs::vfs_ops`).

Definition at line 946 of file `vfs.h`.

16.342.2 Member Function Documentation

16.342.2.1 alloc_fd()

```
virtual int L4Re::Vfs::Fs::alloc_fd (
    cxx::Ref_ptr< File > const & f = cxx::Ref_ptr<>::Nil) [pure virtual], [noexcept]
```

Allocate the next free file descriptor.

Parameters

<i>f</i>	The file to assign to that file descriptor.
----------	---

Returns

The allocated file descriptor, or -EMFILE on error.

16.342.2.2 free_fd()

```
virtual cxx::Ref_ptr< File > L4Re::Vfs::Fs::free_fd (
    int fd) [pure virtual], [noexcept]
```

Free the file descriptor *fd*.

Parameters

<i>fd</i>	The file descriptor to free.
-----------	------------------------------

Returns

A pointer to the file object that was assigned to the *fd*.

16.342.2.3 get_file()

```
virtual cxx::Ref_ptr< File > L4Re::Vfs::Fs::get_file (
    int fd) [pure virtual], [noexcept]
```

Get the [L4Re::Vfs::File](#) for the file descriptor *fd*.

Parameters

<i>fd</i>	The POSIX file descriptor number.
-----------	-----------------------------------

Returns

A pointer to the [File](#) object, or 0 if *fd* is not open.

16.342.2.4 mount()

```
virtual int L4Re::Vfs::Fs::mount (
    char const * path,
    cxx::Ref_ptr< File > const & dir) [pure virtual], [noexcept]
```

Mount a given file object at the given global path in the VFS.

Parameters

<i>path</i>	The global path to mount <i>dir</i> at.
<i>dir</i>	A pointer to the file/directory object that shall be mounted at <i>path</i> .

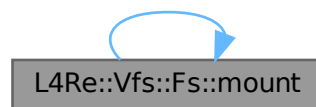
Returns

0 on success, or <0 on error.

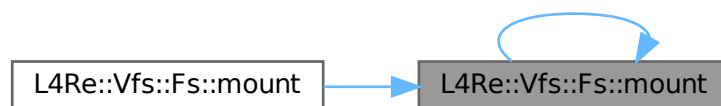
References [mount\(\)](#).

Referenced by [mount\(\)](#), and [mount\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**16.342.2.5 set_fd()**

```

virtual cxx::Pair< cxx::Ref_ptr< File >, int > L4Re::Vfs::Fs::set_fd (
    int fd,
    cxx::Ref_ptr< File > const & f = cxx::Ref_ptr<>::Nil) [pure virtual], [noexcept]
  
```

Set the file object referenced by the file descriptor *fd*.

Parameters

<i>fd</i>	The file descriptor to set to <i>f</i> .
-----------	--

<i>f</i>	The file object to assign.
----------	----------------------------

Returns

A pair of a pointer to the file object that was previously assigned to `fd` (`first`) and a return value (`second`). `second` contains `-#EBADF` if the passed file descriptor is outside the valid range. `first` contains a `Nil` pointer in that case. On success, `second` contains 0.

The documentation for this class was generated from the following files:

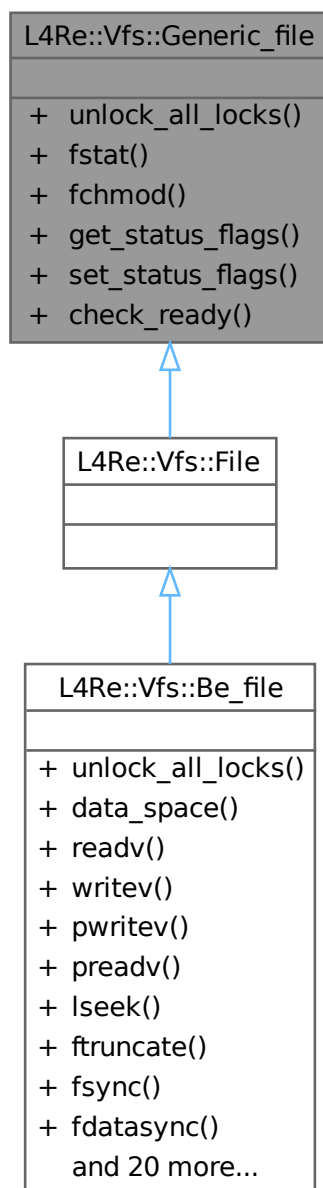
- `l4/l4re_vfs/vfs.h`
- `l4/l4re_vfs/impl/vfs_impl.h`

16.343 L4Re::Vfs::Generic_file Class Reference

The common interface for an open POSIX file.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::Generic_file:



Collaboration diagram for L4Re::Vfs::Generic_file:

L4Re::Vfs::Generic_file
<ul style="list-style-type: none"> + unlock_all_locks() + fstat() + fchmod() + get_status_flags() + set_status_flags() + check_ready()

Public Types

- enum [Ready_type](#) : unsigned
Type of I/O operation/condition a file can indicate readiness.

Public Member Functions

- virtual int [unlock_all_locks](#) () noexcept=0
Unlock all locks on the file.
- virtual int [fstat](#) (struct stat64 *buf) const noexcept=0
Get status information for the file.
- virtual int [fchmod](#) (mode_t) noexcept=0
Change POSIX access rights on the file.
- virtual int [get_status_flags](#) () const noexcept=0
Get file status flags (fcntl F_GETFL).
- virtual int [set_status_flags](#) (long flags) noexcept=0
Set file status flags (fcntl F_SETFL).
- virtual bool [check_ready](#) ([Ready_type](#) rt) noexcept=0
Check whether the file is ready for an I/O operation/condition.

16.343.1 Detailed Description

The common interface for an open POSIX file.

This interface is common to all kinds of open files, independent of the file type (e.g., directory, regular file etc.). However, in the [L4Re::Vfs](#) the interface [File](#) is used for every real object.

See also

[L4Re::Vfs::File](#) for more information.

Definition at line 52 of file [vfs.h](#).

16.343.2 Member Enumeration Documentation

16.343.2.1 Ready_type

```
enum L4Re::Vfs::Generic_file::Ready_type : unsigned
```

Type of I/O operation/condition a file can indicate readiness.

As defined by select() and similar functions.

Definition at line 60 of file [vfs.h](#).

16.343.3 Member Function Documentation

16.343.3.1 check_ready()

```
virtual bool L4Re::Vfs::Generic_file::check_ready (
    Ready_type rt) [pure virtual], [noexcept]
```

Check whether the file is ready for an I/O operation/condition.

This method is used by the implementation of select() and similar functions.

Parameters

<i>rt</i>	Type of the I/O operation/condition to be ready, as defined by the select() and similar functions (Read, Write, Exception).
-----------	---

Return values

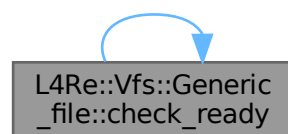
<i>true</i>	The file is ready for the given type of I/O operation/condition.
<i>false</i>	The file is not ready for the given type of I/O operation/condition.

Implemented in [L4Re::Vfs::Be_file](#).

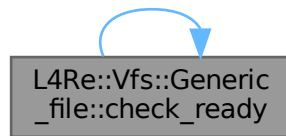
References [check_ready\(\)](#).

Referenced by [check_ready\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.343.3.2 fchmod()

```
virtual int L4Re::Vfs::Generic_file::fchmod (
    mode_t ) [pure virtual], [noexcept]
```

Change POSIX access rights on the file.

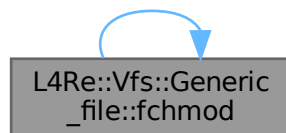
Backend for POSIX chmod and fchmod.

Implemented in [L4Re::Vfs::Be_file](#).

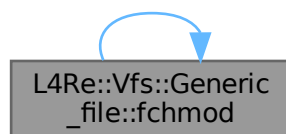
References [fchmod\(\)](#).

Referenced by [fchmod\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.343.3.3 fstat()

```
virtual int L4Re::Vfs::Generic_file::fstat (  
    struct stat64 * buf) const [pure virtual], [noexcept]
```

Get status information for the file.

This is the backend for POSIX fstat, stat, fstat64 and friends.

Parameters

out	<i>buf</i>	This buffer is filled with the status information.
-----	------------	--

Returns

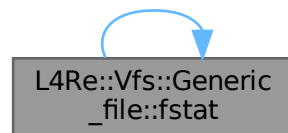
0 on success, or <0 on error.

Implemented in [L4Re::Vfs::Be_file](#).

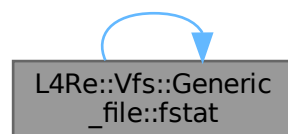
References [fstat\(\)](#).

Referenced by [fstat\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.343.3.4 get_status_flags()

```
virtual int L4Re::Vfs::Generic_file::get_status_flags () const [pure virtual], [noexcept]
```

Get file status flags (fcntl F_GETFL).

This function is used by the fcntl implementation for the F_GETFL command.

Returns

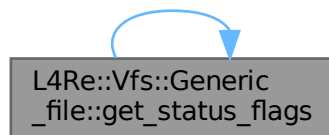
flags such as O_RDONLY, O_WRONLY, O_RDWR, O_DIRECT, O_ASYNC, O_NOATIME, O_NONBLOCK, or <0 on error.

Implemented in [L4Re::Vfs::Be_file](#).

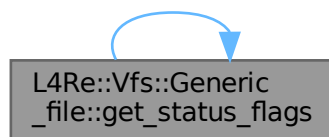
References [get_status_flags\(\)](#).

Referenced by [get_status_flags\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.343.3.5 set_status_flags()

```
virtual int L4Re::Vfs::Generic_file::set_status_flags (
    long flags) [pure virtual], [noexcept]
```

Set file status flags (fcntl F_SETFL).

This function is used by the fcntl implementation for the F_SETFL command.

Parameters

<i>flags</i>	The file status flags to set. This must be a combination of O_RDONLY, O_WRONLY, O_RDWR, O_APPEND, O_ASYNC, O_DIRECT, O_NOATIME, O_NONBLOCK.
--------------	---

Note

Creation flags such as O_CREAT, O_EXCL, O_NOCTTY, O_TRUNC are ignored.

Returns

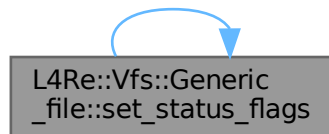
0 on success, or <0 on error.

Implemented in [L4Re::Vfs::Be_file](#).

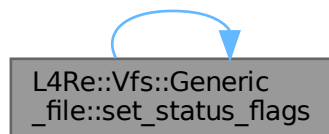
References [set_status_flags\(\)](#).

Referenced by [set_status_flags\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.343.3.6 unlock_all_locks()

```
virtual int L4Re::Vfs::Generic_file::unlock_all_locks () [pure virtual], [noexcept]
```

Unlock all locks on the file.

Note

All locks means all locks independent of which file the locks were taken by.

This method is called by the POSIX close implementation to get the POSIX semantics of releasing all locks taken by this application on a close for any fd referencing the real file.

Returns

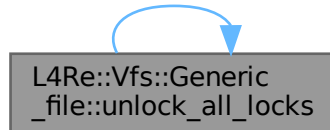
0 on success, or <0 on error.

Implemented in [L4Re::Vfs::Be_file](#).

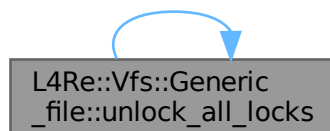
References [unlock_all_locks\(\)](#).

Referenced by [unlock_all_locks\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

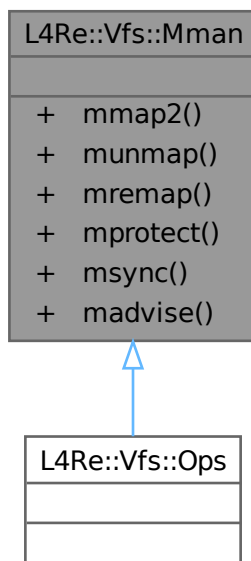
- `l4/l4re_vfs/vfs.h`

16.344 L4Re::Vfs::Mman Class Reference

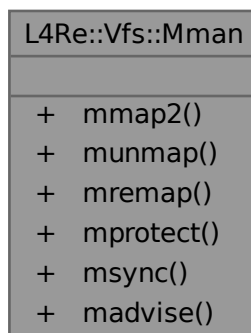
Interface for POSIX memory management.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::Mman:



Collaboration diagram for L4Re::Vfs::Mman:



Public Member Functions

- virtual int **mmap2** (void *start, size_t len, int prot, int flags, int fd, off_t offset, void **ptr) noexcept=0
Backend for the mmap2 system call.
- virtual int **munmap** (void *start, size_t len) noexcept=0
Backend for the munmap system call.
- virtual int **mremap** (void *old, size_t old_sz, size_t new_sz, int flags, void **new_addr) noexcept=0
Backend for the mremap system call.
- virtual int **mprotect** (const void *a, size_t sz, int prot) noexcept=0
Backend for the mprotect system call.
- virtual int **msync** (void *addr, size_t len, int flags) noexcept=0
Backend for the msync system call.
- virtual int **madvise** (void *addr, size_t len, int advice) noexcept=0
Backend for the madvise system call.

16.344.1 Detailed Description

Interface for POSIX memory management.

Note

This interface usually exists as a singleton and as a superclass of [L4Re::Vfs::Ops](#).

An implementation for this interface is in [l4/l4re_vfs/impl/vfs_impl.h](#) and used by the l4re_vfs library or by the VFS implementation in ldso.

Definition at line 773 of file [vfs.h](#).

The documentation for this class was generated from the following file:

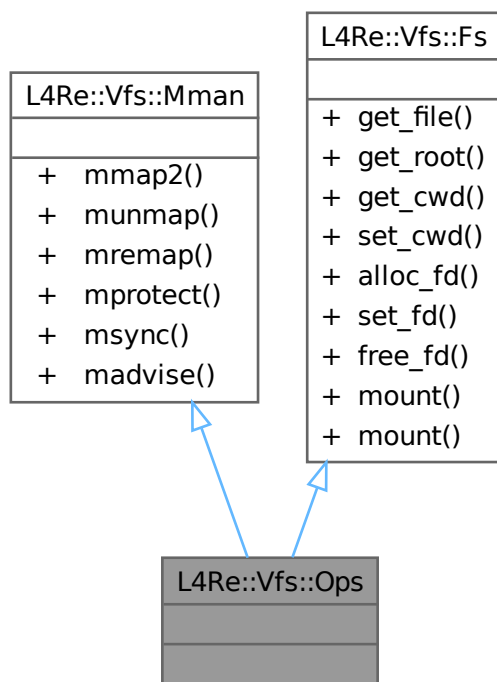
- l4/l4re_vfs/vfs.h

16.345 L4Re::Vfs::Ops Class Reference

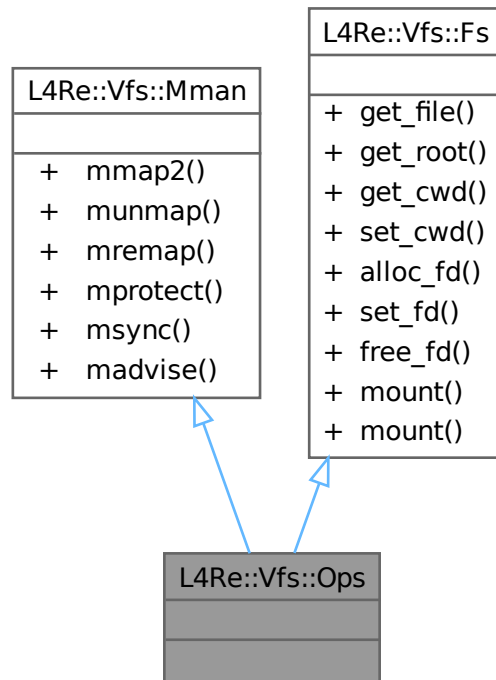
Interface for the POSIX backends of an application.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::Ops:



Collaboration diagram for L4Re::Vfs::Ops:



Additional Inherited Members

Public Member Functions inherited from L4Re::Vfs::Mman

- virtual int **mmap2** (void *start, size_t len, int prot, int flags, int fd, off_t offset, void **ptr) noexcept=0
Backend for the mmap2 system call.
- virtual int **munmap** (void *start, size_t len) noexcept=0
Backend for the munmap system call.
- virtual int **mremap** (void *old, size_t old_sz, size_t new_sz, int flags, void **new_addr) noexcept=0
Backend for the mremap system call.
- virtual int **mprotect** (const void *a, size_t sz, int prot) noexcept=0
Backend for the mprotect system call.
- virtual int **msync** (void *addr, size_t len, int flags) noexcept=0
Backend for the msync system call.
- virtual int **madvise** (void *addr, size_t len, int advice) noexcept=0
Backend for the madvice system call.

Public Member Functions inherited from L4Re::Vfs::Fs

- virtual [cxx::Ref_ptr](#)< [File](#) > [get_file](#) (int fd) noexcept=0
Get the [L4Re::Vfs::File](#) for the file descriptor fd.

- virtual `cxx::Ref_ptr< File > get_root ()` noexcept=0
Get the directory object for the application's root directory.
- virtual `cxx::Ref_ptr< File > get_cwd ()` noexcept
Get the directory object for the application's current working directory.
- virtual void `set_cwd (cxx::Ref_ptr< File > const &)` noexcept
Set the current working directory for the application.
- virtual int `alloc_fd (cxx::Ref_ptr< File > const &f=cxx::Ref_ptr<>::Nil)` noexcept=0
Allocate the next free file descriptor.
- virtual `cxx::Pair< cxx::Ref_ptr< File >, int > set_fd (int fd, cxx::Ref_ptr< File > const &f=cxx::Ref_ptr<>::Nil)` noexcept=0
Set the file object referenced by the file descriptor fd.
- virtual `cxx::Ref_ptr< File > free_fd (int fd)` noexcept=0
Free the file descriptor fd.
- virtual int `mount (char const *path, cxx::Ref_ptr< File > const &dir)` noexcept=0
Mount a given file object at the given global path in the VFS.
- int `mount (char const *source, char const *target, char const *fstype, unsigned long mountflags, void const *data)` noexcept
Backend for the POSIX mount call.

16.345.1 Detailed Description

Interface for the POSIX backends of an application.

Note

There usually exists a single instance of this interface available via `L4Re::Vfs::vfs_ops` that is used for all kinds of C-Library functions.

Definition at line 1109 of file `vfs.h`.

The documentation for this class was generated from the following file:

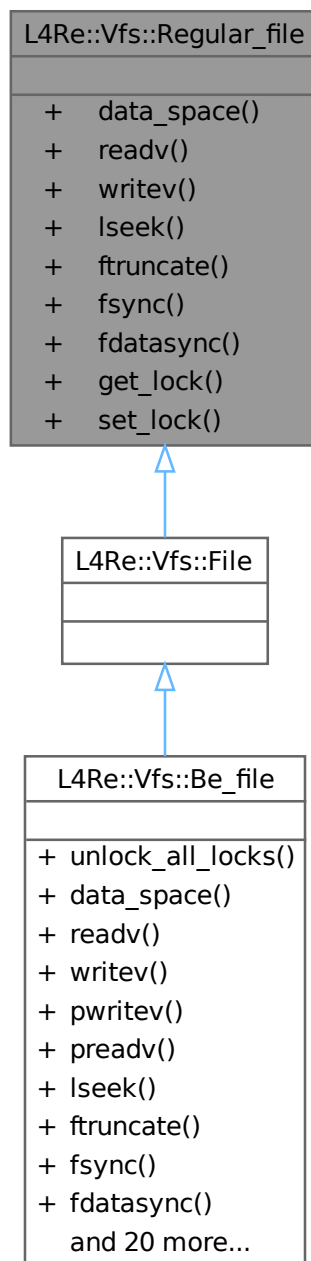
- `l4/l4re_vfs/vfs.h`

16.346 L4Re::Vfs::Regular_file Class Reference

Interface for a POSIX file that provides regular file semantics.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::Regular_file:



Collaboration diagram for L4Re::Vfs::Regular_file:

L4Re::Vfs::Regular_file
<ul style="list-style-type: none"> + data_space() + readv() + writev() + lseek() + ftruncate() + fsync() + fdatasync() + get_lock() + set_lock()

Public Member Functions

- virtual [L4::Cap](#)< [L4Re::Dataspace](#) > [data_space](#) () noexcept=0
Get an [L4Re::Dataspace](#) object for the file.
- virtual ssize_t [readv](#) (const struct iovec *, int iovcnt) noexcept=0
Read one or more blocks of data from the file.
- virtual ssize_t [writev](#) (const struct iovec *, int iovcnt) noexcept=0
Write one or more blocks of data to the file.
- virtual off64_t [lseek](#) (off64_t, int) noexcept=0
Change the file pointer.
- virtual int [ftruncate](#) (off64_t pos) noexcept=0
Truncate the file at the given position.
- virtual int [fsync](#) () const noexcept=0
Sync the data and meta data to persistent storage.
- virtual int [fdatasync](#) () const noexcept=0
Sync the data to persistent storage.
- virtual int [get_lock](#) (struct flock64 *lock) noexcept=0
Test if the given lock can be placed in the file.
- virtual int [set_lock](#) (struct flock64 *lock, bool wait) noexcept=0
Acquire or release the given lock on the file.

16.346.1 Detailed Description

Interface for a POSIX file that provides regular file semantics.

Real objects always use the combined [L4Re::Vfs::File](#) interface.

Definition at line 287 of file [vfs.h](#).

16.346.2 Member Function Documentation

16.346.2.1 data_space()

```
virtual L4::Cap< L4Re::Dataspace > L4Re::Vfs::Regular_file::data_space () [pure virtual],  
[noexcept]
```

Get an [L4Re::Dataspace](#) object for the file.

This is used as a backend for POSIX mmap and mmap2 functions.

Note

mmap is not possible if the function returns an invalid capability.

Returns

A capability to an [L4Re::Dataspace](#) that represents the file contents in an [L4Re](#) way.

Implemented in [L4Re::Vfs::Be_file](#).

16.346.2.2 fdatsync()

```
virtual int L4Re::Vfs::Regular_file::fdatsync () const [pure virtual], [noexcept]
```

Sync the data to persistent storage.

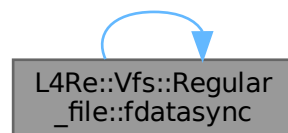
This is the backend for POSIX fdatsync.

Implemented in [L4Re::Vfs::Be_file](#).

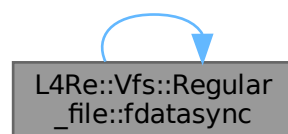
References [fdatsync\(\)](#).

Referenced by [fdatsync\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.346.2.3 fsync()

```
virtual int L4Re::Vfs::Regular_file::fsync () const [pure virtual], [noexcept]
```

Sync the data and meta data to persistent storage.

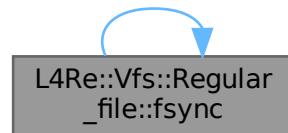
This is the backend for POSIX fsync.

Implemented in [L4Re::Vfs::Be_file](#).

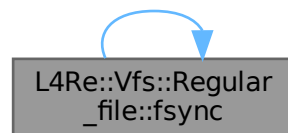
References [fsync\(\)](#).

Referenced by [fsync\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.346.2.4 ftruncate()

```
virtual int L4Re::Vfs::Regular_file::ftruncate (  
    off64_t pos) [pure virtual], [noexcept]
```

Truncate the file at the given position.

This function is the backend for truncate and friends.

Parameters

<i>pos</i>	The offset at which the file shall be truncated.
------------	--

Returns

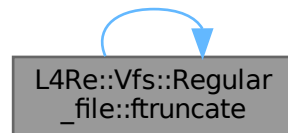
0 on success, or <0 on error.

Implemented in [L4Re::Vfs::Be_file](#).

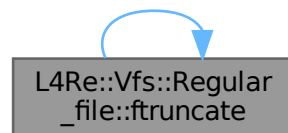
References [ftruncate\(\)](#).

Referenced by [ftruncate\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**16.346.2.5 get_lock()**

```
virtual int L4Re::Vfs::Regular_file::get_lock (
    struct flock64 * lock) [pure virtual], [noexcept]
```

Test if the given lock can be placed in the file.

This function is used as backend for fcntl F_GETLK commands.

Parameters

<i>lock</i>	The lock that shall be placed on the file. The <i>l_type</i> member will contain <code>F_UNLCK</code> if the lock could be placed.
-------------	--

Returns

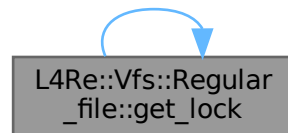
0 on success, <0 on error.

Implemented in [L4Re::Vfs::Be_file](#).

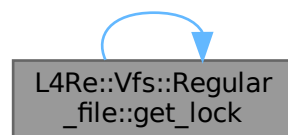
References [get_lock\(\)](#).

Referenced by [get_lock\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.346.2.6 lseek()

```
virtual off64_t L4Re::Vfs::Regular_file::lseek (
    off64_t ,
    int ) [pure virtual], [noexcept]
```

Change the file pointer.

This is the backend for POSIX seek, lseek and friends.

Returns

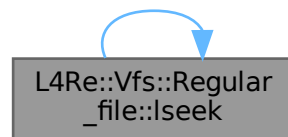
The new file position, or <0 on error.

Implemented in [L4Re::Vfs::Be_file](#).

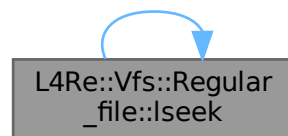
References [lseek\(\)](#).

Referenced by [lseek\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.346.2.7 readv()

```
virtual ssize_t L4Re::Vfs::Regular_file::readv (  
    const struct iovec * ,  
    int iovcnt) [pure virtual], [noexcept]
```

Read one or more blocks of data from the file.

This function acts as backend for POSIX `read` and `readv` calls and reads data starting from the `f_pos` pointer of that open file. The file pointer is advanced according to the number of bytes read.

Returns

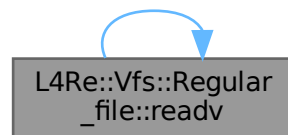
The number of bytes read from the file, or <0 on error.

Implemented in [L4Re::Vfs::Be_file](#).

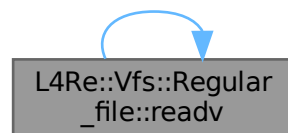
References [readv\(\)](#).

Referenced by [readv\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**16.346.2.8 set_lock()**

```
virtual int L4Re::Vfs::Regular_file::set_lock (
    struct flock64 * lock,
    bool wait) [pure virtual], [noexcept]
```

Acquire or release the given lock on the file.

This function is used as backend for fcntl F_SETLK and F_SETLKW commands.

Parameters

<i>lock</i>	The lock that shall be placed on the file.
-------------	--

<i>wait</i>	If true, then block if there is a conflicting lock on the file.
-------------	---

Returns

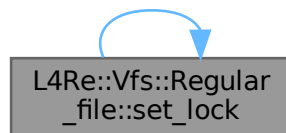
0 on success, <0 on error.

Implemented in [L4Re::Vfs::Be_file](#).

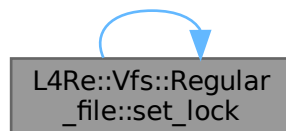
References [set_lock\(\)](#).

Referenced by [set_lock\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**16.346.2.9 writev()**

```
virtual ssize_t L4Re::Vfs::Regular_file::writev (
    const struct iovec * ,
    int iovcnt) [pure virtual], [noexcept]
```

Write one or more blocks of data to the file.

This function acts as backend for POSIX write and writev calls. The data is written starting at the current file pointer and the file pointer must be advanced according to the number of written bytes.

Returns

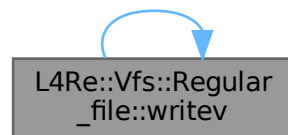
The number of bytes written to the file, or <0 on error.

Implemented in [L4Re::Vfs::Be_file](#).

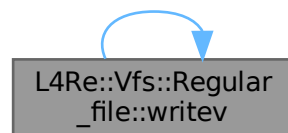
References [writev\(\)](#).

Referenced by [writev\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

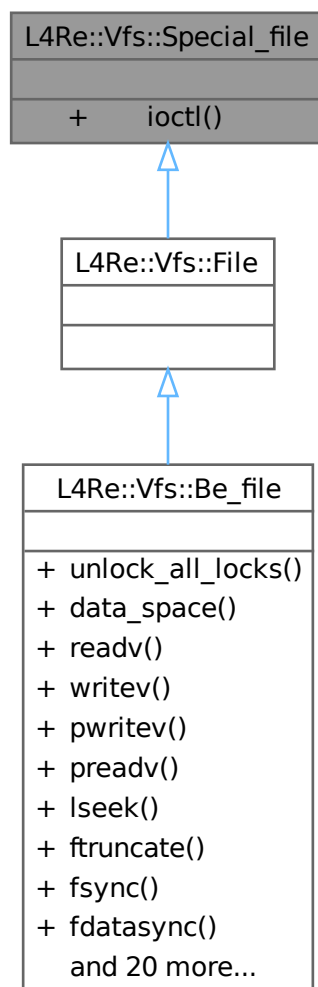
- `l4/l4re_vfs/vfs.h`

16.347 L4Re::Vfs::Special_file Class Reference

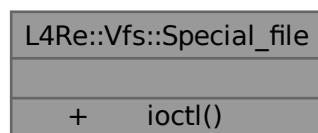
Interface for a POSIX file that provides special file semantics.

```
#include <vfs.h>
```


Inheritance diagram for L4Re::Vfs::Special_file:



Collaboration diagram for L4Re::Vfs::Special_file:



Public Member Functions

- virtual int [ioctl](#) (unsigned long cmd, va_list args) noexcept=0
The famous IO control.

16.347.1 Detailed Description

Interface for a POSIX file that provides special file semantics.

Real objects always use the combined [L4Re::Vfs::File](#) interface.

Definition at line [420](#) of file [vfs.h](#).

16.347.2 Member Function Documentation

16.347.2.1 ioctl()

```
virtual int L4Re::Vfs::Special_file::ioctl (  
    unsigned long cmd,  
    va_list args) [pure virtual], [noexcept]
```

The famous IO control.

Backend for POSIX generic object invocation ioctl.

Parameters

<i>cmd</i>	The ioctl command.
<i>args</i>	The arguments for the ioctl, usually some kind of pointer.

Returns

>=0 on success, or <0 on error.

Implemented in [L4Re::Vfs::Be_file](#).

The documentation for this class was generated from the following file:

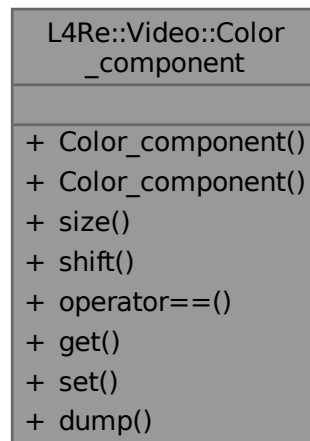
- [l4/l4re_vfs/vfs.h](#)

16.348 L4Re::Video::Color_component Class Reference

A color component.

```
#include <colors>
```

Collaboration diagram for L4Re::Video::Color_component:



Public Member Functions

- **Color_component ()**
Constructor.
- **Color_component** (unsigned char bits, unsigned char shift)
Constructor.
- unsigned char **size** () const
Return the number of bits used by the component.
- unsigned char **shift** () const
Return the position of the component in the pixel.
- bool **operator==** (**Color_component** const &o) const
Compare for equality.
- int **get** (unsigned long v) const
Get component from value (normalized to 16bits).
- long unsigned **set** (int v) const
Transform 16bit normalized value to the component in the color space.
- template<typename OUT>
void **dump** (OUT &s) const
Dump information on the view information to a stream.

16.348.1 Detailed Description

A color component.

Definition at line 21 of file [colors](#).

16.348.2 Constructor & Destructor Documentation

16.348.2.1 Color_component()

```
L4Re::Video::Color_component::Color_component (
    unsigned char bits,
    unsigned char shift) [inline]
```

Constructor.

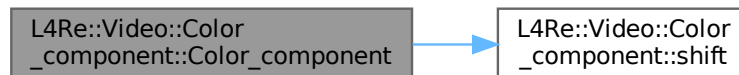
Parameters

<i>bits</i>	Number of bits used by the component
<i>shift</i>	Position in bits of the component in the pixel

Definition at line 36 of file [colors](#).

References [shift\(\)](#).

Here is the call graph for this function:



16.348.3 Member Function Documentation

16.348.3.1 dump()

```
template<typename OUT>
void L4Re::Video::Color_component::dump (
    OUT & s) const [inline]
```

Dump information on the view information to a stream.

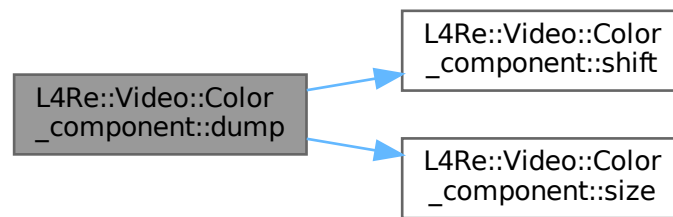
Parameters

<i>s</i>	Stream
----------	--------

Definition at line 81 of file [colors](#).

References [shift\(\)](#), and [size\(\)](#).

Here is the call graph for this function:



16.348.3.2 get()

```
int L4Re::Video::Color_component::get (  
    unsigned long v) const [inline]
```

Get component from value (normalized to 16bits).

Parameters

v	Value
---	-------

Returns

Converted value

Definition at line 63 of file [colors](#).

16.348.3.3 operator==()

```
bool L4Re::Video::Color_component::operator== (  
    Color_component const & o) const [inline]
```

Compare for equality.

Returns

True if the same components are described, false if not.

Definition at line 55 of file [colors](#).

References [Color_component\(\)](#).

Here is the call graph for this function:

**16.348.3.4 set()**

```
long unsigned L4Re::Video::Color_component::set (
    int v) const [inline]
```

Transform 16bit normalized value to the component in the color space.

Parameters

<i>v</i>	Value return Converted value.
----------	-------------------------------

Definition at line 73 of file [colors](#).

16.348.3.5 shift()

```
unsigned char L4Re::Video::Color_component::shift () const [inline]
```

Return the position of the component in the pixel.

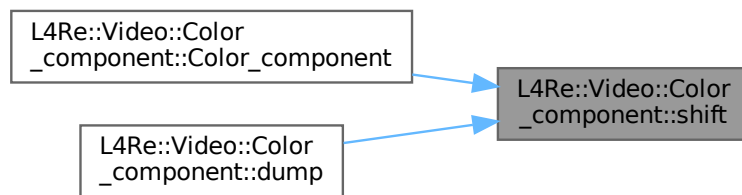
Returns

Position in bits of the component in the pixel

Definition at line 49 of file [colors](#).

Referenced by [Color_component\(\)](#), and [dump\(\)](#).

Here is the caller graph for this function:



16.348.3.6 `size()`

```
unsigned char L4Re::Video::Color_component::size () const [inline]
```

Return the number of bits used by the component.

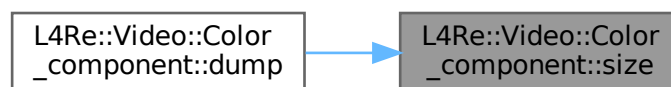
Returns

Number of bits used by the component

Definition at line 43 of file [colors](#).

Referenced by [dump\(\)](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

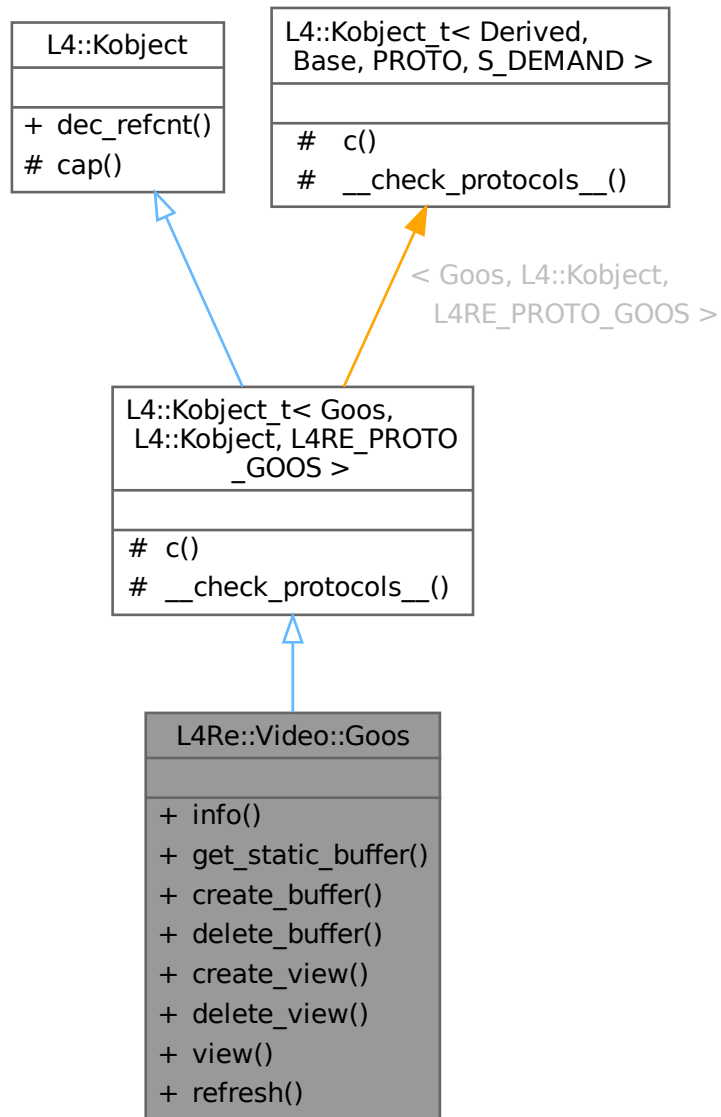
- [l4/re/video/colors](#)

16.349 L4Re::Video::Goos Class Reference

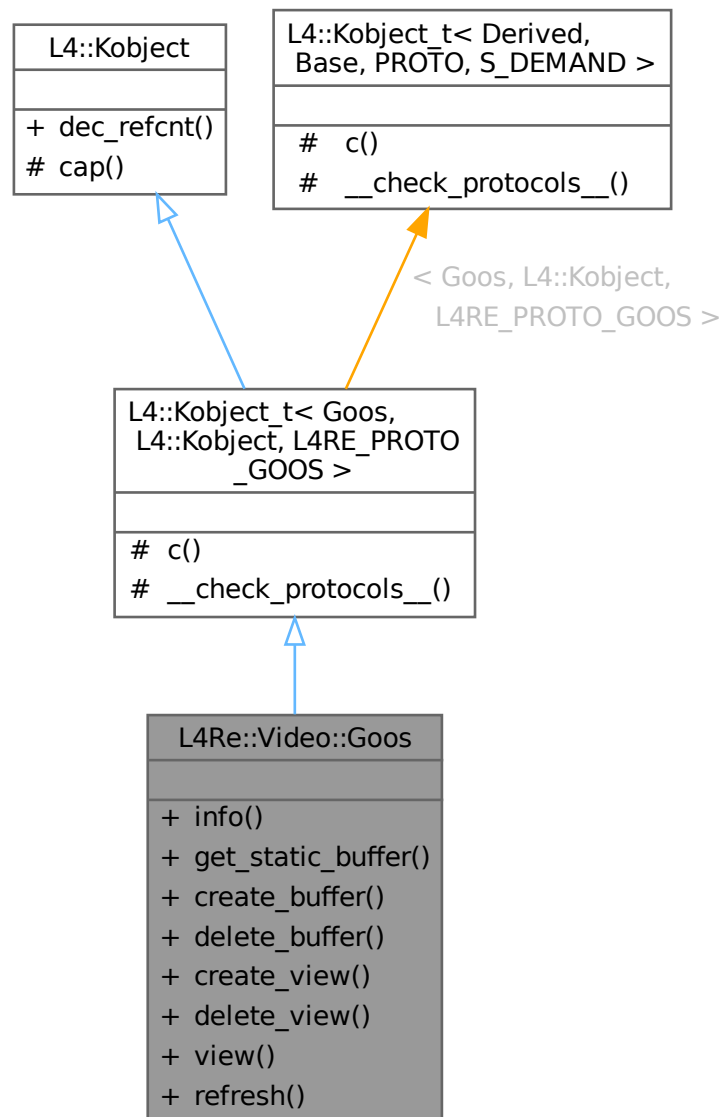
Class that abstracts framebuffers.

```
#include <goos>
```

Inheritance diagram for L4Re::Video::Goos:



Collaboration diagram for L4Re::Video::Goos:



Data Structures

- struct [Info](#)

Information structure of a [Goos](#).

Public Types

- enum [Flags](#) { [F_auto_refresh](#) = 0x01 , [F_pointer](#) = 0x02 , [F_dynamic_views](#) = 0x04 , [F_dynamic_buffers](#) = 0x08 }

[Flags](#) for a [Goos](#).

Public Member Functions

- long **info** (Info *info)
Return the Goos information of the Goos.
- long **get_static_buffer** (unsigned idx, L4::lpc::Out< L4::Cap< L4Re::Dataspace > > rbuf)
Return a static buffer of a Goos.
- long **create_buffer** (unsigned long size, L4::lpc::Out< L4::Cap< L4Re::Dataspace > > rbuf)
Create a buffer.
- long **delete_buffer** (unsigned idx)
Delete a buffer.
- int **create_view** (View *view, l4_utcb_t *utcb=l4_utcb()) const noexcept
Create a view.
- int **delete_view** (View const &v, l4_utcb_t *utcb=l4_utcb()) const noexcept
Delete a view.
- View **view** (unsigned index) const noexcept
Return a view.
- long **refresh** (int x, int y, int w, int h)
Trigger refreshing of the given area on the virtual screen.

Public Member Functions inherited from L4::Kobject

- l4_msgtag_t **dec_refcnt** (l4_mword_t diff, l4_utcb_t *utcb=l4_utcb())
Decrement the in kernel reference counter for the object.

Additional Inherited Members

Protected Types inherited from

L4::Kobject_t< Goos, L4::Kobject, L4RE_PROTO_GOOS >

- typedef Goos **Class**
The target interface type (inheriting from Kobject_t).
- typedef Typeid::Iface< PROTO, Goos > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< __Iface >, typename L4::Kobject::__Iface_list > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from

L4::Kobject_t< Goos, L4::Kobject, L4RE_PROTO_GOOS >

- L4::Cap< Class > **c** () const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from L4::Kobject

- l4_cap_idx_t **cap** () const noexcept
Return capability selector.

Static Protected Member Functions inherited from [L4::Kobject_t](#)< [Goos](#), [L4::Kobject](#), [L4RE_PROTO_GOOS](#) >

- static void `__check_protocols__()` noexcept
Helper to check for protocol conflicts.

16.349.1 Detailed Description

[Class](#) that abstracts framebuffer.

A framebuffer is the pixel data that is displayed on a screen and a [Goos](#) object lets the user manipulate that data. A [Goos](#) makes use of two kinds of objects:

- Buffers in the form of [L4Re::Dataspace](#) objects. These hold the bytes for the pixel data.
- [L4Re::Video::View](#) objects.

Both can either be static, that is their number and configuration is fixed and determined by the framebuffer, or they can be dynamic, with the user allocating them.

Definition at line [223](#) of file [goos](#).

16.349.2 Member Enumeration Documentation

16.349.2.1 Flags

enum [L4Re::Video::Goos::Flags](#)

[Flags](#) for a [Goos](#).

Enumerator

<code>F_auto_refresh</code>	The graphics display is automatically refreshed.
<code>F_pointer</code>	We have a mouse pointer.
<code>F_dynamic_views</code>	Supports dynamically allocated views.
<code>F_dynamic_buffers</code>	Supports dynamically allocated buffers.

Definition at line [228](#) of file [goos](#).

16.349.3 Member Function Documentation

16.349.3.1 `create_buffer()`

```
long L4Re::Video::Goos::create_buffer (
    unsigned long size,
    L4::Ipc::Out< L4::Cap< L4Re::Dataspace > > rbuf)
```

Create a buffer.

Parameters

<i>size</i>	Size of buffer in bytes.
<i>rbuf</i>	Capability slot to point the buffer dataspace to.

Return values

≥ 0	Success, the value returned is the buffer index.
< 0	Error

Referenced by [get_static_buffer\(\)](#).

Here is the caller graph for this function:



16.349.3.2 create_view()

```
int L4Re::Video::Goos::create_view (
    View * view,
    l4_utcb_t * utcb = l4_utcb()) const [inline], [noexcept]
```

Create a view.

Parameters

out	<i>view</i>	A view object.
	<i>utcb</i>	UTCB of the caller. This is a default parameter.

Return values

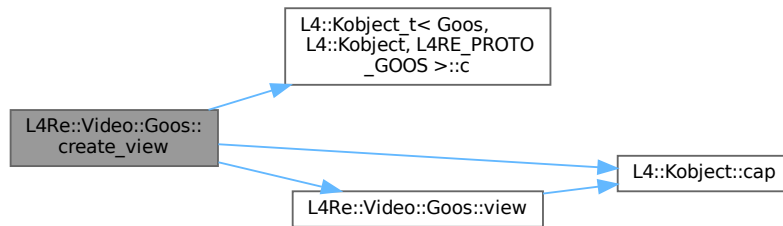
≥ 0	Success, the value returned is the view index.
< 0	Error

Definition at line 312 of file [goos](#).

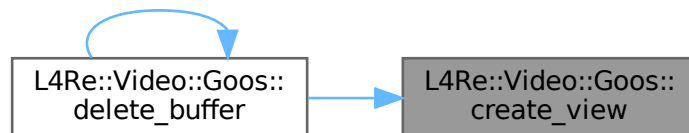
References [L4::Kobject_t< Goos](#), [L4::Kobject](#), [L4RE_PROTO_GOOS >::c\(\)](#), [L4::Kobject::cap\(\)](#), and [view\(\)](#).

Referenced by [delete_buffer\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.349.3.3 delete_buffer()

```
long L4Re::Video::Goos::delete_buffer (
    unsigned idx)
```

Delete a buffer.

Parameters

<i>idx</i>	Buffer to delete.
------------	-------------------

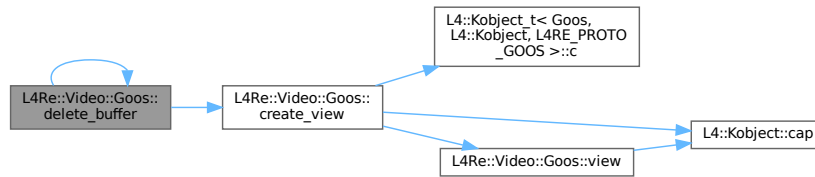
Return values

0	Success
<0	Error

References [create_view\(\)](#), and [delete_buffer\(\)](#).

Referenced by [delete_buffer\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.349.3.4 delete_view()

```

int L4Re::Video::Goos::delete_view (
    View const & v,
    l4_utcb_t * utcb = l4_utcb()) const [inline], [noexcept]

```

Delete a view.

Parameters

<i>v</i>	The view object to delete.
<i>utcb</i>	UTCB of the caller. This is a default parameter.

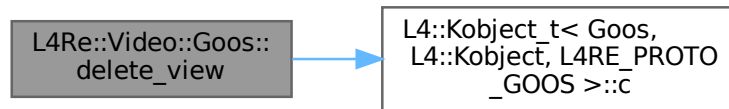
Return values

0	Success
<0	Error

Definition at line 332 of file [goos](#).

References [L4::Kobject_t< Goos, L4::Kobject, L4RE_PROTO_GOOS >::c\(\)](#).

Here is the call graph for this function:



16.349.3.5 get_static_buffer()

```
long L4Re::Video::Goos::get_static_buffer (
    unsigned idx,
    L4::Ipc::Out< L4::Cap< L4Re::Dataspace > > rbuf)
```

Return a static buffer of a [Goos](#).

Parameters

<i>idx</i>	Index of the static buffer.
<i>rbuf</i>	Capability slot to point the buffer dataspace to.

Return values

0	Success
<0	Error

References [create_buffer\(\)](#).

Referenced by [info\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.349.3.6 info()

```
long L4Re::Video::Goos::info (
    Info * info)
```

Return the [Goos](#) information of the [Goos](#).

Parameters

out	info	Goos information structure pointer.
-----	------	---

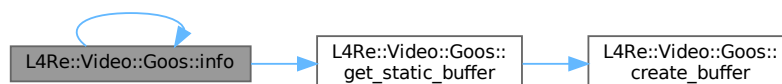
Return values

0	Success
<0	Error

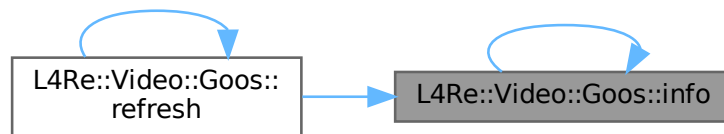
References [get_static_buffer\(\)](#), and [info\(\)](#).

Referenced by [info\(\)](#), and [refresh\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.349.3.7 view()

```
View L4Re::Video::Goos::view (
    unsigned index) const [inline], [noexcept]
```

Return a view.

Parameters

<i>index</i>	Index of the view to return.
--------------	------------------------------

Returns

The view.

Definition at line 363 of file [goos](#).

References [L4::Kobject::cap\(\)](#).

Referenced by [create_view\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

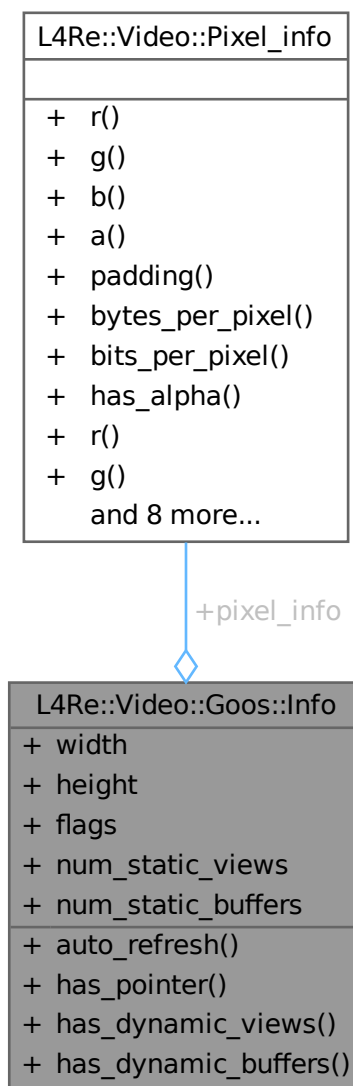
- `I4/re/video/goos`

16.350 L4Re::Video::Goos::Info Struct Reference

Information structure of a [Goos](#).

```
#include <goos>
```

Collaboration diagram for L4Re::Video::Goos::Info:



Public Member Functions

- bool **auto_refresh** () const
Return whether this [Goos](#) does auto refreshing or the view refresh functions must be used to make changes visible.
- bool **has_pointer** () const
Return whether a pointer is used by the provider of the [Goos](#).
- bool **has_dynamic_views** () const
Return whether dynamic view are supported.
- bool **has_dynamic_buffers** () const
Return whether dynamic buffers are supported.

Data Fields

- unsigned long **width**
Width.
- unsigned long **height**
Height.
- unsigned **flags**
Flags, see [Flags](#).
- unsigned **num_static_views**
Number of static view.
- unsigned **num_static_buffers**
Number of static buffers.
- [Pixel_info](#) **pixel_info**
Pixel information.

16.350.1 Detailed Description

Information structure of a [Goos](#).

Definition at line [237](#) of file [goos](#).

The documentation for this struct was generated from the following file:

- [l4/re/video/goos](#)

16.351 L4Re::Video::Pixel_info Class Reference

Pixel information.

```
#include <colors>
```

Collaboration diagram for L4Re::Video::Pixel_info:

L4Re::Video::Pixel_info
<ul style="list-style-type: none"> + r() + g() + b() + a() + padding() + bytes_per_pixel() + bits_per_pixel() + has_alpha() + r() + g() and 8 more...

Public Member Functions

- [Color_component](#) const & [r](#) () const
Return the red color compoment of the pixel.
- [Color_component](#) const & [g](#) () const
Return the green color compoment of the pixel.
- [Color_component](#) const & [b](#) () const
Return the blue color compoment of the pixel.
- [Color_component](#) const & [a](#) () const
Return the alpha color compoment of the pixel.
- [Color_component](#) const [padding](#) () const
Compute the padding pseudo component.
- unsigned char [bytes_per_pixel](#) () const
Query size of pixel in bytes.
- unsigned char [bits_per_pixel](#) () const
Number of bits of the pixel.
- bool [has_alpha](#) () const
Return whether the pixel has an alpha channel.
- void [r](#) ([Color_component](#) const &c)
Set the red color component of the pixel.
- void [g](#) ([Color_component](#) const &c)
Set the green color component of the pixel.
- void [b](#) ([Color_component](#) const &c)
Set the blue color component of the pixel.
- void [a](#) ([Color_component](#) const &c)
Set the alpha color component of the pixel.
- void [bytes_per_pixel](#) (unsigned char bpp)
Set the size of the pixel in bytes.
- **Pixel_info** ()
Constructor.
- [Pixel_info](#) (unsigned char bpp, char [r](#), char [rs](#), char [g](#), char [gs](#), char [b](#), char [bs](#), char [a](#)=0, char [as](#)=0)
Constructor.
- template<typename VBI>
[Pixel_info](#) (VBI const *vbi)
Convenience constructor.
- bool [operator==](#) ([Pixel_info](#) const &o) const
Compare for complete equality of the color space.
- template<typename OUT>
void [dump](#) (OUT &s) const
Dump information on the pixel to a stream.

16.351.1 Detailed Description

Pixel information.

This class wraps the information on a pixel, such as the size and position of each color component in the pixel.

Definition at line 94 of file [colors](#).

16.351.2 Constructor & Destructor Documentation

16.351.2.1 Pixel_info() [1/2]

```
L4Re::Video::Pixel_info::Pixel_info (  
    unsigned char bpp,  
    char r,  
    char rs,  
    char g,  
    char gs,  
    char b,  
    char bs,  
    char a = 0,  
    char as = 0) [inline]
```

Constructor.

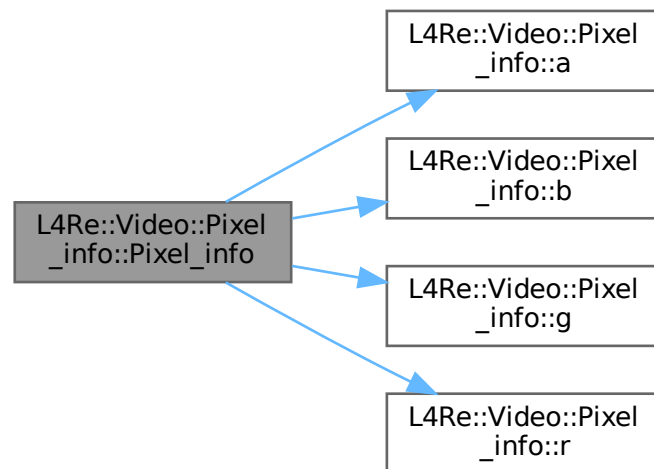
Parameters

<i>bpp</i>	Size of pixel in bytes.
<i>r</i>	Red component size.
<i>rs</i>	Red component shift.
<i>g</i>	Green component size.
<i>gs</i>	Green component shift.
<i>b</i>	Blue component size.
<i>bs</i>	Blue component shift.
<i>a</i>	Alpha component size, defaults to 0.
<i>as</i>	Alpha component shift, defaults to 0.

Definition at line 212 of file [colors](#).

References [a\(\)](#), [b\(\)](#), [g\(\)](#), and [r\(\)](#).

Here is the call graph for this function:



16.351.2.2 Pixel_info() [2/2]

```
template<typename VBI>
L4Re::Video::Pixel_info::Pixel_info (
    VBI const * vbi) [inline], [explicit]
```

Convenience constructor.

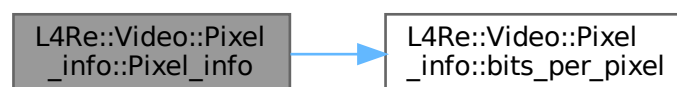
Parameters

<i>vbi</i>	Suitable information structure. Convenience constructor to create the pixel info from a VESA Framebuffer Info.
------------	--

Definition at line 224 of file [colors](#).

References [bits_per_pixel\(\)](#).

Here is the call graph for this function:



16.351.3 Member Function Documentation

16.351.3.1 `a()` [1/2]

```
Color_component const & L4Re::Video::Pixel_info::a () const [inline]
```

Return the alpha color compoment of the pixel.

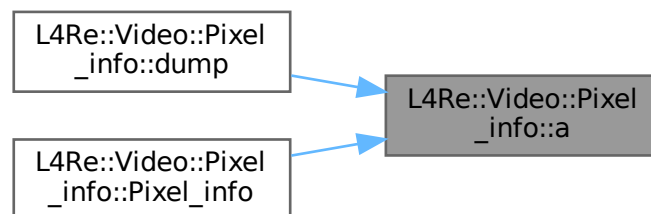
Returns

Alpha color component.

Definition at line 123 of file `colors`.

Referenced by `dump()`, and `Pixel_info()`.

Here is the caller graph for this function:



16.351.3.2 `a()` [2/2]

```
void L4Re::Video::Pixel_info::a (
    Color_component const & c) [inline]
```

Set the alpha color component of the pixel.

Parameters

<code>c</code>	Alpha color component.
----------------	------------------------

Definition at line 187 of file `colors`.

16.351.3.3 b() [1/2]

```
Color_component const & L4Re::Video::Pixel_info::b () const [inline]
```

Return the blue color compoment of the pixel.

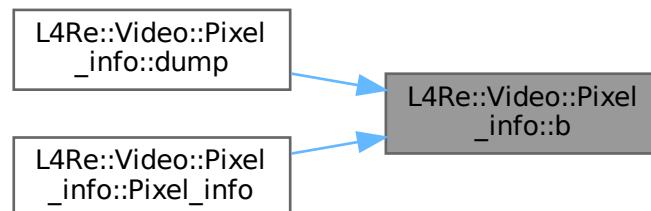
Returns

Blue color component.

Definition at line 117 of file [colors](#).

Referenced by [dump\(\)](#), and [Pixel_info\(\)](#).

Here is the caller graph for this function:



16.351.3.4 b() [2/2]

```
void L4Re::Video::Pixel_info::b (  
    Color_component const & c) [inline]
```

Set the blue color component of the pixel.

Parameters

<code>c</code>	Blue color component.
----------------	-----------------------

Definition at line 181 of file [colors](#).

16.351.3.5 bits_per_pixel()

```
unsigned char L4Re::Video::Pixel_info::bits_per_pixel () const [inline]
```

Number of bits of the pixel.

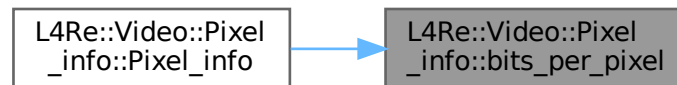
Returns

Number of bits used by the pixel.

Definition at line 156 of file [colors](#).

Referenced by [Pixel_info\(\)](#).

Here is the caller graph for this function:



16.351.3.6 bytes_per_pixel() [1/2]

```
unsigned char L4Re::Video::Pixel_info::bytes_per_pixel () const [inline]
```

Query size of pixel in bytes.

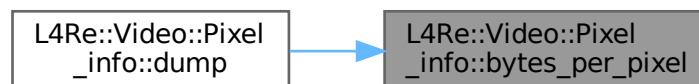
Returns

Size of pixel in bytes.

Definition at line 150 of file [colors](#).

Referenced by [dump\(\)](#).

Here is the caller graph for this function:



16.351.3.7 bytes_per_pixel() [2/2]

```
void L4Re::Video::Pixel_info::bytes_per_pixel (  
    unsigned char bpp) [inline]
```

Set the size of the pixel in bytes.

Parameters

<i>bpp</i>	Size of pixel in bytes.
------------	-------------------------

Definition at line 193 of file [colors](#).

16.351.3.8 dump()

```
template<typename OUT>
void L4Re::Video::Pixel_info::dump (
    OUT & s) const [inline]
```

Dump information on the pixel to a stream.

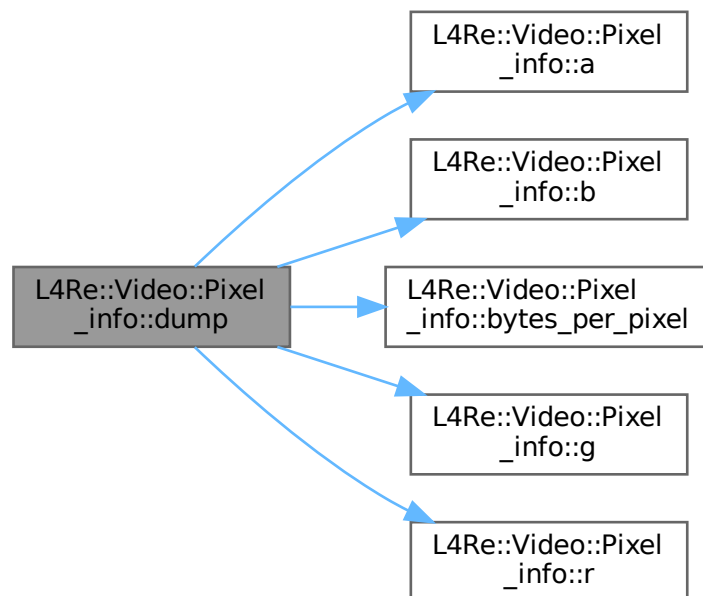
Parameters

<i>s</i>	Stream
----------	--------

Definition at line 246 of file [colors](#).

References [a\(\)](#), [b\(\)](#), [bytes_per_pixel\(\)](#), [g\(\)](#), and [r\(\)](#).

Here is the call graph for this function:



16.351.3.9 g() [1/2]

```
Color_component const & L4Re::Video::Pixel_info::g () const [inline]
```

Return the green color component of the pixel.

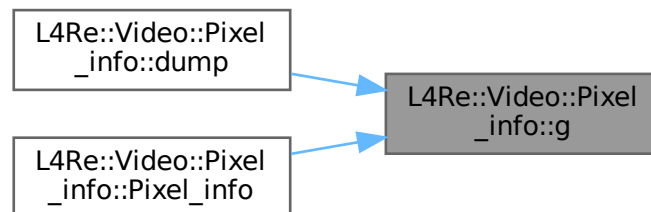
Returns

Green color component.

Definition at line 111 of file [colors](#).

Referenced by [dump\(\)](#), and [Pixel_info\(\)](#).

Here is the caller graph for this function:

**16.351.3.10 g() [2/2]**

```
void L4Re::Video::Pixel_info::g (
    Color_component const & c) [inline]
```

Set the green color component of the pixel.

Parameters

<code>c</code>	Green color component.
----------------	------------------------

Definition at line 175 of file [colors](#).

16.351.3.11 has_alpha()

```
bool L4Re::Video::Pixel_info::has_alpha () const [inline]
```

Return whether the pixel has an alpha channel.

Returns

True if the pixel has an alpha channel, false if not.

Definition at line 163 of file [colors](#).

16.351.3.12 operator==()

```
bool L4Re::Video::Pixel_info::operator== (
    Pixel_info const & o) const [inline]
```

Compare for complete equality of the color space.

Parameters

<i>o</i>	A Pixel_info to compare to.
----------	---

Returns

true if the both [Pixel_info](#)'s are equal, false if not.

Definition at line 236 of file [colors](#).

References [Pixel_info\(\)](#).

Here is the call graph for this function:



16.351.3.13 padding()

```
Color_component const L4Re::Video::Pixel_info::padding () const [inline]
```

Compute the padding pseudo component.

The padding pseudo component represents the tailing bits that are reserved in RGB32 and similar pixel formats.

Returns

Padding pseudo component.

Definition at line 131 of file [colors](#).

16.351.3.14 `r()` [1/2]

```
Color_component const & L4Re::Video::Pixel_info::r () const [inline]
```

Return the red color component of the pixel.

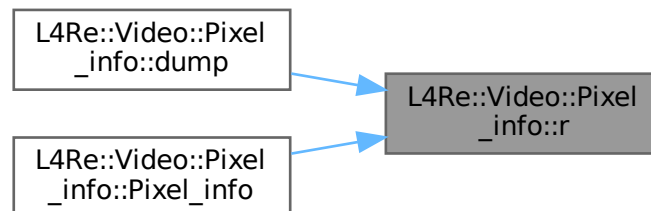
Returns

Red color component.

Definition at line 105 of file [colors](#).

Referenced by [dump\(\)](#), and [Pixel_info\(\)](#).

Here is the caller graph for this function:

**16.351.3.15** `r()` [2/2]

```
void L4Re::Video::Pixel_info::r (
    Color_component const & c) [inline]
```

Set the red color component of the pixel.

Parameters

<code>c</code>	Red color component.
----------------	----------------------

Definition at line 169 of file [colors](#).

The documentation for this class was generated from the following file:

- `l4/re/video/colors`

16.352 L4Re::Video::View Class Reference

[View](#) of a framebuffer.

```
#include <goos>
```

Collaboration diagram for L4Re::Video::View:

L4Re::Video::View
<ul style="list-style-type: none"> + info() + set_info() + set_viewport() + stack() + push_top() + push_bottom() + refresh() + valid()

Data Structures

- struct [Info](#)
Information structure of a view.

Public Types

- enum [Flags](#) {
[F_none](#) = 0x00 , [F_set_buffer](#) = 0x01 , [F_set_buffer_offset](#) = 0x02 , [F_set_bytes_per_line](#) = 0x04 ,
[F_set_pixel](#) = 0x08 , [F_set_position](#) = 0x10 , [F_dyn_allocated](#) = 0x20 , [F_set_background](#) = 0x40 ,
[F_set_flags](#) = 0x80 , [F_fully_dynamic](#) }
Flags on a view.
- enum [V_flags](#) { [F_above](#) = 0x1000 , [F_flags_mask](#) = 0xff000 }
Property flags of a view.

Public Member Functions

- int [info](#) ([Info](#) *info) const noexcept
Return the view information of the view.
- int [set_info](#) ([Info](#) const &info) const noexcept
Set the information structure for this view.
- int [set_viewport](#) (int scr_x, int scr_y, int w, int h, unsigned long buf_offset) const noexcept
Set the position of the view in the [Goos](#).

- int [stack](#) ([View](#) const &pivot, bool behind=true) const noexcept
Move this view in the view stack.
- int [push_top](#) () const noexcept
Make this view the top-most view.
- int [push_bottom](#) () const noexcept
Push this view the back.
- int [refresh](#) (int x, int y, int w, int h) const noexcept
Refresh/Redraw the view.
- bool [valid](#) () const
Return whether this view is valid.

16.352.1 Detailed Description

[View](#) of a framebuffer.

A view is a rectangular subset of a framebuffer managed by a [Goos](#) object. The [Goos](#) orders multiple views in a stack which determines which view is on top in case they overlap. The view's pixel data is provided by a backing buffer, which must belong to the [Goos](#). It can be static or dynamically allocated, depending on the framebuffer.

See also

[L4Re::Video::Goos](#)

Definition at line 39 of file [goos](#).

16.352.2 Member Enumeration Documentation

16.352.2.1 Flags

enum [L4Re::Video::View::Flags](#)

[Flags](#) on a view.

Enumerator

F_none	everything for this view is static (the VESA-FB case)
F_set_buffer	buffer object for this view can be changed
F_set_buffer_offset	buffer offset can be set
F_set_bytes_per_line	bytes per line can be set
F_set_pixel	pixel type can be set
F_set_position	position on screen can be set
F_dyn_allocated	View is dynamically allocated.
F_set_background	Set view as background for session.

<code>F_set_flags</code>	Set view flags (. See also V_flags)
<code>F_fully_dynamic</code>	Flags for a fully dynamic view.

Definition at line 59 of file [goos](#).

16.352.2.2 V_flags

```
enum L4Re::Video::View::V_flags
```

Property flags of a view.

Such flags can be set or deleted with the [F_set_flags](#) operation using the [set_info\(\)](#) method.

Enumerator

<code>F_above</code>	Flag the view as stay on top.
<code>F_flags_mask</code>	Mask containing all possible property flags.

Definition at line 82 of file [goos](#).

16.352.3 Member Function Documentation

16.352.3.1 info()

```
int L4Re::Video::View::info (
    Info * info) const [inline], [noexcept]
```

Return the view information of the view.

Parameters

out	<i>info</i>	Information structure pointer.
-----	-------------	--------------------------------

Return values

0	Success
<0	Error

Definition at line 367 of file [goos](#).

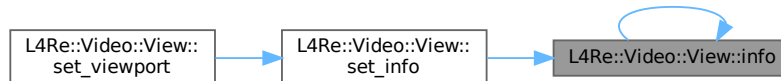
References [info\(\)](#).

Referenced by [info\(\)](#), and [set_info\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.352.3.2 refresh()

```

int L4Re::Video::View::refresh (
    int x,
    int y,
    int w,
    int h) const [inline], [noexcept]
  
```

Refresh/Redraw the view.

Parameters

<i>x</i>	X position.
<i>y</i>	Y position.
<i>w</i>	Width.
<i>h</i>	Height.

Return values

0	Success
<0	Error

Definition at line [379](#) of file [goos](#).

16.352.3.3 set_info()

```
int L4Re::Video::View::set_info (  
    Info const & info) const [inline], [noexcept]
```

Set the information structure for this view.

Parameters

<i>info</i>	Information structure.
-------------	------------------------

Return values

0	Success
<0	Error

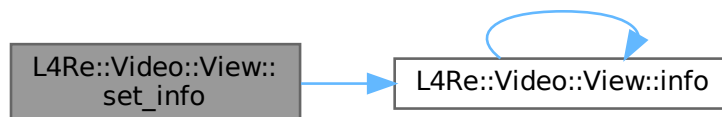
The function will also set the view port according to the values given in the information structure.

Definition at line 371 of file [goos](#).

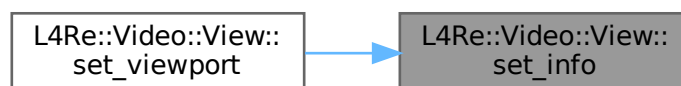
References [info\(\)](#).

Referenced by [set_viewport\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.352.3.4 set_viewport()

```
int L4Re::Video::View::set_viewport (
    int scr_x,
    int scr_y,
    int w,
    int h,
    unsigned long buf_offset) const [inline], [noexcept]
```

Set the position of the view in the [Goos](#).

Parameters

<i>scr_x</i>	X position
<i>scr_y</i>	Y position
<i>w</i>	Width
<i>h</i>	Height
<i>buf_offset</i>	Offset in the buffer in bytes

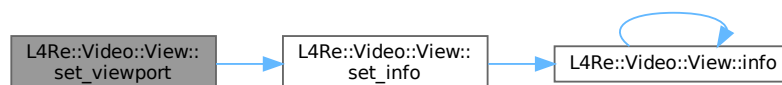
Return values

0	Success
<0	Error

Definition at line 383 of file [goos](#).

References [L4Re::Video::View::Info::buffer_index](#), [L4Re::Video::View::Info::buffer_offset](#), [L4Re::Video::View::Info::bytes_per_line](#), [F_set_buffer_offset](#), [F_set_position](#), [L4Re::Video::View::Info::flags](#), [L4Re::Video::View::Info::height](#), [L4Re::Video::View::Info::pixel_info](#), [set_info\(\)](#), [L4Re::Video::View::Info::view_index](#), [L4Re::Video::View::Info::width](#), [L4Re::Video::View::Info::xpos](#), and [L4Re::Video::View::Info::ypos](#).

Here is the call graph for this function:



16.352.3.5 stack()

```
int L4Re::Video::View::stack (
    View const & pivot,
    bool behind = true) const [inline], [noexcept]
```

Move this view in the view stack.

Parameters

<i>pivot</i>	View to move relative to
<i>behind</i>	When true move the view behind the pivot view, if false move the view before the pivot view.

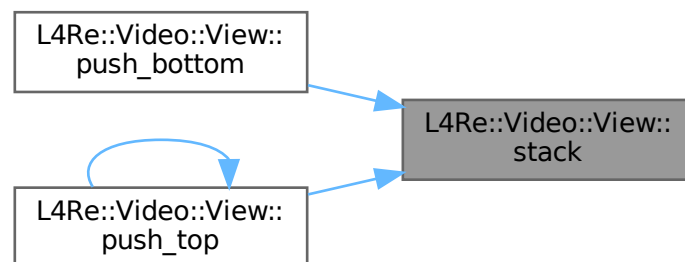
Return values

0	Success
<0	Error

Definition at line 375 of file [goos](#).

Referenced by [push_bottom\(\)](#), and [push_top\(\)](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

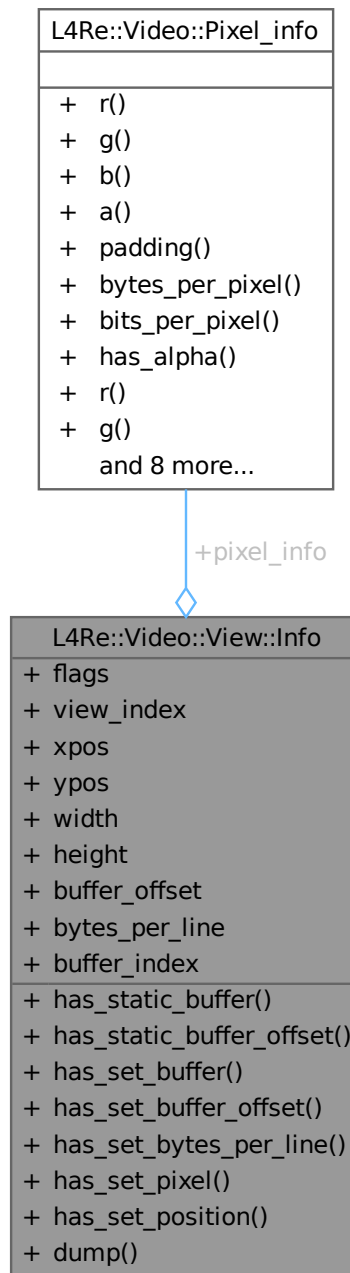
- `l4/re/video/goos`

16.353 L4Re::Video::View::Info Struct Reference

Information structure of a view.

```
#include <goos>
```

Collaboration diagram for L4Re::Video::View::Info:



Public Member Functions

- bool **has_static_buffer** () const
Return whether the view has a static buffer.
- bool **has_static_buffer_offset** () const
Return whether the static buffer offset is available.
- bool **has_set_buffer** () const

Return whether a buffer is set.

- bool **has_set_buffer_offset** () const

Return whether the given buffer offset is valid.

- bool **has_set_bytes_per_line** () const

Return whether the given bytes-per-line value is valid.

- bool **has_set_pixel** () const

Return whether the given pixel information is valid.

- bool **has_set_position** () const

Return whether the position information given is valid.

- template<typename OUT>

void **dump** (OUT &s) const

Dump information on the view information to a stream.

Data Fields

- unsigned **flags** = 0

Flags, see [Flags](#) and [V_flags](#).

- unsigned **view_index** = 0

Index of the view.

- unsigned long **xpos** = 0

X position in pixels of the view in the [Goos](#).

- unsigned long **ypos** = 0

Y position in pixels of the view in the [Goos](#).

- unsigned long **width** = 0

Width of the view in pixels.

- unsigned long **height** = 0

Height of the view in pixels.

- unsigned long **buffer_offset** = 0

Offset in the memory buffer in bytes.

- unsigned long **bytes_per_line** = 0

Bytes per line.

- [Pixel_info](#) **pixel_info**

Pixel information.

- unsigned **buffer_index** = 0

Number of the buffer used for this view.

16.353.1 Detailed Description

Information structure of a view.

Definition at line 91 of file [goos](#).

The documentation for this struct was generated from the following file:

- l4/re/video/goos

16.354 l4re_aux_t Struct Reference

Auxiliary descriptor.

```
#include <l4aux.h>
```

Collaboration diagram for l4re_aux_t:

l4re_aux_t
+ binary
+ kip_ds
+ dbg_lvl
+ ldr_flags
+ ldr_base

Data Fields

- char const * **binary**
Binary name.
- [l4_cap_idx_t](#) **kip_ds**
Data space of the KIP.
- [l4_umword_t](#) **dbg_lvl**
Debug levels for l4re.
- [l4_umword_t](#) **ldr_flags**
Flags for l4re, see [l4re_aux_ldr_flags_t](#).
- [l4_addr_t](#) **ldr_base**
Load offset of executable.

16.354.1 Detailed Description

Auxiliary descriptor.

Definition at line 40 of file [l4aux.h](#).

The documentation for this struct was generated from the following file:

- [l4/re/l4aux.h](#)

16.355 l4re_ds_stats_t Struct Reference

Information about the data space.

```
#include <dataspace.h>
```

Collaboration diagram for l4re_ds_stats_t:

l4re_ds_stats_t	
+	size
+	flags

Data Fields

- l4re_ds_size_t **size**
size
- l4re_ds_flags_t **flags**
flags

16.355.1 Detailed Description

Information about the data space.

Definition at line 39 of file [dataspace.h](#).

The documentation for this struct was generated from the following file:

- [l4re/c/dataspace.h](#)

16.356 l4re_elf_aux_mword_t Struct Reference

Auxiliary vector element for a single unsigned data word.

```
#include <elf_aux.h>
```

Collaboration diagram for l4re_elf_aux_mword_t:

l4re_elf_aux_mword_t	

16.356.1 Detailed Description

Auxiliary vector element for a single unsigned data word.

Definition at line 118 of file [elf_aux.h](#).

The documentation for this struct was generated from the following file:

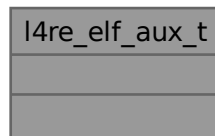
- [l4re/elf_aux.h](#)

16.357 l4re_elf_aux_t Struct Reference

Generic header for each auxiliary vector element.

```
#include <elf_aux.h>
```

Collaboration diagram for l4re_elf_aux_t:



16.357.1 Detailed Description

Generic header for each auxiliary vector element.

Definition at line 98 of file [elf_aux.h](#).

The documentation for this struct was generated from the following file:

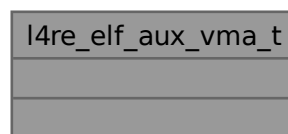
- [l4re/elf_aux.h](#)

16.358 l4re_elf_aux_vma_t Struct Reference

Auxiliary vector element for a reserved virtual memory area.

```
#include <elf_aux.h>
```

Collaboration diagram for l4re_elf_aux_vma_t:



16.358.1 Detailed Description

Auxiliary vector element for a reserved virtual memory area.

Definition at line 107 of file [elf_aux.h](#).

The documentation for this struct was generated from the following file:

- [l4re/elf_aux.h](#)

16.359 l4re_env_cap_entry_t Struct Reference

Entry in the [L4Re](#) environment array for the named initial objects.

```
#include <env.h>
```

Collaboration diagram for l4re_env_cap_entry_t:

l4re_env_cap_entry_t
+ cap
+ flags
+ name
+ l4re_env_cap_entry_t()
+ l4re_env_cap_entry_t()

Public Member Functions

- [l4re_env_cap_entry_t\(\)](#) [L4_NOTHROW](#)
Create an invalid entry.
- [l4re_env_cap_entry_t](#) (char const *n, [l4_cap_idx_t](#) c, [l4_umword_t](#) f=0) [L4_NOTHROW](#)
Create an entry with the name n, capability c, and flags f.

Data Fields

- [l4_cap_idx_t](#) cap
The capability selector for the object.
- [l4_umword_t](#) flags
Flags for the object.
- char **name** [16]
The name of the object.

16.359.1 Detailed Description

Entry in the [L4Re](#) environment array for the named initial objects.

Definition at line [39](#) of file [env.h](#).

16.359.2 Constructor & Destructor Documentation

16.359.2.1 `l4re_env_cap_entry_t()`

```
l4re_env_cap_entry_t::l4re_env_cap_entry_t (
    char const * n,
    l4_cap_idx_t c,
    l4_umword_t f = 0) [inline]
```

Create an entry with the name *n*, capability *c*, and flags *f*.

Parameters

<i>n</i>	is the name of the initial object.
<i>c</i>	is the capability selector that refers the initial object.
<i>f</i>	are the additional flags for the object.

Definition at line [70](#) of file [env.h](#).

References [cap](#), [flags](#), [L4_NOTHROW](#), and [name](#).

16.359.3 Field Documentation

16.359.3.1 `flags`

```
l4_umword_t l4re_env_cap_entry_t::flags
```

Flags for the object.

Note

Currently unused, except as an end marker for the entry list.

Definition at line [50](#) of file [env.h](#).

Referenced by [l4re_env_cap_entry_t\(\)](#), [l4re_env_cap_entry_t\(\)](#), and [l4re_env_get_cap_l\(\)](#).

The documentation for this struct was generated from the following file:

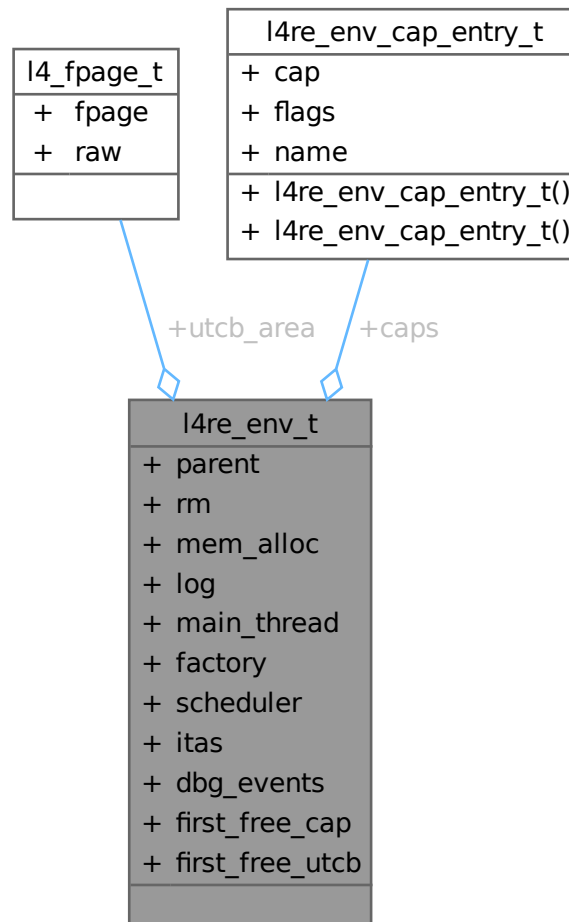
- [l4re/env.h](#)

16.360 l4re_env_t Struct Reference

Initial environment data structure.

```
#include <env.h>
```

Collaboration diagram for l4re_env_t:



Data Fields

- [l4_cap_idx_t](#) **parent**
Parent object-capability.
- [l4_cap_idx_t](#) **rm**
Region map object-capability.
- [l4_cap_idx_t](#) **mem_alloc**
Memory allocator object-capability.
- [l4_cap_idx_t](#) **log**
Logging object-capability.

- [l4_cap_idx_t](#) **main_thread**
Object-capability of the first user thread.
- [l4_cap_idx_t](#) **factory**
Object-capability of the factory available to the task.
- [l4_cap_idx_t](#) **scheduler**
Object capability for the scheduler set to use.
- [l4_cap_idx_t](#) **itas**
ITAS services object-capability.
- [l4_cap_idx_t](#) **dbg_events**
Object-capability of the debug events service.
- [l4_cap_idx_t](#) **first_free_cap**
First capability index available to the application.
- [l4_fpage_t](#) **utcb_area**
UTCB area of the task.
- [l4_addr_t](#) **first_free_utcb**
First UTCB within the UTCB area available to the application.
- [l4re_env_cap_entry_t](#) * **caps**
Pointer to the first entry in the initial objects array which contains [l4re_env_cap_entry_t](#) elements.

16.360.1 Detailed Description

Initial environment data structure.

See also

[Initial environment](#)

Definition at line 98 of file [env.h](#).

16.360.2 Field Documentation

16.360.2.1 caps

```
l4re\_env\_cap\_entry\_t* l4re\_env\_t::caps
```

Pointer to the first entry in the initial objects array which contains [l4re_env_cap_entry_t](#) elements.

The array is terminated by an invalid entry with a `flags` value of `~0ul`.

Definition at line 117 of file [env.h](#).

The documentation for this struct was generated from the following file:

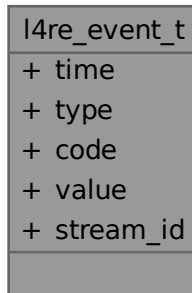
- [l4/re/env.h](#)

16.361 l4re_event_t Struct Reference

Event structure used in buffer.

```
#include <event.h>
```

Collaboration diagram for l4re_event_t:



Data Fields

- long long **time**
Time stamp of the event.
- unsigned short **type**
Type of the event.
- unsigned short **code**
Code of the event.
- int **value**
Value of the event.
- [l4_umword_t](#) **stream_id**
Stream ID.

16.361.1 Detailed Description

Event structure used in buffer.

Definition at line 30 of file [event.h](#).

The documentation for this struct was generated from the following file:

- [l4/re/c/event.h](#)

16.362 l4re_video_color_component_t Struct Reference

Color component structure.

```
#include <colors.h>
```

Collaboration diagram for l4re_video_color_component_t:

l4re_video_color_component_t	
+	size
+	shift

Data Fields

- unsigned char **size**
Size in bits.
- unsigned char **shift**
offset in pixel

16.362.1 Detailed Description

Color component structure.

Definition at line 20 of file [colors.h](#).

The documentation for this struct was generated from the following file:

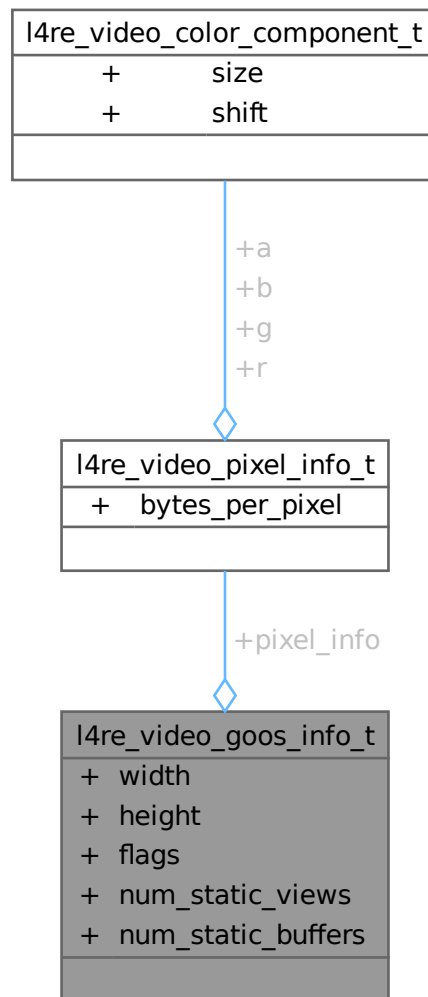
- [l4re/c/video/colors.h](#)

16.363 l4re_video_goos_info_t Struct Reference

Goos information structure.

```
#include <goos.h>
```


Collaboration diagram for l4re_video_goos_info_t:



Data Fields

- unsigned long **width**
Width of the goos.
- unsigned long **height**
Height of the goos.
- unsigned **flags**
Flags of the framebuffer, see [l4re_video_goos_info_flags_t](#).
- unsigned **num_static_views**
Number of static views.
- unsigned **num_static_buffers**
Number of static buffers.
- [l4re_video_pixel_info_t](#) **pixel_info**
Pixel layout of the goos.

16.363.1 Detailed Description

Goos information structure.

Definition at line 41 of file [goos.h](#).

The documentation for this struct was generated from the following file:

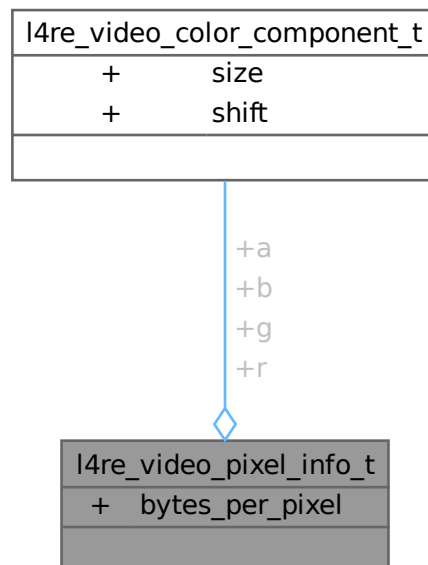
- [l4re/c/video/goos.h](#)

16.364 l4re_video_pixel_info_t Struct Reference

Pixel_info structure.

```
#include <colors.h>
```

Collaboration diagram for l4re_video_pixel_info_t:



Data Fields

- [l4re_video_color_component_t a](#)
Colors.
- unsigned char **bytes_per_pixel**
Bytes per pixel.

16.364.1 Detailed Description

Pixel_info structure.

Definition at line 30 of file [colors.h](#).

The documentation for this struct was generated from the following file:

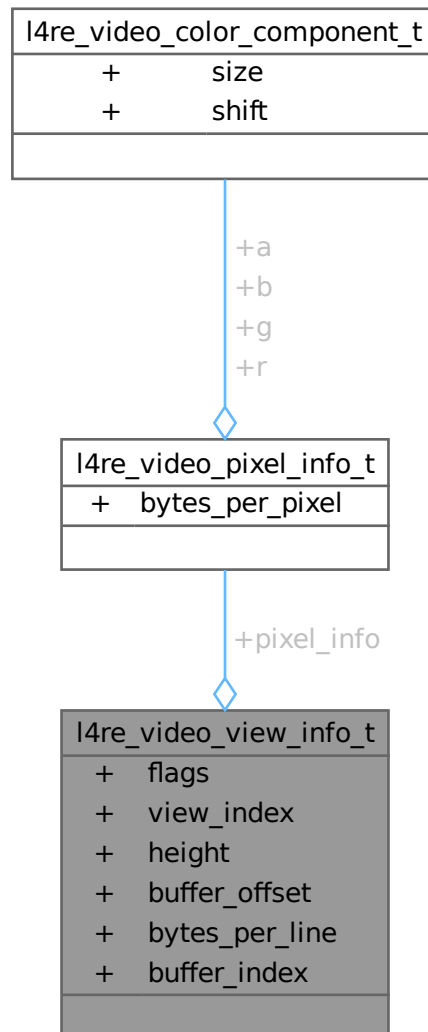
- [l4re/c/video/colors.h](#)

16.365 l4re_video_view_info_t Struct Reference

View information structure.

```
#include <view.h>
```

Collaboration diagram for l4re_video_view_info_t:



Data Fields

- unsigned **flags**
Flags.
- unsigned **view_index**
Number of view in the goos.
- unsigned long **height**
Position in goos and size of view.
- unsigned long **buffer_offset**
Memory offset in goos buffer.
- unsigned long **bytes_per_line**
Size of line in view.
- [l4re_video_pixel_info_t](#) **pixel_info**
Pixel info.
- unsigned **buffer_index**
Number of buffer of goos.

16.365.1 Detailed Description

View information structure.

Definition at line 49 of file [view.h](#).

The documentation for this struct was generated from the following file:

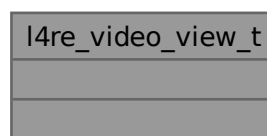
- [l4re/c/video/view.h](#)

16.366 l4re_video_view_t Struct Reference

C representation of a goos view.

```
#include <view.h>
```

Collaboration diagram for `l4re_video_view_t`:



16.366.1 Detailed Description

C representation of a goos view.

A view is a visible rectangle that provides a view to the contents of a buffer (frame buffer) memory object and is placed on a real screen.

Definition at line 68 of file [view.h](#).

The documentation for this struct was generated from the following file:

- [l4/re/c/video/view.h](#)

16.367 l4shmc_ringbuf_head_t Struct Reference

Head field of a ring buffer.

```
#include <ringbuf.h>
```

Collaboration diagram for l4shmc_ringbuf_head_t:

l4shmc_ringbuf_head_t	
+	next_read
+	next_write
+	bytes_filled
+	sender_waits
+	data

Data Fields

- unsigned **next_read**
offset to next read packet
- unsigned **next_write**
offset to next write packet
- unsigned **bytes_filled**
bytes filled in buffer
- unsigned **sender_waits**
sender waiting?
- char **data** []
tail pointer -> data

16.367.1 Detailed Description

Head field of a ring buffer.

Definition at line 59 of file [ringbuf.h](#).

The documentation for this struct was generated from the following file:

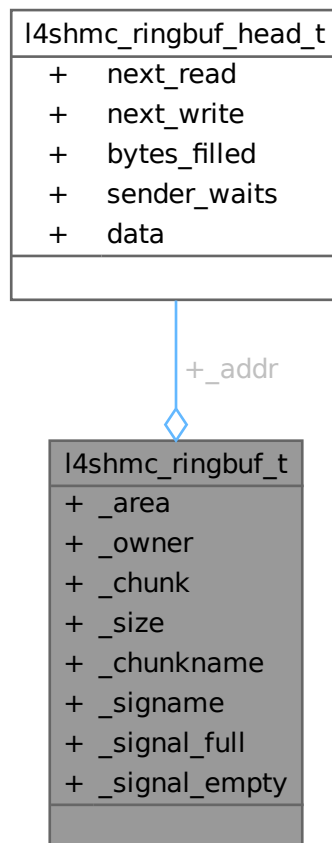
- [l4shmc/ringbuf.h](#)

16.368 l4shmc_ringbuf_t Struct Reference

Ring buffer.

```
#include <ringbuf.h>
```

Collaboration diagram for l4shmc_ringbuf_t:



Data Fields

- `l4shmc_area_t * _area`
L4SHM area this buffer is located in.
- `l4_cap_idx_t _owner`
owner (attached to send/recv signal)
- `l4shmc_chunk_t _chunk`
chunk descriptor
- `unsigned _size`
chunk size // XXX do we need this?
- `char * _chunkname`
name of the ring buffer chunk
- `char * _signame`
base name of the ring buffer signals
- `l4shmc_ringbuf_head_t * _addr`
pointer to ring buffer head
- `l4shmc_signal_t _signal_full`
"full" signal - triggered when data is produced
- `l4shmc_signal_t _signal_empty`
"empty" signal - triggered when data is consumed

16.368.1 Detailed Description

Ring buffer.

Definition at line 85 of file [ringbuf.h](#).

The documentation for this struct was generated from the following file:

- [l4/shmc/ringbuf.h](#)

16.369 l4util_l4mod_info Struct Reference

Base module structure.

```
#include <l4mod.h>
```

Collaboration diagram for `l4util_l4mod_info`:

<code>l4util_l4mod_info</code>
+ flags
+ cmdline
+ mods_addr
+ mods_count
+ vbe_ctrl_info
+ vbe_mode_info

Data Fields

- [l4_uint64_t](#) **flags**
Flags.
- [l4_uint64_t](#) **cmdline**
Pointer to kernel command line.
- [l4_uint64_t](#) **mods_addr**
Module list.
- [l4_uint32_t](#) **mods_count**
Number of modules.
- [l4_uint64_t](#) **vbe_ctrl_info**
VESA video info, valid if one of vbe_ctrl_info or vbe_mode_info is not zero.
- [l4_uint64_t](#) **vbe_mode_info**
VESA video mode info.

16.369.1 Detailed Description

Base module structure.

Definition at line 36 of file [l4mod.h](#).

16.369.2 Field Documentation**16.369.2.1 vbe_ctrl_info**

```
l4\_uint64\_t l4util_l4mod_info::vbe_ctrl_info
```

VESA video info, valid if one of vbe_ctrl_info or vbe_mode_info is not zero.

VESA video controller info

Definition at line 48 of file [l4mod.h](#).

The documentation for this struct was generated from the following file:

- [l4/util/l4mod.h](#)

16.370 l4util_l4mod_mod Struct Reference

A single module.

```
#include <l4mod.h>
```

Collaboration diagram for l4util_l4mod_mod:

l4util_l4mod_mod
+ flags
+ mod_start
+ mod_end
+ cmdline

Data Fields

- [l4_uint64_t](#) **flags**
Module flags (l4util_l4mod_mod_info_flag).
- [l4_uint64_t](#) **mod_start**
Starting address of module in memory.
- [l4_uint64_t](#) **mod_end**
End address of module in memory.
- [l4_uint64_t](#) **cmdline**
Module command line.

16.370.1 Detailed Description

A single module.

Definition at line 27 of file [l4mod.h](#).

The documentation for this struct was generated from the following file:

- [l4/util/l4mod.h](#)

16.371 l4util_mb_addr_range_t Struct Reference

INT-15, AX=E820 style "AddressRangeDescriptor" ...with a "size" parameter on the front which is the structure size - 4, pointing to the next one, up until the full buffer length of the memory map has been reached.

```
#include <mb_info.h>
```

Collaboration diagram for l4util_mb_addr_range_t:

l4util_mb_addr_range_t	
+	struct_size
+	addr
+	size
+	type

Data Fields

- [l4_uint32_t](#) **struct_size**
Size of structure.
- [l4_uint64_t](#) **addr**
Start address.
- [l4_uint64_t](#) **size**
Size of memory range.
- [l4_uint32_t](#) **type**
type of memory range

16.371.1 Detailed Description

INT-15, AX=E820 style "AddressRangeDescriptor" ...with a "size" parameter on the front which is the structure size - 4, pointing to the next one, up until the full buffer length of the memory map has been reached.

Definition at line 48 of file [mb_info.h](#).

The documentation for this struct was generated from the following file:

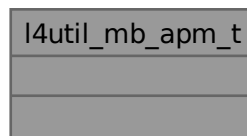
- [l4/util/mb_info.h](#)

16.372 l4util_mb_apm_t Struct Reference

APM BIOS info.

```
#include <mb_info.h>
```

Collaboration diagram for l4util_mb_apm_t:



16.372.1 Detailed Description

APM BIOS info.

Definition at line 90 of file [mb_info.h](#).

The documentation for this struct was generated from the following file:

- [l4/util/mb_info.h](#)

16.373 l4util_mb_drive_t Struct Reference

Drive Info structure.

```
#include <mb_info.h>
```

Collaboration diagram for l4util_mb_drive_t:

l4util_mb_drive_t
+ drive_number
+ drive_mode
+ drive_cylinders
+ drive_heads
+ drive_sectors
+ drive_ports

Data Fields

- [l4_uint8_t](#) **drive_number**
<The size of this structure.
- [l4_uint8_t](#) **drive_mode**
<The BIOS drive number.
- [l4_uint16_t](#) **drive_cylinders**
<The access mode (see below).
- [l4_uint8_t](#) **drive_heads**
<number of cylinders
- [l4_uint8_t](#) **drive_sectors**
<number of heads
- [l4_uint16_t](#) **drive_ports** [0]
<number of sectors per track

16.373.1 Detailed Description

Drive Info structure.

Definition at line 73 of file [mb_info.h](#).

The documentation for this struct was generated from the following file:

- [l4/util/mb_info.h](#)

16.374 l4util_mb_info_t Struct Reference

MultiBoot Info description.

```
#include <mb_info.h>
```

Collaboration diagram for l4util_mb_info_t:

l4util_mb_info_t
+ flags
+ mem_lower
+ mem_upper
+ boot_device
+ cmdline
+ mods_count
+ mods_addr
+ tabsize
+ num
+ mmap_length
and 12 more...

Data Fields

- [l4_uint32_t](#) **flags**
MultiBoot info version number.
- [l4_uint32_t](#) **mem_lower**
available memory below 1MB
- [l4_uint32_t](#) **mem_upper**
available memory starting from 1MB [KB]
- [l4_uint32_t](#) **boot_device**
"root" partition
- [l4_uint32_t](#) **cmdline**
Kernel command line.
- [l4_uint32_t](#) **mods_count**
number of modules
- [l4_uint32_t](#) **mods_addr**
module list
- [l4_uint32_t](#) **mmap_length**
size of memory mapping buffer
- [l4_uint32_t](#) **mmap_addr**
address of memory mapping buffer

- [l4_uint32_t drives_length](#)
size of drive info buffer
- [l4_uint32_t drives_addr](#)
address of driver info buffer
- [l4_uint32_t config_table](#)
ROM configuration table.
- [l4_uint32_t boot_loader_name](#)
Boot Loader Name.
- [l4_uint32_t apm_table](#)
APM table.
- [l4_uint32_t vbe_ctrl_info](#)
VESA video controller info.
- [l4_uint32_t vbe_mode_info](#)
VESA video mode info.
- [l4_uint16_t vbe_mode](#)
VESA video mode number.
- [l4_uint16_t vbe_interface_seg](#)
VESA segment of prot BIOS interface.
- [l4_uint16_t vbe_interface_off](#)
VESA offset of prot BIOS interface.
- [l4_uint16_t vbe_interface_len](#)
VESA lenght of prot BIOS interface.

16.374.1 Detailed Description

MultiBoot Info description.

This is the struct passed to the boot image. This is done by placing its address in the EAX register.

Definition at line 247 of file [mb_info.h](#).

The documentation for this struct was generated from the following file:

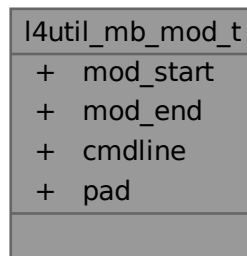
- [l4/util/mb_info.h](#)

16.375 l4util_mb_mod_t Struct Reference

The structure type "mod_list" is used by the [multiboot_info](#) structure.

```
#include <mb_info.h>
```

Collaboration diagram for `l4util_mb_mod_t`:



Data Fields

- [l4_uint32_t](#) **mod_start**
Starting address of module in memory.
- [l4_uint32_t](#) **mod_end**
End address of module in memory.
- [l4_uint32_t](#) **cmdline**
Module command line.
- [l4_uint32_t](#) **pad**
padding to take it to 16 bytes

16.375.1 Detailed Description

The structure type "mod_list" is used by the [multiboot_info](#) structure.

Definition at line 33 of file [mb_info.h](#).

The documentation for this struct was generated from the following file:

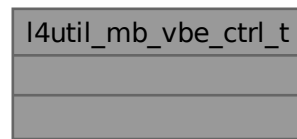
- [l4/util/mb_info.h](#)

16.376 l4util_mb_vbe_ctrl_t Struct Reference

VBE controller information.

```
#include <mb_info.h>
```

Collaboration diagram for l4util_mb_vbe_ctrl_t:



16.376.1 Detailed Description

VBE controller information.

Definition at line [106](#) of file [mb_info.h](#).

The documentation for this struct was generated from the following file:

- [l4/util/mb_info.h](#)

16.377 l4util_mb_vbe_mode_t Struct Reference

VBE mode information.

```
#include <mb_info.h>
```

Collaboration diagram for `l4util_mb_vbe_mode_t`:

<code>l4util_mb_vbe_mode_t</code>
+ mode_attributes
+ win_a_attributes
+ win_b_attributes
+ win_granularity
+ win_size
+ win_a_segment
+ win_b_segment
+ win_func
+ bytes_per_scanline
+ x_resolution
+ y_resolution
+ x_char_size
+ y_char_size
+ number_of_planes
+ bits_per_pixel
+ number_of_banks
+ memory_model
+ bank_size
+ number_of_image_pages
+ reserved0
+ red_mask_size
+ red_field_position
+ green_mask_size
+ green_field_position
+ blue_mask_size
+ blue_field_position
+ reserved_mask_size
+ reserved_field_position
+ direct_color_mode_info
+ phys_base
+ reserved1
+ reversed2
+ linear_bytes_per_scanline
+ banked_number_of_image_pages
+ linear_number_of_image_pages
+ linear_red_mask_size
+ linear_red_field_position
+ linear_green_mask_size
+ linear_green_field_position
+ linear_blue_mask_size
+ linear_blue_field_position
+ linear_reserved_mask_size
+ linear_reserved_field_position
+ max_pixel_clock
+ reserved3
* mode_attributes
* win_a_attributes
* win_b_attributes
* win_granularity
* win_size
* win_a_segment
* win_b_segment
* win_func
* bytes_per_scanline
* x_resolution
* y_resolution
* x_char_size
* y_char_size
* number_of_planes
* bits_per_pixel
* number_of_banks
* memory_model
* bank_size
* number_of_image_pages
* reserved0
* red_mask_size
* red_field_position
* green_mask_size
* green_field_position
* blue_mask_size
* blue_field_position
* reserved_mask_size
* reserved_field_position
* direct_color_mode_info
* phys_base
* reserved1
* reversed2
* linear_bytes_per_scanline
* banked_number_of_image_pages
* linear_number_of_image_pages
* linear_red_mask_size
* linear_red_field_position
* linear_green_mask_size
* linear_green_field_position
* linear_blue_mask_size
* linear_blue_field_position
* linear_reserved_mask_size
* linear_reserved_field_position
* max_pixel_clock
* reserved3

Data Fields

all VESA versions

- [l4_uint16_t mode_attributes](#)
Mode attributes.
- [l4_uint8_t win_a_attributes](#)
Window A attributes.

- [l4_uint8_t win_b_attributes](#)
Window B attributes.
- [l4_uint16_t win_granularity](#)
Window granularity.
- [l4_uint16_t win_size](#)
Window size.
- [l4_uint16_t win_a_segment](#)
Window A start segment.
- [l4_uint16_t win_b_segment](#)
Window B start segment.
- [l4_uint32_t win_func](#)
Real mode pointer to window function.
- [l4_uint16_t bytes_per_scanline](#)
Bytes per scan line.

>= VESA version 1.2

- [l4_uint16_t x_resolution](#)
Horizontal resolution in pixels or characters.
- [l4_uint16_t y_resolution](#)
Vertical resolution in pixels or characters.
- [l4_uint8_t x_char_size](#)
Character cell width in pixels.
- [l4_uint8_t y_char_size](#)
Character cell height in pixels.
- [l4_uint8_t number_of_planes](#)
Number of memory planes.
- [l4_uint8_t bits_per_pixel](#)
Bits per pixel.
- [l4_uint8_t number_of_banks](#)
Number of banks.
- [l4_uint8_t memory_model](#)
Memory model type.
- [l4_uint8_t bank_size](#)
Bank size in KiB.
- [l4_uint8_t number_of_image_pages](#)
Number of images.
- [l4_uint8_t reserved0](#)
Reserved for page function.

direct color

- [l4_uint8_t red_mask_size](#)
Size of direct color red mask in bits.
- [l4_uint8_t red_field_position](#)
Bit position of LSB of red mask.
- [l4_uint8_t green_mask_size](#)
Size of direct color green mask in bits.
- [l4_uint8_t green_field_position](#)
Bit position of LSB of green mask.
- [l4_uint8_t blue_mask_size](#)
Size of direct color blue mask in bits.
- [l4_uint8_t blue_field_position](#)
Bit position of LSB of blue mask.
- [l4_uint8_t reserved_mask_size](#)
Size of direct color reserved mask in bits.
- [l4_uint8_t reserved_field_position](#)
Bit position of LSB of reserved mask.

- [l4_uint8_t direct_color_mode_info](#)

Direct color mode attributes.

>= VESA version 2.0

- [l4_uint32_t phys_base](#)
Physical address for flat memory memory frame buffer.
- [l4_uint32_t reserved1](#)
Reserved – always set to 0.
- [l4_uint16_t reversed2](#)
Reserved – always set to 0.

>= VESA version 3.0

- [l4_uint16_t linear_bytes_per_scanline](#)
Bytes per scan line for linear modes.
- [l4_uint8_t banked_number_of_image_pages](#)
Number of images for banked modes.
- [l4_uint8_t linear_number_of_image_pages](#)
Number of images for linear modes.
- [l4_uint8_t linear_red_mask_size](#)
Size of direct color red mask (linear modes).
- [l4_uint8_t linear_red_field_position](#)
Bit position of LSB of red mask (linear modes).
- [l4_uint8_t linear_green_mask_size](#)
Size of direct color green mask (linear modes).
- [l4_uint8_t linear_green_field_position](#)
Bit position of LSB of green mask (linear modes).
- [l4_uint8_t linear_blue_mask_size](#)
Size of direct color blue mask (linear modes).
- [l4_uint8_t linear_blue_field_position](#)
Bit position of LSB of blue mask (linear modes).
- [l4_uint8_t linear_reserved_mask_size](#)
Size of direct color reserved mask (linear modes).
- [l4_uint8_t linear_reserved_field_position](#)
Bit position of LSB of reserved mask (linear modes).
- [l4_uint32_t max_pixel_clock](#)
Maximum pixel clock (in Hz) for graphics mode.
- [l4_uint8_t reserved3](#) [190]
Reserved (padding).

16.377.1 Detailed Description

VBE mode information.

Definition at line 125 of file [mb_info.h](#).

The documentation for this struct was generated from the following file:

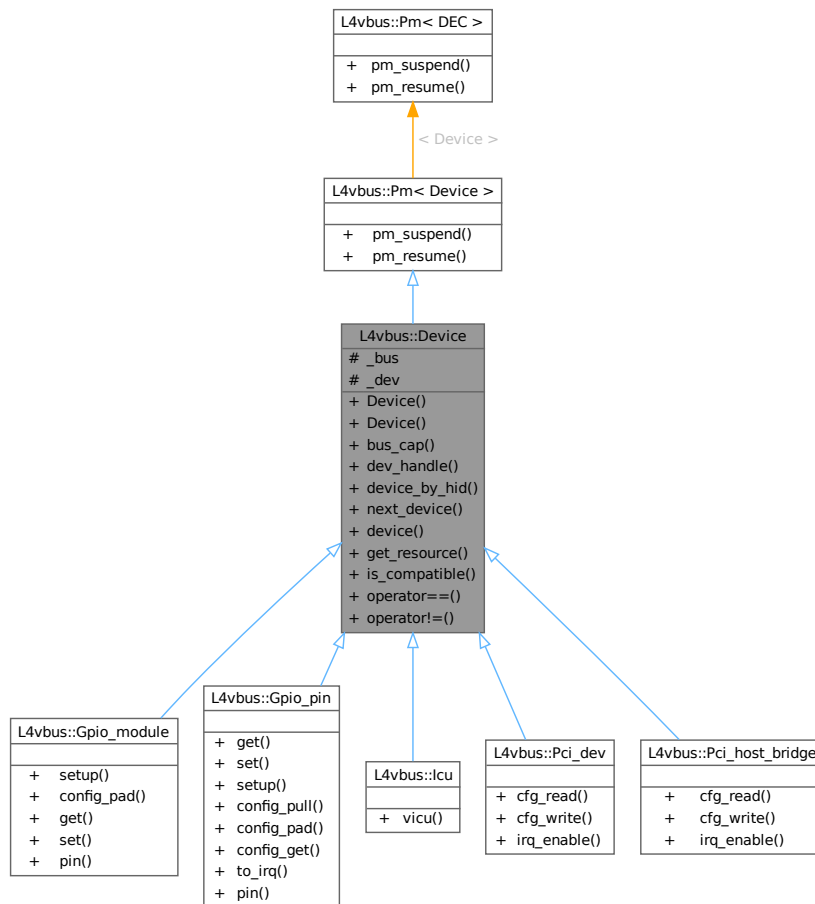
- [l4/util/mb_info.h](#)

16.378 L4vbus::Device Class Reference

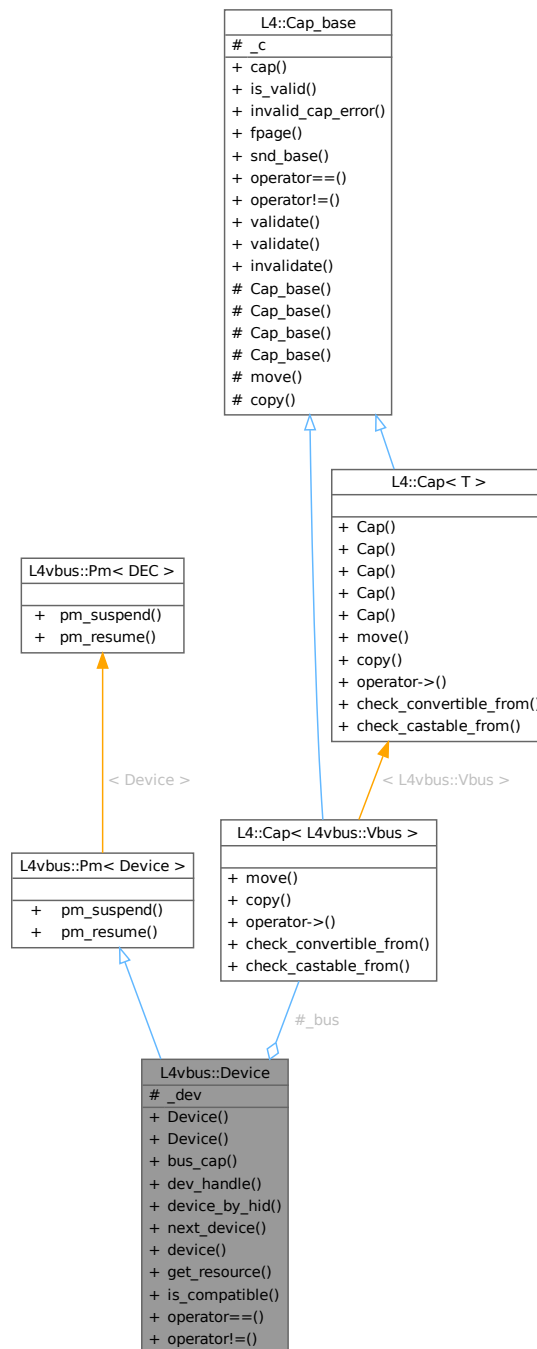
Device on a [L4vbus::Vbus](#).

```
#include <vbus>
```

Inheritance diagram for L4vbus::Device:



Collaboration diagram for L4vbus::Device:



Public Member Functions

- **Device ()**
Construct a new vbus device using the NULL device `L4VBUS_NULL`.
- **Device (L4::Cap< Vbus > bus, l4vbus_device_handle_t dev)**
Construct a new vbus device using a device handle.
- **L4::Cap< Vbus > bus_cap () const**

- Access the [Vbus](#) capability of the underlying virtual bus.

 - [l4vbus_device_handle_t dev_handle](#) () const

Access the device handle of this device.
- int [device_by_hid](#) ([Device](#) *child, char const *hid, int depth=L4VBUS_MAX_DEPTH, [l4vbus_device_t](#) *devinfo=0) const

Find a device by the hardware interface identifier (HID).
- int [next_device](#) ([Device](#) *child, int depth=L4VBUS_MAX_DEPTH, [l4vbus_device_t](#) *devinfo=0) const

Find next child following *child*.
- int [device](#) ([l4vbus_device_t](#) *devinfo) const

Obtain detailed information about a [Vbus](#) device.
- int [get_resource](#) (unsigned res_idx, [l4vbus_resource_t](#) *res) const

Obtain the resource description of an individual device resource.
- int [is_compatible](#) (char const *cid) const

Check if the given device has a compatibility ID (CID) or HID that matches *cid*.
- bool [operator==](#) ([Device](#) const &o) const

Test if two devices are the same [Vbus](#) device.
- bool [operator!=](#) ([Device](#) const &o) const

Test if two [Vbus](#) devices are not the same.

Public Member Functions inherited from [L4vbus::Pm](#)< [Device](#) >

- int [pm_suspend](#) () const

Suspend the device.
- int [pm_resume](#) () const

Resume the device.

Protected Attributes

- [L4::Cap](#)< [Vbus](#) > [_bus](#)
- The [Vbus](#) capability where this device is located on.
- [l4vbus_device_handle_t](#) [_dev](#)
- The device handle for this device.

16.378.1 Detailed Description

[Device](#) on a [L4vbus::Vbus](#).

Definition at line 83 of file [vbus](#).

16.378.2 Constructor & Destructor Documentation

16.378.2.1 Device()

```
L4vbus::Device::Device (
    L4::Cap< Vbus > bus,
    l4vbus\_device\_handle\_t dev) [inline]
```

Construct a new vbus device using a device handle.

Specifying the special root bus device handle [L4VBUS_ROOT_BUS](#) forms the root device of the corresponding vbus, see [Vbus::root](#).

Parameters

<i>bus</i>	The vbus capability where this device is assigned.
<i>dev</i>	The device handle of the device.

Definition at line 100 of file [vbus](#).

References [_bus](#), and [_dev](#).

16.378.3 Member Function Documentation

16.378.3.1 bus_cap()

```
L4::Cap< Vbus > L4vbus::Device::bus_cap () const [inline]
```

Access the [Vbus](#) capability of the underlying virtual bus.

Returns

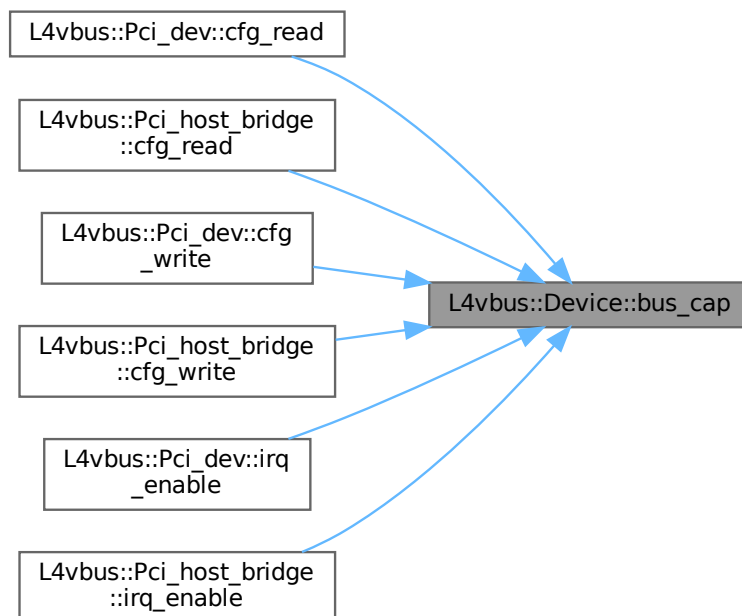
the capability to the underlying [Vbus](#).

Definition at line 107 of file [vbus](#).

References [_bus](#).

Referenced by [L4vbus::Pci_dev::cfg_read\(\)](#), [L4vbus::Pci_host_bridge::cfg_read\(\)](#), [L4vbus::Pci_dev::cfg_write\(\)](#), [L4vbus::Pci_host_bridge::cfg_write\(\)](#), [L4vbus::Pci_dev::irq_enable\(\)](#), and [L4vbus::Pci_host_bridge::irq_enable\(\)](#).

Here is the caller graph for this function:



16.378.3.2 dev_handle()

```
l4vbus_device_handle_t L4vbus::Device::dev_handle () const [inline]
```

Access the device handle of this device.

Returns

the device handle for this device.

The device handle is used to directly address the device on its virtual bus.

Definition at line 116 of file [vbus](#).

References [_dev](#).

16.378.3.3 device()

```
int L4vbus::Device::device (  
    l4vbus_device_t * devinfo) const [inline]
```

Obtain detailed information about a [Vbus](#) device.

Parameters

out	<i>devinfo</i>	Information structure which contains details about the device. The pointer might be NULL.
-----	----------------	---

Return values

0	Success.
-L4_ENODEV	No device with the given device handle <i>dev</i> could be found.

Definition at line 189 of file [vbus](#).

References [_bus](#), [_dev](#), and [l4vbus_get_device\(\)](#).

Here is the call graph for this function:



16.378.3.4 device_by_hid()

```
int L4vbus::Device::device_by_hid (
    Device * child,
    char const * hid,
    int depth = L4VBUS_MAX_DEPTH,
    l4vbus_device_t * devinfo = 0) const [inline]
```

Find a device by the hardware interface identifier (HID).

This function searches the vbus for a device with the given HID and returns a handle to the first matching device. The HID usually conforms to an ACPI HID or a Linux device tree compatible identifier.

It is possible to have multiple devices with the same HID on a vbus. In order to find all matching devices this function has to be called repeatedly with `child` pointing to the device found in the previous iteration. The iteration starts at `child` that might be any device node in the tree.

Parameters

in, out	<i>child</i>	Handle of the device from where in the device tree the search should start. To start searching from the beginning <code>child</code> must be initialized using the default (L4VBUS_NULL). If a matching device is found, its handle is returned through this parameter.
	<i>hid</i>	HID of the device
	<i>depth</i>	Maximum depth for the recursive lookup
out	<i>devinfo</i>	Device information structure (might be NULL)

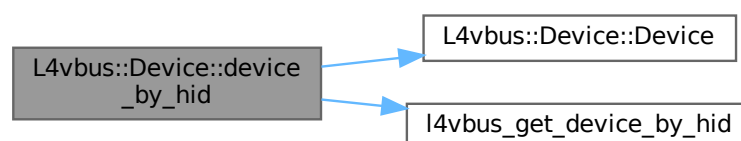
Return values

<code>>=0</code>	A device with the given HID was found.
<code>-L4_ENOENT</code>	No device with the given HID could be found on the vbus.
<code>-L4_EINVAL</code>	Invalid or no HID provided.
<code>-L4_ENODEV</code>	Function called on a non-existing device.

Definition at line [148](#) of file [vbus](#).

References [_bus](#), [_dev](#), [Device\(\)](#), and [l4vbus_get_device_by_hid\(\)](#).

Here is the call graph for this function:



16.378.3.5 get_resource()

```
int L4vbus::Device::get_resource (
    unsigned res_idx,
    l4vbus_resource_t * res) const [inline]
```

Obtain the resource description of an individual device resource.

Parameters

	<i>res_idx</i>	Index of the resource for which the resource description should be returned. The total number of resources for a device is available in the l4vbus_device_t structure that is returned by L4vbus::Device::device_by_hid() and L4vbus::Device::next_device() .
out	<i>res</i>	Descriptor of the resource.

This function returns the resource descriptor of an individual device resource selected by the *res_idx* parameter.

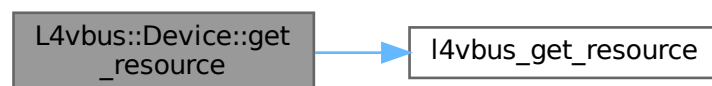
Return values

0	Success.
-L4_ENOENT	Invalid resource index <i>res_idx</i> .

Definition at line 209 of file [vbus](#).

References [_bus](#), [_dev](#), and [l4vbus_get_resource\(\)](#).

Here is the call graph for this function:



16.378.3.6 is_compatible()

```
int L4vbus::Device::is_compatible (
    char const * cid) const [inline]
```

Check if the given device has a compatibility ID (CID) or HID that matches *cid*.

Parameters

<i>cid</i>	the compatibility ID to test
------------	------------------------------

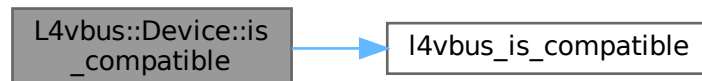
Returns

1 when the given ID (*cid*) matches this device, 0 when the given ID does not match, <0 on error.

Definition at line 223 of file [vbus](#).

References [_bus](#), [_dev](#), and [l4vbus_is_compatible\(\)](#).

Here is the call graph for this function:

**16.378.3.7 next_device()**

```

int L4vbus::Device::next_device (
    Device * child,
    int depth = L4VBUS_MAX_DEPTH,
    l4vbus_device_t * devinfo = 0) const [inline]
  
```

Find next child following `child`.

Parameters

in, out	<i>child</i>	Handle of the device that precedes the device that shall be returned. To start from the beginning, <i>child</i> must be initialized with L4VBUS_NULL (Device::Device). If a device is found, its handle is returned through this parameter.
	<i>depth</i>	Depth to look for
out	<i>devinfo</i>	Device information (might be NULL)

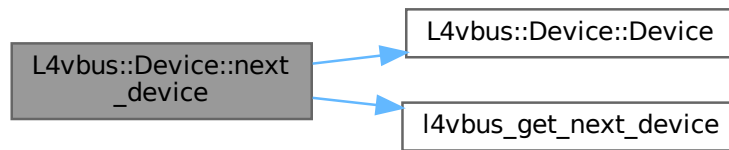
Returns

0 on success, else failure

Definition at line 171 of file [vbus](#).

References [_bus](#), [_dev](#), [Device\(\)](#), and [l4vbus_get_next_device\(\)](#).

Here is the call graph for this function:



16.378.3.8 `operator!=()`

```
bool L4vbus::Device::operator!= (
    Device const & o) const [inline]
```

Test if two [Vbus](#) devices are not the same.

Returns

true if the two devices are different, false else.

Definition at line [239](#) of file [vbus](#).

References [_bus](#), [_dev](#), and [Device\(\)](#).

Here is the call graph for this function:



16.378.3.9 `operator==()`

```
bool L4vbus::Device::operator== (
    Device const & o) const [inline]
```

Test if two devices are the same [Vbus](#) device.

Returns

true if the two devices are the same, false else.

Definition at line 230 of file [vbus](#).

References [_bus](#), [_dev](#), and [Device\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

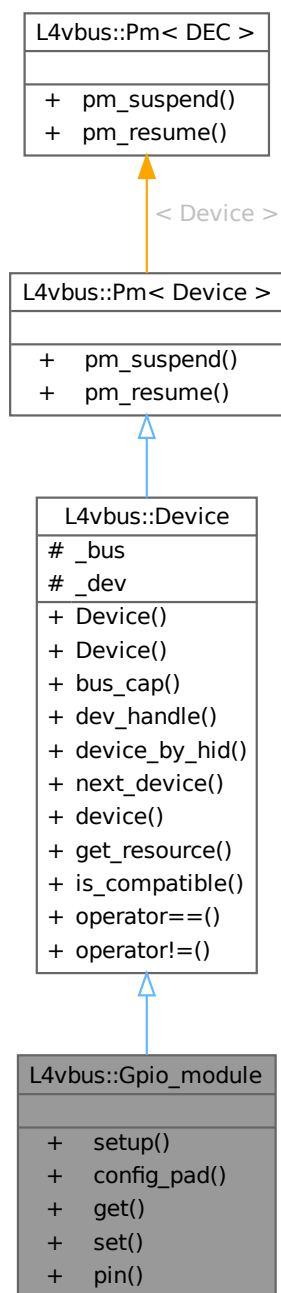
- `I4/vbus/vbus`

16.379 L4vbus::Gpio_module Class Reference

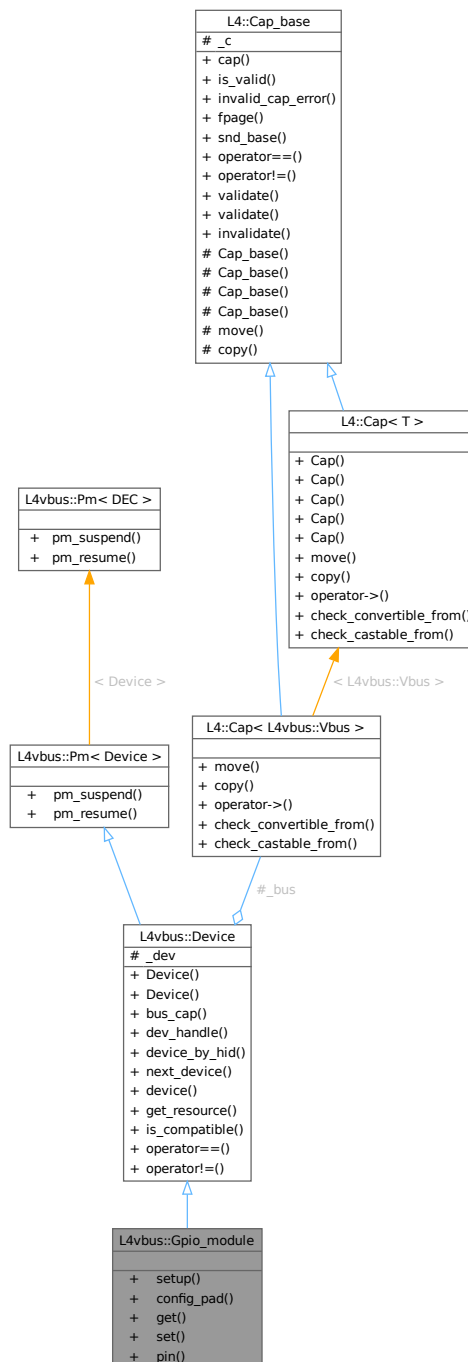
A [Gpio_module](#) groups multiple GPIO pins together.

```
#include <vbus_gpio>
```

Inheritance diagram for L4vbus::Gpio_module:



Collaboration diagram for L4vbus::Gpio_module:



Data Structures

- struct `Pin_slice`
A slice of the pins provided by this module.

Public Member Functions

- int **setup** (Pin_slice const &mask, unsigned mode, unsigned value) const

- *Configure function of multiple GPIO pins at once.*
int [config_pad](#) ([Pin_slice](#) const &mask, unsigned func, unsigned value) const
- *Hardware specific configuration function for multiple GPIO pins.*
int [get](#) (unsigned offset, unsigned *data) const
- *Read values of multiple GPIO pins at once.*
int [set](#) ([Pin_slice](#) const &mask, unsigned data)
- *Set multiple GPIO output pins at once.*
[Gpio_pin](#) [pin](#) (unsigned pin) const
- *Get [Gpio_pin](#) for a specific pin of this [Gpio_module](#).*

Public Member Functions inherited from [L4vbus::Device](#)

- [Device](#) ()
Construct a new vbus device using the NULL device [L4VBUS_NULL](#).
- [Device](#) ([L4::Cap](#)< [Vbus](#) > bus, [l4vbus_device_handle_t](#) dev)
Construct a new vbus device using a device handle.
- [L4::Cap](#)< [Vbus](#) > [bus_cap](#) () const
Access the [Vbus](#) capability of the underlying virtual bus.
- [l4vbus_device_handle_t](#) [dev_handle](#) () const
Access the device handle of this device.
- int [device_by_hid](#) ([Device](#) *child, char const *hid, int depth=L4VBUS_MAX_DEPTH, [l4vbus_device_t](#) *devinfo=0) const
Find a device by the hardware interface identifier (HID).
- int [next_device](#) ([Device](#) *child, int depth=L4VBUS_MAX_DEPTH, [l4vbus_device_t](#) *devinfo=0) const
Find next child following [child](#).
- int [device](#) ([l4vbus_device_t](#) *devinfo) const
Obtain detailed information about a [Vbus](#) device.
- int [get_resource](#) (unsigned res_idx, [l4vbus_resource_t](#) *res) const
Obtain the resource description of an individual device resource.
- int [is_compatible](#) (char const *cid) const
Check if the given device has a compatibility ID (CID) or HID that matches [cid](#).
- bool [operator==](#) ([Device](#) const &o) const
Test if two devices are the same [Vbus](#) device.
- bool [operator!=](#) ([Device](#) const &o) const
Test if two [Vbus](#) devices are not the same.

Public Member Functions inherited from [L4vbus::Pm](#)< [Device](#) >

- int [pm_suspend](#) () const
Suspend the device.
- int [pm_resume](#) () const
Resume the device.

Additional Inherited Members

Protected Attributes inherited from [L4vbus::Device](#)

- [L4::Cap](#)< [Vbus](#) > [_bus](#)
The [Vbus](#) capability where this device is located on.
- [l4vbus_device_handle_t](#) [_dev](#)
The device handle for this device.

16.379.1 Detailed Description

A [Gpio_module](#) groups multiple GPIO pins together.

Definition at line 133 of file [vbus_gpio](#).

16.379.2 Member Function Documentation

16.379.2.1 config_pad()

```
int L4vbus::Gpio_module::config_pad (  
    Pin\_slice const & mask,  
    unsigned func,  
    unsigned value) const [inline]
```

Hardware specific configuration function for multiple GPIO pins.

Parameters

<i>mask</i>	Mask of GPIO pins to configure. A bit set to 1 configures this pin. A maximum of 32 pins can be configured at once. The real number depends on the hardware and the driver implementation.
<i>func</i>	Hardware specific configuration register, usually offset to the GPIO chip's base address.
<i>value</i>	Value which is written into the hardware specific configuration register for the specified pins

Returns

0 if OK, error code otherwise

Definition at line 185 of file [vbus_gpio](#).

References [L4vbus::Device::_bus](#), [L4vbus::Device::_dev](#), and [l4vbus_gpio_multi_config_pad\(\)](#).

Here is the call graph for this function:



16.379.2.2 get()

```
int L4vbus::Gpio_module::get (  
    unsigned offset,  
    unsigned * data) const [inline]
```

Read values of multiple GPIO pins at once.

Parameters

	<i>offset</i>	Pin corresponding to the LSB in <i>data</i> . Note: allowed may be hardware specific.
out	<i>data</i>	Each bit returns the value (0 or 1) for the corresponding GPIO pin. The value of pins that are not accessible is undefined.

Returns

0 if OK, error code otherwise

Definition at line 201 of file [vbus_gpio](#).

References [L4vbus::Device::_bus](#), [L4vbus::Device::_dev](#), and [l4vbus_gpio_multi_get\(\)](#).

Here is the call graph for this function:



16.379.2.3 pin()

```
Gpio_pin L4vbus::Gpio_module::pin (  
    unsigned pin) const [inline]
```

Get [Gpio_pin](#) for a specific pin of this [Gpio_module](#).

Parameters

<i>pin</i>	GPIO pin number to get Gpio_pin for.
------------	--

Returns

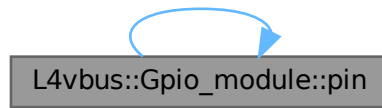
[Gpio_pin](#)

Definition at line 229 of file [vbus_gpio](#).

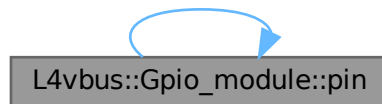
References [pin\(\)](#).

Referenced by [pin\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.379.2.4 set()

```
int L4vbus::Gpio_module::set (  
    Pin_slice const & mask,  
    unsigned data) [inline]
```

Set multiple GPIO output pins at once.

Parameters

<i>mask</i>	Mask of GPIO pins to set. A bit set to 1 selects this pin. A maximum of 32 pins can be set at once. The real number depends on the hardware and the driver implementation.
<i>data</i>	Each bit corresponds to the GPIO pin in <i>mask</i> . The value of each bit is written to the GPIO pin if its bit in <i>mask</i> is set.

Returns

0 if OK, error code otherwise

Definition at line 217 of file `vbus_gpio`.

References `L4vbus::Device::_bus`, `L4vbus::Device::_dev`, and `l4vbus_gpio_multi_set()`.

Here is the call graph for this function:



16.379.2.5 setup()

```
int L4vbus::Gpio_module::setup (
    Pin_slice const & mask,
    unsigned mode,
    unsigned value) const [inline]
```

Configure function of multiple GPIO pins at once.

Parameters

<i>mask</i>	Mask of GPIO pins to configure. A bit set to 1 configures this pin. A maximum of 32 pins can be configured at once. The real number depends on the hardware and the driver implementation.
<i>mode</i>	GPIO function, see L4vbus_gpio_generic_func for generic functions. Hardware specific functions must be provided in the lower 8 bits.
<i>value</i>	Optional value to set the GPIO pins to if they are configured as output pins

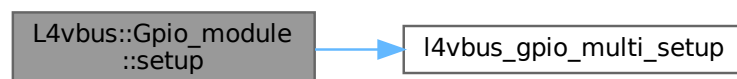
Returns

0 if OK, error code otherwise

Definition at line 166 of file [vbus_gpio](#).

References [L4vbus::Device::_bus](#), [L4vbus::Device::_dev](#), and [l4vbus_gpio_multi_setup\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

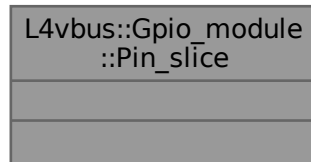
- [l4/vbus/vbus_gpio](#)

16.380 L4vbus::Gpio_module::Pin_slice Struct Reference

A slice of the pins provided by this module.

```
#include <vbus_gpio>
```

Collaboration diagram for L4vbus::Gpio_module::Pin_slice:



16.380.1 Detailed Description

A slice of the pins provided by this module.

Data type to specify a selection of pins for the 'multi' methods.

Definition at line [146](#) of file [vbus_gpio](#).

The documentation for this struct was generated from the following file:

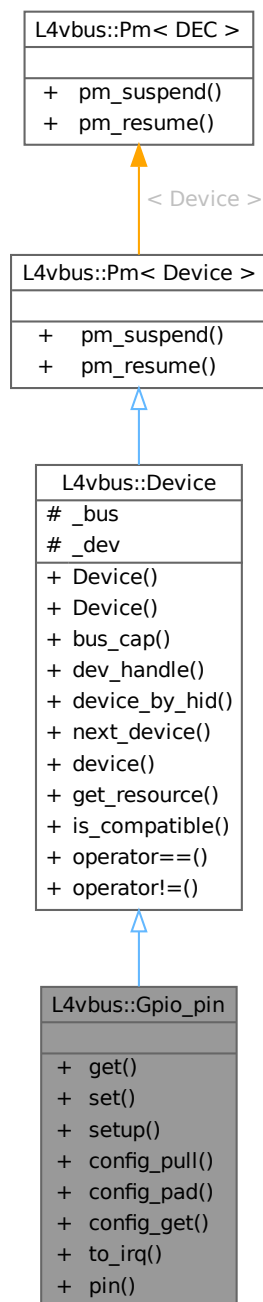
- l4/vbus/vbus_gpio

16.381 L4vbus::Gpio_pin Class Reference

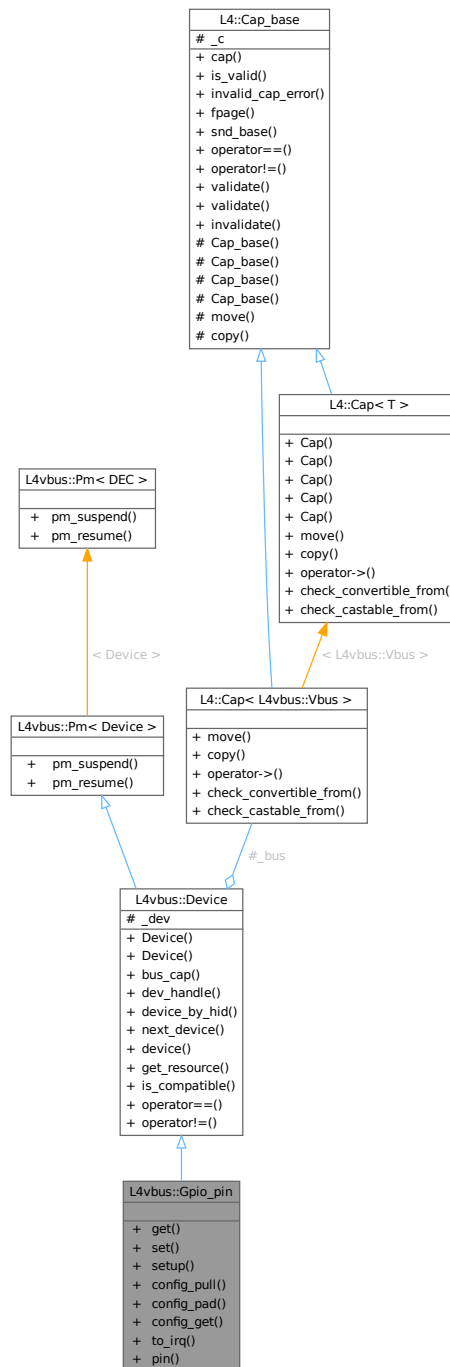
A GPIO pin.

```
#include <vbus_gpio>
```

Inheritance diagram for L4vbus::Gpio_pin:



Collaboration diagram for L4vbus::Gpio_pin:



Public Member Functions

- int [get](#) () const
Read value of GPIO input pin.
- int [set](#) (int value) const
Set GPIO output pin.
- int [setup](#) (unsigned mode, unsigned value) const

- *Configure the function of a GPIO pin.*
- int [config_pull](#) (unsigned mode) const
Generic function to set pull up/down mode.
- int [config_pad](#) (unsigned func, unsigned value) const
Hardware specific configuration function.
- int [config_get](#) (unsigned func, unsigned *value) const
Read hardware specific configuration.
- int [to_irq](#) () const
Create IRQ for GPIO pin.
- unsigned [pin](#) () const
Get pin number.

Public Member Functions inherited from [L4vbus::Device](#)

- [Device](#) ()
Construct a new vbus device using the NULL device [L4VBUS_NULL](#).
- [Device](#) ([L4::Cap](#)< [Vbus](#) > bus, [l4vbus_device_handle_t](#) dev)
Construct a new vbus device using a device handle.
- [L4::Cap](#)< [Vbus](#) > [bus_cap](#) () const
Access the [Vbus](#) capability of the underlying virtual bus.
- [l4vbus_device_handle_t](#) [dev_handle](#) () const
Access the device handle of this device.
- int [device_by_hid](#) ([Device](#) *child, char const *hid, int depth=L4VBUS_MAX_DEPTH, [l4vbus_device_t](#) *devinfo=0) const
Find a device by the hardware interface identifier (HID).
- int [next_device](#) ([Device](#) *child, int depth=L4VBUS_MAX_DEPTH, [l4vbus_device_t](#) *devinfo=0) const
*Find next child following *child*.*
- int [device](#) ([l4vbus_device_t](#) *devinfo) const
Obtain detailed information about a [Vbus](#) device.
- int [get_resource](#) (unsigned res_idx, [l4vbus_resource_t](#) *res) const
Obtain the resource description of an individual device resource.
- int [is_compatible](#) (char const *cid) const
*Check if the given device has a compatibility ID (CID) or HID that matches *cid*.*
- bool [operator==](#) ([Device](#) const &o) const
Test if two devices are the same [Vbus](#) device.
- bool [operator!=](#) ([Device](#) const &o) const
Test if two [Vbus](#) devices are not the same.

Public Member Functions inherited from [L4vbus::Pm](#)< [Device](#) >

- int [pm_suspend](#) () const
Suspend the device.
- int [pm_resume](#) () const
Resume the device.

Additional Inherited Members

Protected Attributes inherited from [L4vbus::Device](#)

- [L4::Cap< Vbus > _bus](#)
The [Vbus](#) capability where this device is located on.
- [l4vbus_device_handle_t _dev](#)
The device handle for this device.

16.381.1 Detailed Description

A GPIO pin.

Definition at line 26 of file [vbus_gpio](#).

16.381.2 Member Function Documentation

16.381.2.1 `config_get()`

```
int L4vbus::Gpio_pin::config_get (
    unsigned func,
    unsigned * value) const [inline]
```

Read hardware specific configuration.

Parameters

	<i>func</i>	Hardware specific configuration register to read from. Usually this is an offset to the GPIO chip's base address.
out	<i>value</i>	The configuration value.

Returns

0 if OK, error code otherwise

Definition at line 102 of file [vbus_gpio](#).

References [L4vbus::Device::_bus](#), [L4vbus::Device::_dev](#), and [l4vbus_gpio_config_get\(\)](#).

Here is the call graph for this function:



16.381.2.2 config_pad()

```
int L4vbus::Gpio_pin::config_pad (  
    unsigned func,  
    unsigned value) const [inline]
```

Hardware specific configuration function.

Parameters

<i>func</i>	Hardware specific configuration register, usually offset to the GPIO chip's base address
<i>value</i>	Value which is written into the hardware specific configuration register for the specified pin

Returns

0 if OK, error code otherwise

Definition at line 89 of file [vbus_gpio](#).

References [L4vbus::Device::_bus](#), [L4vbus::Device::_dev](#), and [l4vbus_gpio_config_pad\(\)](#).

Here is the call graph for this function:



16.381.2.3 config_pull()

```
int L4vbus::Gpio_pin::config_pull (  
    unsigned mode) const [inline]
```

Generic function to set pull up/down mode.

Parameters

<i>mode</i>	mode for pull up/down resistors, see L4vbus_gpio_pull_modes
-------------	---

Returns

0 if OK, error code otherwise

Definition at line 75 of file [vbus_gpio](#).

References [L4vbus::Device::_bus](#), [L4vbus::Device::_dev](#), and [l4vbus_gpio_config_pull\(\)](#).

Here is the call graph for this function:



16.381.2.4 get()

```
int L4vbus::Gpio_pin::get () const [inline]
```

Read value of GPIO input pin.

Returns

Value of GPIO pin (usually 0 or 1), negative error code otherwise.

Definition at line 38 of file [vbus_gpio](#).

References [L4vbus::Device::_bus](#), [L4vbus::Device::_dev](#), and [l4vbus_gpio_get\(\)](#).

Here is the call graph for this function:



16.381.2.5 pin()

```
unsigned L4vbus::Gpio_pin::pin () const [inline]
```

Get pin number.

Returns

GPIO pin number

Definition at line 122 of file [vbus_gpio](#).

16.381.2.6 set()

```
int L4vbus::Gpio_pin::set (  
    int value) const [inline]
```

Set GPIO output pin.

Parameters

<i>value</i>	Value to write to the GPIO pin (usually 0 or 1)
--------------	---

Returns

0 if OK, error code otherwise

Definition at line 49 of file [vbus_gpio](#).

References [L4vbus::Device::_bus](#), [L4vbus::Device::_dev](#), and [l4vbus_gpio_set\(\)](#).

Here is the call graph for this function:

**16.381.2.7 setup()**

```
int L4vbus::Gpio_pin::setup (  
    unsigned mode,  
    unsigned value) const [inline]
```

Configure the function of a GPIO pin.

Parameters

<i>mode</i>	GPIO function, see L4vbus_gpio_generic_func for generic functions. Hardware specific functions must be provided in the lower 8 bits.
<i>value</i>	Optional value to set the GPIO pin to if it is configured as an output pin

Returns

0 if OK, error code otherwise

Definition at line 64 of file [vbus_gpio](#).

References [L4vbus::Device::_bus](#), [L4vbus::Device::_dev](#), and [l4vbus_gpio_setup\(\)](#).

Here is the call graph for this function:



16.381.2.8 to_irq()

```
int L4vbus::Gpio_pin::to_irq () const [inline]
```

Create IRQ for GPIO pin.

Returns

IRQ number if OK, negative error code otherwise

Definition at line 112 of file [vbus_gpio](#).

References [L4vbus::Device::_bus](#), [L4vbus::Device::_dev](#), and [l4vbus_gpio_to_irq\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

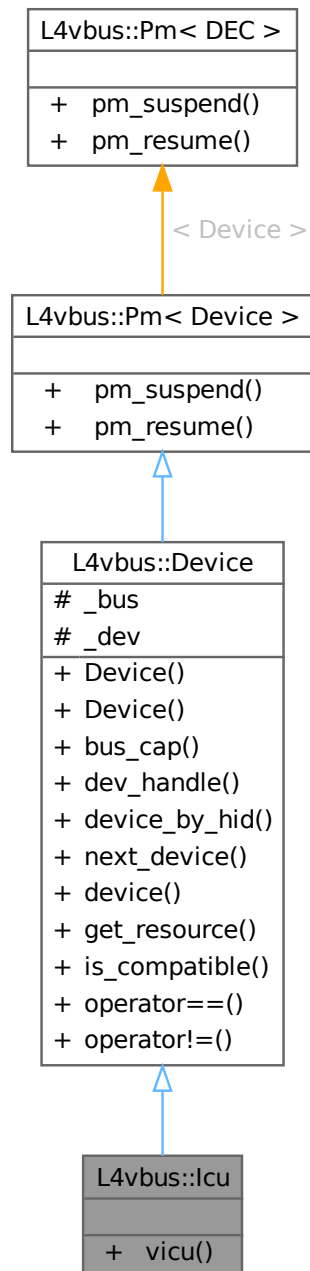
- [l4/vbus/vbus_gpio](#)

16.382 L4vbus::lcu Class Reference

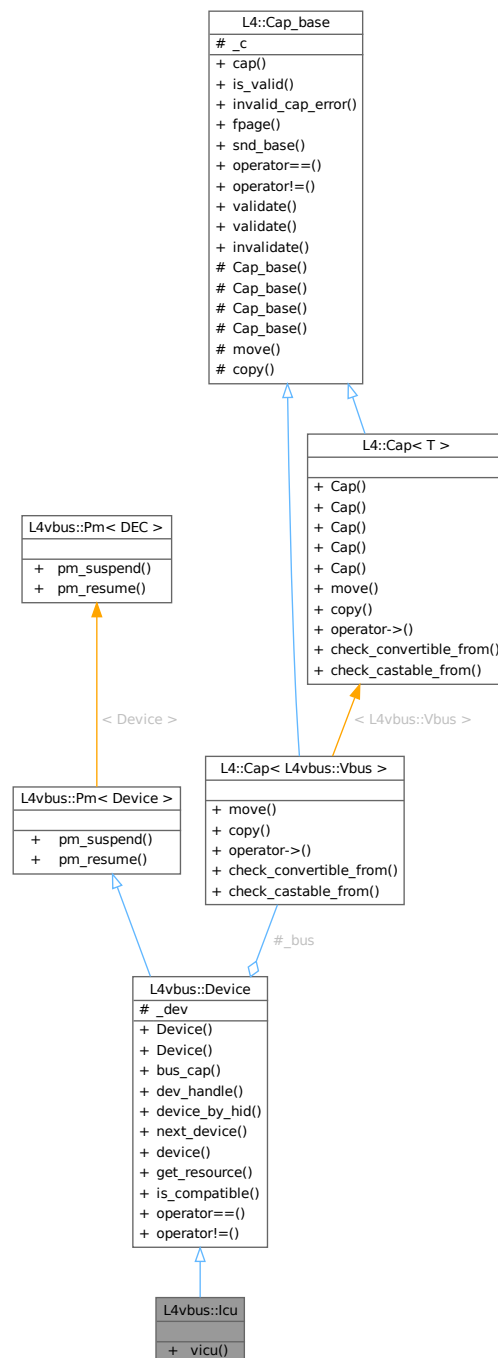
[Vbus](#) Interrupt controller API.

```
#include <vbus>
```

Inheritance diagram for L4vbus::lcu:



Collaboration diagram for L4vbus::lcu:



Public Types

- enum `Src_types` { `Src_dev_handle` = `L4VBUS_ICU_SRC_DEV_HANDLE` }
Flags that can be used with the ICU on a vbus device.

Public Member Functions

- int `vicu` (`L4::Cap< L4::lcu >` icu) const

Request an [L4::Icu](#) capability for this [Vbus](#)'s virtual ICU.

Public Member Functions inherited from [L4vbus::Device](#)

- [Device](#) ()
Construct a new vbus device using the NULL device [L4VBUS_NULL](#).
- [Device](#) ([L4::Cap](#)< [Vbus](#) > bus, [l4vbus_device_handle_t](#) dev)
Construct a new vbus device using a device handle.
- [L4::Cap](#)< [Vbus](#) > [bus_cap](#) () const
Access the [Vbus](#) capability of the underlying virtual bus.
- [l4vbus_device_handle_t](#) [dev_handle](#) () const
Access the device handle of this device.
- int [device_by_hid](#) ([Device](#) *child, char const *hid, int depth=L4VBUS_MAX_DEPTH, [l4vbus_device_t](#) *devinfo=0) const
Find a device by the hardware interface identifier (HID).
- int [next_device](#) ([Device](#) *child, int depth=L4VBUS_MAX_DEPTH, [l4vbus_device_t](#) *devinfo=0) const
Find next child following [child](#).
- int [device](#) ([l4vbus_device_t](#) *devinfo) const
Obtain detailed information about a [Vbus](#) device.
- int [get_resource](#) (unsigned res_idx, [l4vbus_resource_t](#) *res) const
Obtain the resource description of an individual device resource.
- int [is_compatible](#) (char const *cid) const
Check if the given device has a compatibility ID (CID) or HID that matches [cid](#).
- bool [operator==](#) ([Device](#) const &o) const
Test if two devices are the same [Vbus](#) device.
- bool [operator!=](#) ([Device](#) const &o) const
Test if two [Vbus](#) devices are not the same.

Public Member Functions inherited from [L4vbus::Pm](#)< [Device](#) >

- int [pm_suspend](#) () const
Suspend the device.
- int [pm_resume](#) () const
Resume the device.

Additional Inherited Members

Protected Attributes inherited from [L4vbus::Device](#)

- [L4::Cap](#)< [Vbus](#) > [_bus](#)
The [Vbus](#) capability where this device is located on.
- [l4vbus_device_handle_t](#) [_dev](#)
The device handle for this device.

16.382.1 Detailed Description

[Vbus](#) Interrupt controller API.

Every [Vbus](#) contains a virtual interrupt control unit that manages the IRQs of the devices on the bus. This class provides access to a capability to an [L4::Icu](#) object which can then be used to interface with the IRQs.

See also

[L4::Icu](#)

Definition at line 260 of file [vbus](#).

16.382.2 Member Enumeration Documentation

16.382.2.1 Src_types

```
enum L4vbus::Icu::Src_types
```

Flags that can be used with the ICU on a vbus device.

Enumerator

Src_dev_handle	Flag to denote that the value should be interpreted as a device handle. This flag may be used in the <code>source</code> parameter in L4::Icu::msi_info() to denote that the ICU should interpret the source ID as a device handle.
----------------	---

Definition at line 264 of file [vbus](#).

16.382.3 Member Function Documentation

16.382.3.1 vicu()

```
int L4vbus::Icu::vicu (  
    L4::Cap< L4::Icu > icu) const [inline]
```

Request an [L4::Icu](#) capability for this [Vbus](#)'s virtual ICU.

Parameters

<i>out</i>	<i>icu</i>	Capability slot where the L4::Icu capability shall be stored.
------------	------------	---

Return values

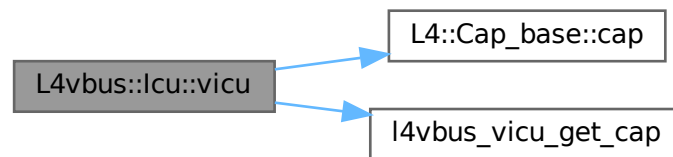
0	Success.
---	----------

<i>otherwise</i>	IPC error.
------------------	------------

Definition at line 285 of file [vbus](#).

References [L4vbus::Device::_bus](#), [L4vbus::Device::_dev](#), [L4::Cap_base::cap\(\)](#), and [l4vbus_vicu_get_cap\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

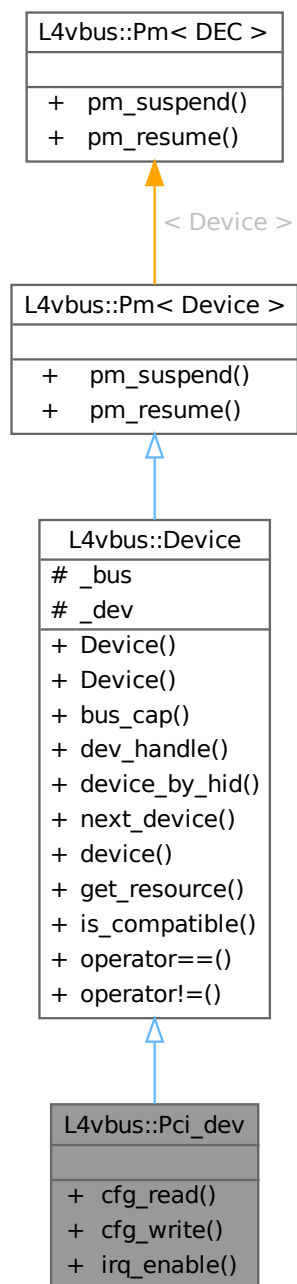
- `l4/vbus/vbus`

16.383 L4vbus::Pci_dev Class Reference

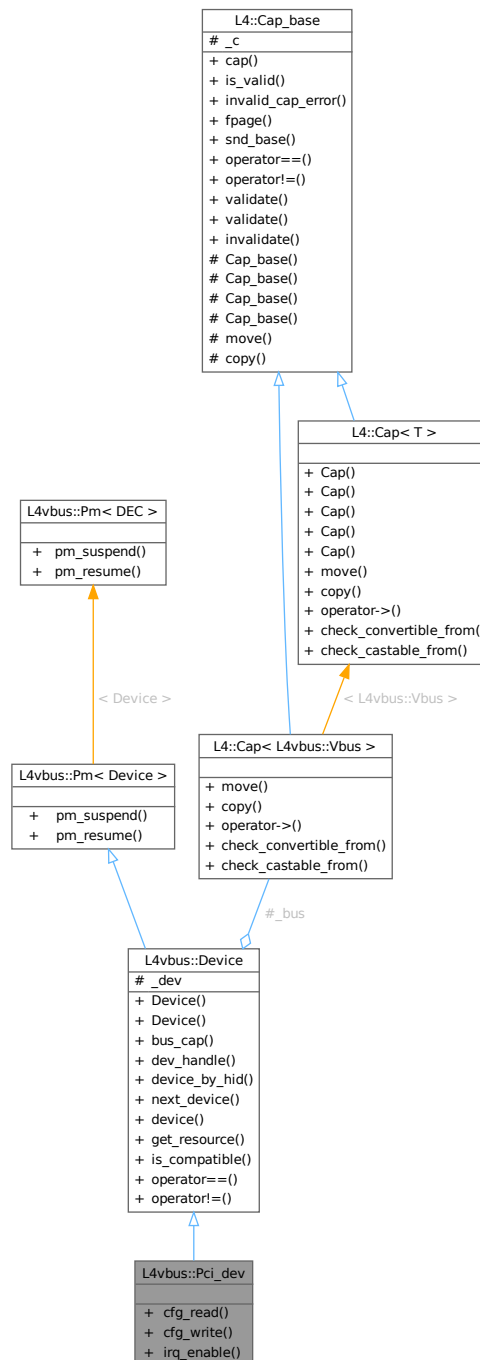
A PCI device.

```
#include <vbus_pci>
```

Inheritance diagram for L4vbus::Pci_dev:



Collaboration diagram for L4vbus::Pci_dev:



Public Member Functions

- `int cfg_read (l4_uint32_t reg, l4_uint32_t *value, l4_uint32_t width) const`
Read from the device's vPCI configuration space.
- `int cfg_write (l4_uint32_t reg, l4_uint32_t value, l4_uint32_t width) const`
Write to the device's vPCI configuration space.
- `int irq_enable (unsigned char *trigger, unsigned char *polarity) const`
Enable the device's PCI interrupt.

Public Member Functions inherited from L4vbus::Device

- **Device** ()
Construct a new vbus device using the NULL device L4VBUS_NULL.
- **Device** (L4::Cap< Vbus > bus, l4vbus_device_handle_t dev)
Construct a new vbus device using a device handle.
- **L4::Cap< Vbus > bus_cap** () const
Access the Vbus capability of the underlying virtual bus.
- **l4vbus_device_handle_t dev_handle** () const
Access the device handle of this device.
- **int device_by_hid** (Device *child, char const *hid, int depth=L4VBUS_MAX_DEPTH, l4vbus_device_t *devinfo=0) const
Find a device by the hardware interface identifier (HID).
- **int next_device** (Device *child, int depth=L4VBUS_MAX_DEPTH, l4vbus_device_t *devinfo=0) const
Find next child following child.
- **int device** (l4vbus_device_t *devinfo) const
Obtain detailed information about a Vbus device.
- **int get_resource** (unsigned res_idx, l4vbus_resource_t *res) const
Obtain the resource description of an individual device resource.
- **int is_compatible** (char const *cid) const
Check if the given device has a compatibility ID (CID) or HID that matches cid.
- **bool operator==** (Device const &o) const
Test if two devices are the same Vbus device.
- **bool operator!=** (Device const &o) const
Test if two Vbus devices are not the same.

Public Member Functions inherited from L4vbus::Pm< Device >

- **int pm_suspend** () const
Suspend the device.
- **int pm_resume** () const
Resume the device.

Additional Inherited Members

Protected Attributes inherited from L4vbus::Device

- **L4::Cap< Vbus > _bus**
The Vbus capability where this device is located on.
- **l4vbus_device_handle_t _dev**
The device handle for this device.

16.383.1 Detailed Description

A PCI device.

Definition at line 93 of file `vbus_pci`.

16.383.2 Member Function Documentation

16.383.2.1 `cfg_read()`

```
int L4vbus::Pci_dev::cfg_read (
    14_uint32_t reg,
    14_uint32_t * value,
    14_uint32_t width) const [inline]
```

Read from the device's vPCI configuration space.

Parameters

	<i>reg</i>	Register in configuration space to read
out	<i>value</i>	Value that has been read
	<i>width</i>	Width to read in bits (e.g. 8, 16, 32)

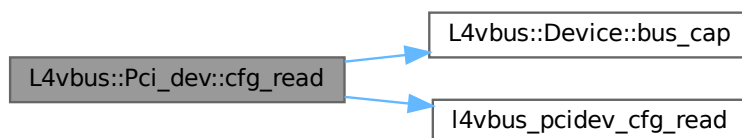
Returns

0 on success, else failure

Definition at line 105 of file `vbus_pci`.

References `L4vbus::Device::_dev`, `L4vbus::Device::bus_cap()`, and `l4vbus_pcidev_cfg_read()`.

Here is the call graph for this function:



16.383.2.2 `cfg_write()`

```
int L4vbus::Pci_dev::cfg_write (
    14_uint32_t reg,
    14_uint32_t value,
    14_uint32_t width) const [inline]
```

Write to the device's vPCI configuration space.

Parameters

<i>reg</i>	Register in configuration space to write
<i>value</i>	Value to write
<i>width</i>	Width to write in bits (e.g. 8, 16, 32)

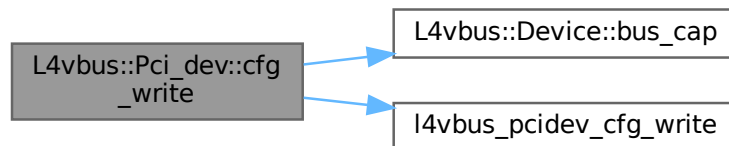
Returns

0 on success, else failure

Definition at line 121 of file [vbus_pci](#).

References [L4vbus::Device::_dev](#), [L4vbus::Device::bus_cap\(\)](#), and [l4vbus_pciddev_cfg_write\(\)](#).

Here is the call graph for this function:



16.383.2.3 irq_enable()

```
int L4vbus::Pci_dev::irq_enable (  
    unsigned char * trigger,  
    unsigned char * polarity) const [inline]
```

Enable the device's PCI interrupt.

Parameters

out	<i>trigger</i>	False if interrupt is level-triggered
out	<i>polarity</i>	True if interrupt is of low polarity

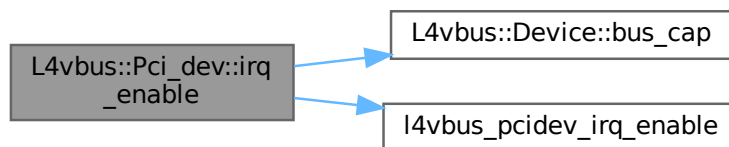
Returns

On success: Interrupt line to be used, else failure

Definition at line 137 of file [vbus_pci](#).

References [L4vbus::Device::_dev](#), [L4vbus::Device::bus_cap\(\)](#), and [l4vbus_pcidev_irq_enable\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

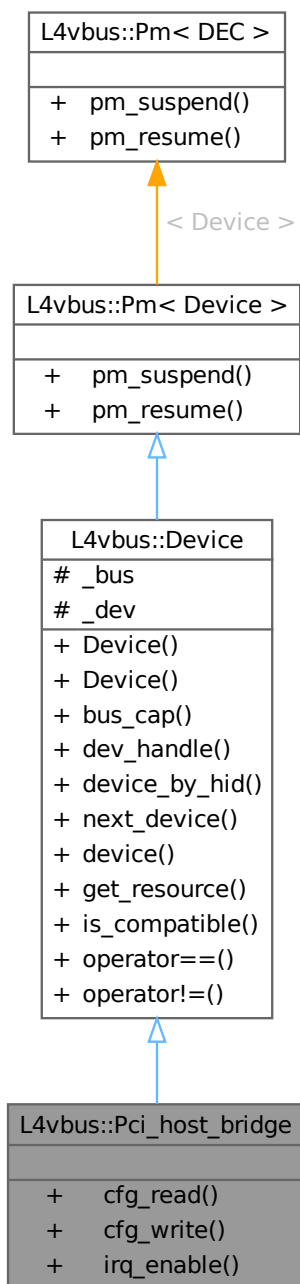
- `l4/vbus/vbus_pci`

16.384 L4vbus::Pci_host_bridge Class Reference

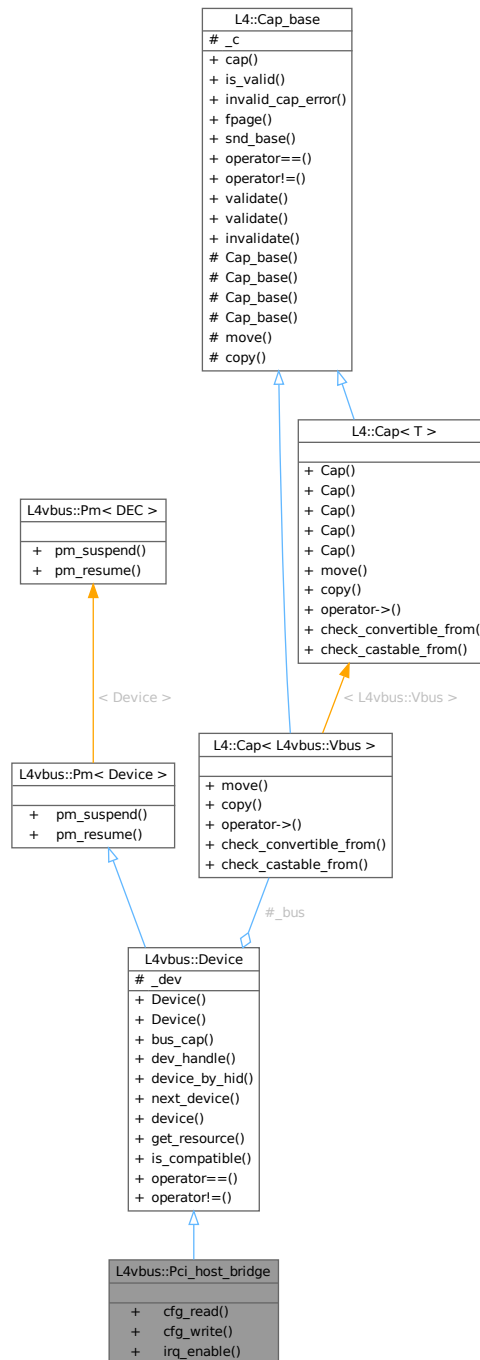
A Pci host bridge.

```
#include <vbus_pci>
```


Inheritance diagram for L4vbus::Pci_host_bridge:



Collaboration diagram for L4vbus::Pci_host_bridge:



Public Member Functions

- `int cfg_read(l4_uint32_t bus, l4_uint32_t devfn, l4_uint32_t reg, l4_uint32_t *value, l4_uint32_t width) const`
Read from the vPCI configuration space using the PCI root bridge.
- `int cfg_write(l4_uint32_t bus, l4_uint32_t devfn, l4_uint32_t reg, l4_uint32_t value, l4_uint32_t width) const`
Write to the vPCI configuration space using the PCI root bridge.

- int [irq_enable](#) ([l4_uint32_t](#) bus, [l4_uint32_t](#) devfn, int pin, unsigned char *trigger, unsigned char *polarity) const

Enable PCI interrupt for a specific device using the PCI root bridge.

Public Member Functions inherited from [L4vbus::Device](#)

- [Device](#) ()
Construct a new vbus device using the NULL device [L4VBUS_NULL](#).
- [Device](#) ([L4::Cap](#)< [Vbus](#) > bus, [l4vbus_device_handle_t](#) dev)
Construct a new vbus device using a device handle.
- [L4::Cap](#)< [Vbus](#) > [bus_cap](#) () const
Access the [Vbus](#) capability of the underlying virtual bus.
- [l4vbus_device_handle_t](#) [dev_handle](#) () const
Access the device handle of this device.
- int [device_by_hid](#) ([Device](#) *child, char const *hid, int depth=[L4VBUS_MAX_DEPTH](#), [l4vbus_device_t](#) *devinfo=0) const
Find a device by the hardware interface identifier (HID).
- int [next_device](#) ([Device](#) *child, int depth=[L4VBUS_MAX_DEPTH](#), [l4vbus_device_t](#) *devinfo=0) const
*Find next child following *child*.*
- int [device](#) ([l4vbus_device_t](#) *devinfo) const
Obtain detailed information about a [Vbus](#) device.
- int [get_resource](#) (unsigned res_idx, [l4vbus_resource_t](#) *res) const
Obtain the resource description of an individual device resource.
- int [is_compatible](#) (char const *cid) const
Check if the given device has a compatibility ID (CID) or HID that matches cid.
- bool [operator==](#) ([Device](#) const &o) const
Test if two devices are the same [Vbus](#) device.
- bool [operator!=](#) ([Device](#) const &o) const
Test if two [Vbus](#) devices are not the same.

Public Member Functions inherited from [L4vbus::Pm](#)< [Device](#) >

- int [pm_suspend](#) () const
Suspend the device.
- int [pm_resume](#) () const
Resume the device.

Additional Inherited Members

Protected Attributes inherited from [L4vbus::Device](#)

- [L4::Cap](#)< [Vbus](#) > [_bus](#)
The [Vbus](#) capability where this device is located on.
- [l4vbus_device_handle_t](#) [_dev](#)
The device handle for this device.

16.384.1 Detailed Description

A Pci host bridge.

Definition at line 25 of file [vbus_pci](#).

16.384.2 Member Function Documentation

16.384.2.1 `cfg_read()`

```
int L4vbus::Pci_host_bridge::cfg_read (
    l4_uint32_t bus,
    l4_uint32_t devfn,
    l4_uint32_t reg,
    l4_uint32_t * value,
    l4_uint32_t width) const [inline]
```

Read from the vPCI configuration space using the PCI root bridge.

Parameters

	<i>bus</i>	Bus number
	<i>devfn</i>	Device id (upper 16bit) and function (lower 16bit)
	<i>reg</i>	Register in configuration space to read
out	<i>value</i>	Value that has been read
	<i>width</i>	Width to read in bits (e.g. 8, 16, 32)

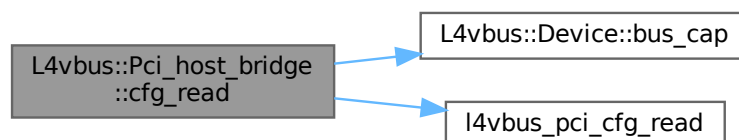
Returns

0 on success, else failure

Definition at line 39 of file [vbus_pci](#).

References [L4vbus::Device::_dev](#), [L4vbus::Device::bus_cap\(\)](#), and [l4vbus_pci_cfg_read\(\)](#).

Here is the call graph for this function:



16.384.2.2 `cfg_write()`

```
int L4vbus::Pci_host_bridge::cfg_write (
    14_uint32_t bus,
    14_uint32_t devfn,
    14_uint32_t reg,
    14_uint32_t value,
    14_uint32_t width) const [inline]
```

Write to the vPCI configuration space using the PCI root bridge.

Parameters

<i>bus</i>	Bus number
<i>devfn</i>	Device id (upper 16bit) and function (lower 16bit)
<i>reg</i>	Register in configuration space to write
<i>value</i>	Value to write
<i>width</i>	Width to write in bits (e.g. 8, 16, 32)

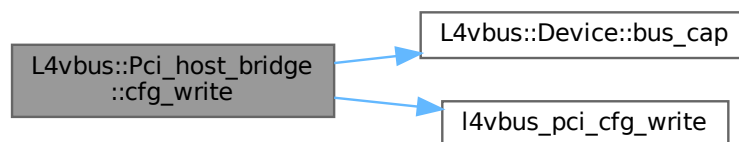
Returns

0 on success, else failure

Definition at line 58 of file [vbus_pci](#).

References [L4vbus::Device::_dev](#), [L4vbus::Device::bus_cap\(\)](#), and [l4vbus_pci_cfg_write\(\)](#).

Here is the call graph for this function:



16.384.2.3 `irq_enable()`

```
int L4vbus::Pci_host_bridge::irq_enable (
    14_uint32_t bus,
    14_uint32_t devfn,
    int pin,
    unsigned char * trigger,
    unsigned char * polarity) const [inline]
```

Enable PCI interrupt for a specific device using the PCI root bridge.

Parameters

	<i>bus</i>	Bus number
	<i>devfn</i>	Device id (upper 16bit) and function (lower 16bit)
	<i>pin</i>	Interrupt pin (normally as reported in configuration register INTR)
out	<i>trigger</i>	False if interrupt is level-triggered
out	<i>polarity</i>	True if interrupt is of low polarity

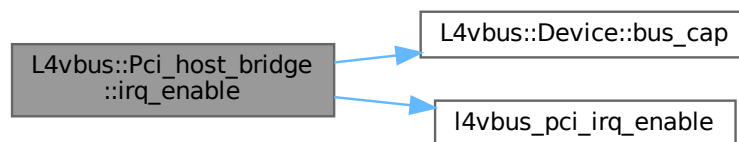
Returns

On success: Interrupt line to be used, else failure

Definition at line 79 of file [vbus_pci](#).

References [L4vbus::Device::_dev](#), [L4vbus::Device::bus_cap\(\)](#), and [l4vbus_pci_irq_enable\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

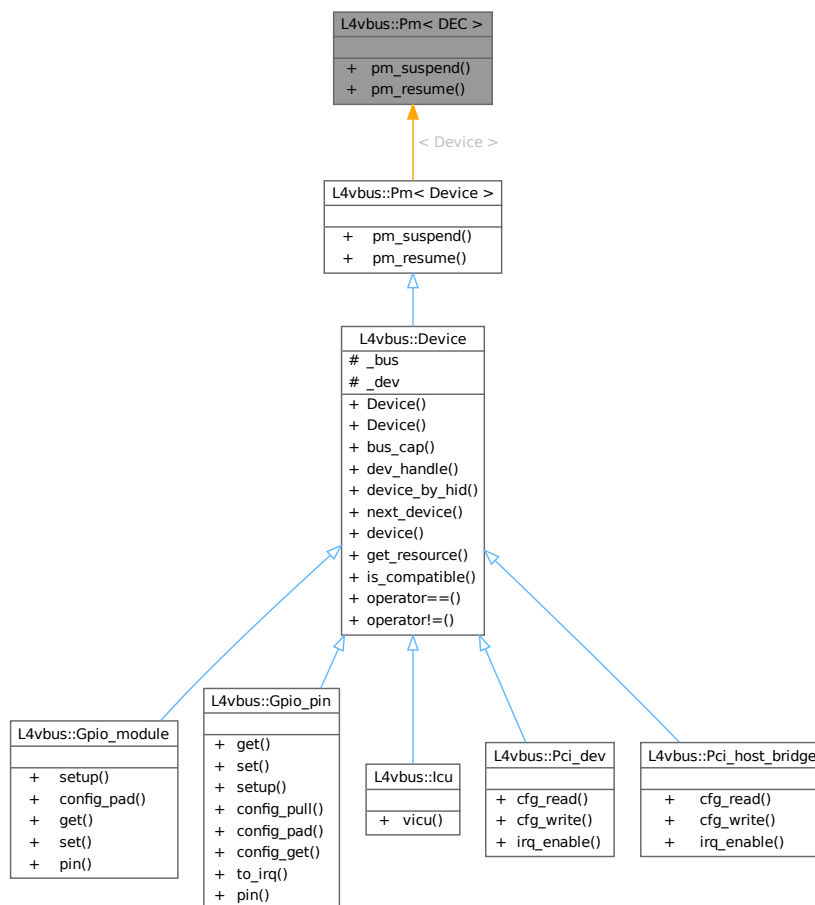
- `l4/vbus/vbus_pci`

16.385 L4vbus::Pm< DEC > Class Template Reference

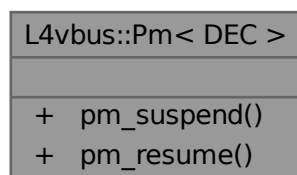
Power-management API mixin.

```
#include <vbus>
```

Inheritance diagram for L4vbus::Pm< DEC >:



Collaboration diagram for L4vbus::Pm< DEC >:



Public Member Functions

- int [pm_suspend](#) () const
Suspend the device.
- int [pm_resume](#) () const
Resume the device.

16.385.1 Detailed Description

```
template<typename DEC>
class L4vbus::Pm< DEC >
```

Power-management API mixin.

Devices that inherit from this mixin provide an API to be suspended and resumed.

Definition at line 50 of file [vbus](#).

16.385.2 Member Function Documentation

16.385.2.1 pm_resume()

```
template<typename DEC>
int L4vbus::Pm< DEC >::pm_resume () const [inline]
```

Resume the device.

Switches the device from low-power mode to normal operation and restores the saved state.

Return values

0	Success.
---	----------

Definition at line 74 of file [vbus](#).

References [l4vbus_pm_resume\(\)](#).

Here is the call graph for this function:



16.385.2.2 pm_suspend()

```
template<typename DEC>
int L4vbus::Pm< DEC >::pm_suspend () const [inline]
```

Suspend the device.

Saves the state of the device and puts it into a low-power mode.

Return values

0	Success.
---	----------

Definition at line 63 of file [vbus](#).

References [l4vbus_pm_suspend\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

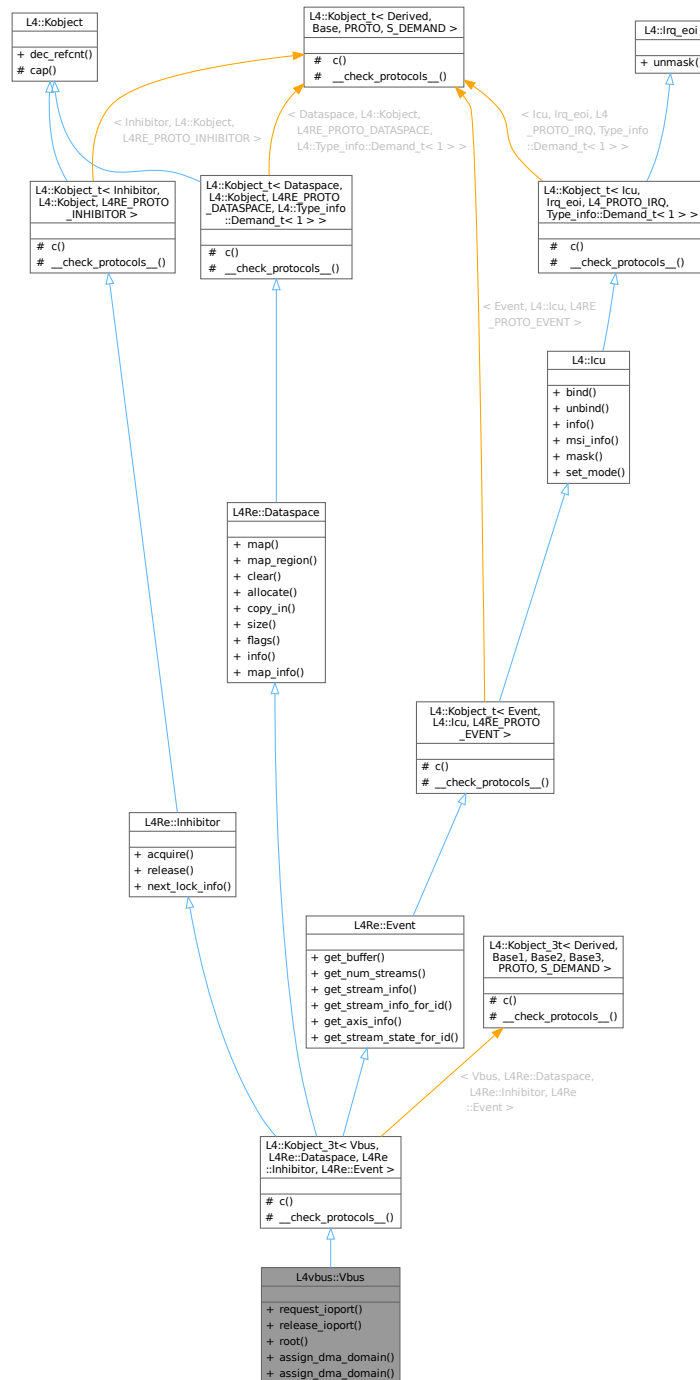
- `I4/vbus/vbus`

16.386 L4vbus::Vbus Class Reference

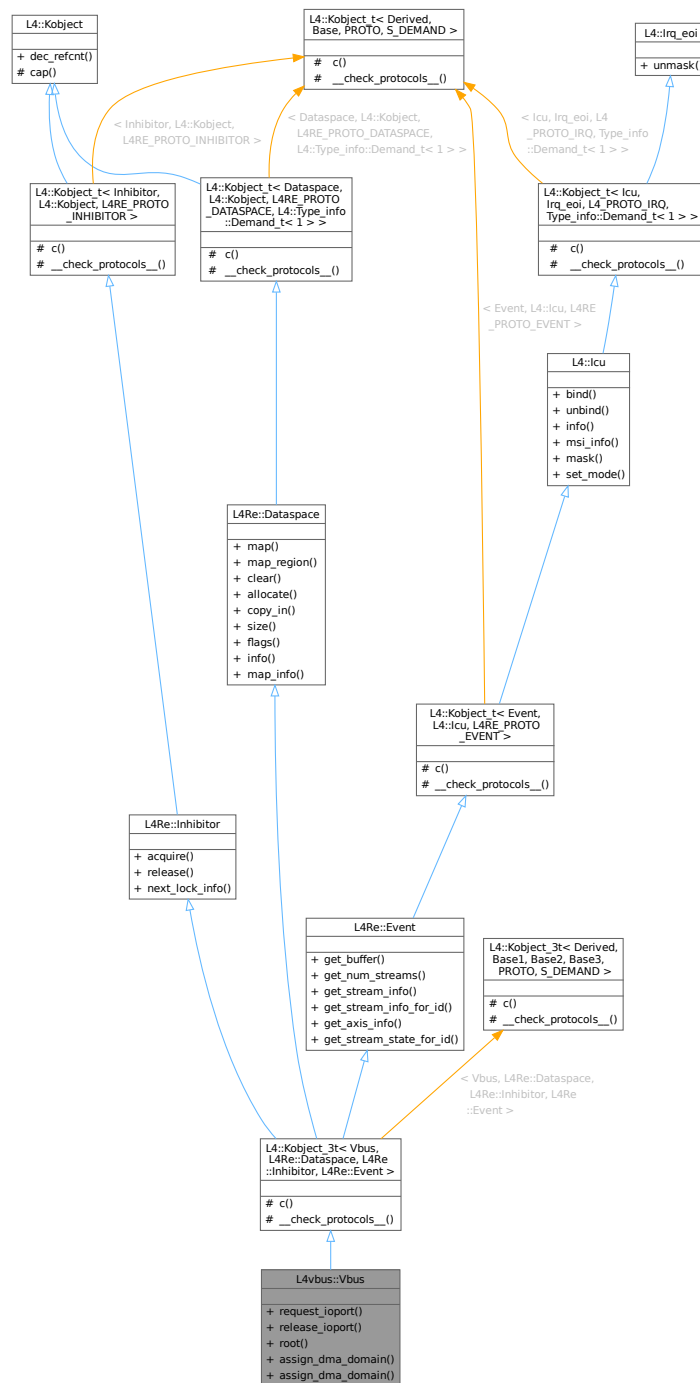
The virtual bus ([Vbus](#)) interface.

```
#include <vbus>
```

Inheritance diagram for L4vbus::Vbus:



Collaboration diagram for L4vbus::Vbus:



Public Member Functions

- `int request_ioport (l4vbus_resource_t *res) const`
Request the given IO port resource from the bus.
- `int release_ioport (l4vbus_resource_t *res) const`
Release the given IO port resource from the bus.
- `Device root () const`

Get the root device of the device tree of this bus.

- int [assign_dma_domain](#) (unsigned domain_id, unsigned flags, [L4::Cap](#)< [L4Re::Dma_space](#) > dma_space) const

Bind or unbind an [L4Re::Dma_space](#) to a DMA domain.

- int [assign_dma_domain](#) (unsigned domain_id, unsigned flags, [L4::Cap](#)< [L4::Task](#) > dma_space) const

Bind or unbind a kernel [DMA space](#) to a DMA domain.

Public Member Functions inherited from [L4Re::Dataspace](#)

- long [map](#) (Offset offset, Flags flags, Map_addr local_addr, Map_addr min_addr, Map_addr max_addr, [L4::Cap](#)< [L4::Task](#) > dst=[L4::Cap](#)< [L4::Task](#) >::Invalid) const noexcept

Request a flexpage mapping from the dataspace.

- long [map_region](#) (Offset offset, Flags flags, Map_addr min_addr, Map_addr max_addr, [L4::Cap](#)< [L4::Task](#) > dst=[L4::Cap](#)< [L4::Task](#) >::Invalid) const noexcept

Map a part of a dataspace into a local memory area.

- long [clear](#) (Offset offset, Size size)

Clear parts of a dataspace.

- long [allocate](#) (Offset offset, Size size)

Allocate a range in the dataspace.

- long [copy_in](#) (Offset dst_offs, [L4::lpc::Cap](#)< [Dataspace](#) > src, Offset src_offs, Size size)

Copy contents from another dataspace.

- Size [size](#) () const noexcept

Get size of a dataspace.

- Flags [flags](#) () const noexcept

Get flags of the dataspace.

- long [info](#) ([Stats](#) *stats)

Get information on the dataspace.

- long [map_info](#) ([l4_addr_t](#) *start_addr, [l4_addr_t](#) *end_addr)

Get mapping range of dataspace.

Public Member Functions inherited from [L4::Kobject](#)

- [l4_msgtag_t](#) [dec_refcnt](#) ([l4_mword_t](#) diff, [l4_utcb_t](#) *utcb=[l4_utcb](#)())

Decrement the in kernel reference counter for the object.

Public Member Functions inherited from [L4Re::Inhibitor](#)

- long [acquire](#) ([l4_umword_t](#) id, [L4::lpc::String](#)<> reason)

Acquire a specific inhibitor lock.

- long [release](#) ([l4_umword_t](#) id)

Release a specific inhibitor lock.

- long [next_lock_info](#) (char *name, unsigned len, [l4_mword_t](#) current_id=-1, [l4_utcb_t](#) *utcb=[l4_utcb](#)())

Get information for the next available inhibitor lock.

Public Member Functions inherited from L4Re::Event

- long [get_buffer](#) (L4::lpc::Out< L4::Cap< Dataspace > > ds)
Get event signal buffer.
- long [get_num_streams](#) ()
Get number of event streams.
- long [get_stream_info](#) (int idx, Event_stream_info *info)
Get event stream infos.
- long [get_stream_info_for_id](#) (l4_umword_t stream_id, Event_stream_info *info)
Get event stream infos.
- long [get_axis_info](#) (l4_umword_t stream_id, unsigned naxes, unsigned const *axis, Event_absinfo *info) const noexcept
Get event stream axis infos.
- long [get_stream_state_for_id](#) (l4_umword_t stream_id, Event_stream_state *state)
Get event stream state.

Public Member Functions inherited from L4::Icu

- [l4_msgtag_t bind](#) (unsigned irqnum, L4::Cap< Triggerable > irq, l4_utcb_t *utcb=l4_utcb()) noexcept
Bind an interrupt line of an interrupt controller to an interrupt object.
- [l4_msgtag_t unbind](#) (unsigned irqnum, L4::Cap< Triggerable > irq, l4_utcb_t *utcb=l4_utcb()) noexcept
Remove binding of an interrupt line from the interrupt controller object.
- [l4_msgtag_t info](#) (l4_icu_info_t *info, l4_utcb_t *utcb=l4_utcb()) noexcept
Get information about the ICU features.
- [l4_msgtag_t msi_info](#) (l4_umword_t irqnum, l4_uint64_t source, l4_icu_msi_info_t *msi_info)
Get MSI info about IRQ.
- [l4_msgtag_t mask](#) (unsigned irqnum, l4_umword_t *label=0, l4_timeout_t to=L4_IPC_NEVER, l4_utcb_t *utcb=l4_utcb()) noexcept
Mask an IRQ line.
- [l4_msgtag_t set_mode](#) (unsigned irqnum, l4_umword_t mode, l4_utcb_t *utcb=l4_utcb()) noexcept
Set interrupt mode.

Public Member Functions inherited from L4::Irq_eoi

- [l4_msgtag_t unmask](#) (unsigned irqnum, l4_umword_t *label=0, l4_timeout_t to=L4_IPC_NEVER, l4_utcb_t *utcb=l4_utcb()) noexcept
Unmask the given interrupt line.

Additional Inherited Members

Public Types inherited from L4Re::Inhibitor

- enum { [Name_max](#) = 20 }

Protected Types inherited from

[L4::Kobject_3t< Vbus, L4Re::Dataspace, L4Re::Inhibitor, L4Re::Event >](#)

- typedef Vbus **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< [PROTO_ANY](#), Vbus > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__Iface** >, Typeid::Merge_list< typename L4Re::↵
 Dataspace::__Iface_list, Typeid::Merge_list< typename L4Re::Inhibitor::__Iface_list, typename L4Re::↵
 Event::__Iface_list > > > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Types inherited from

[L4::Kobject_t< Dataspace, L4::Kobject, L4RE_PROTO_DATASPACE, L4::Type_info::Demand_t< 1 > >](#)

- typedef Dataspace **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< [PROTO](#), Dataspace > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__Iface** >, typename L4::Kobject::__Iface_list > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Types inherited from

[L4::Kobject_t< Inhibitor, L4::Kobject, L4RE_PROTO_INHIBITOR >](#)

- typedef Inhibitor **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< [PROTO](#), Inhibitor > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__Iface** >, typename L4::Kobject::__Iface_list > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Types inherited from [L4::Kobject_t< Event, L4::lcu, L4RE_PROTO_EVENT >](#)

- typedef Event **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< [PROTO](#), Event > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__Iface** >, typename L4::lcu::__Iface_list > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Types inherited from

[L4::Kobject_t< lcu, Irq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > >](#)

- typedef lcu **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< [PROTO](#), lcu > **__Iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__Iface** >, typename Irq_eoi::__Iface_list > **__Iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from**L4::Kobject_3t< Vbus, L4Re::Dataspace, L4Re::Inhibitor, L4Re::Event >**

- **L4::Cap< Class > c ()** const noexcept

*Get the capability to ourselves.***Protected Member Functions inherited from****L4::Kobject_t< Dataspace, L4::Kobject, L4RE_PROTO_DATASPACE, L4::Type_info::Demand_t< 1 > >**

- **L4::Cap< Class > c ()** const noexcept

*Get the capability to ourselves.***Protected Member Functions inherited from L4::Kobject**

- **l4_cap_idx_t cap ()** const noexcept

*Return capability selector.***Protected Member Functions inherited from****L4::Kobject_t< Inhibitor, L4::Kobject, L4RE_PROTO_INHIBITOR >**

- **L4::Cap< Class > c ()** const noexcept

*Get the capability to ourselves.***Protected Member Functions inherited from****L4::Kobject_t< Event, L4::lcu, L4RE_PROTO_EVENT >**

- **L4::Cap< Class > c ()** const noexcept

*Get the capability to ourselves.***Protected Member Functions inherited from****L4::Kobject_t< lcu, Irq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > >**

- **L4::Cap< Class > c ()** const noexcept

*Get the capability to ourselves.***Static Protected Member Functions inherited from****L4::Kobject_3t< Vbus, L4Re::Dataspace, L4Re::Inhibitor, L4Re::Event >**

- static void **__check_protocols__ ()** noexcept

*Helper to check for protocol conflicts.***Static Protected Member Functions inherited from****L4::Kobject_t< Dataspace, L4::Kobject, L4RE_PROTO_DATASPACE, L4::Type_info::Demand_t< 1 > >**

- static void **__check_protocols__ ()** noexcept

Helper to check for protocol conflicts.

Static Protected Member Functions inherited from**[L4::Kobject_t< Inhibitor, L4::Kobject, L4RE_PROTO_INHIBITOR >](#)**

- static void `__check_protocols__()` noexcept

*Helper to check for protocol conflicts.***Static Protected Member Functions inherited from****[L4::Kobject_t< Event, L4::lcu, L4RE_PROTO_EVENT >](#)**

- static void `__check_protocols__()` noexcept

*Helper to check for protocol conflicts.***Static Protected Member Functions inherited from****[L4::Kobject_t< lcu, Irq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > >](#)**

- static void `__check_protocols__()` noexcept

*Helper to check for protocol conflicts.***16.386.1 Detailed Description**

The virtual bus ([Vbus](#)) interface.

See also

[L4Re Vbus API](#)

Definition at line [298](#) of file [vbus](#).

16.386.2 Member Function Documentation**16.386.2.1 `assign_dma_domain()` [1/2]**

```
int L4vbus::Vbus::assign_dma_domain (
    unsigned domain_id,
    unsigned flags,
    L4::Cap< L4::Task > dma_space) const [inline]
```

Bind or unbind a kernel [DMA space](#) to a DMA domain.

Parameters

<i>domain_id</i>	DMA domain ID (resource address of DMA domain found on the vBUS). If the value is ~0U the DMA space of the whole vBUS is used.
<i>flags</i>	A combination of L4vbus_dma_domain_assign_flags .
<i>dma_space</i>	The DMA space capability to bind or unbind, this must be a kernel DMA space (L4::Task created with <code>L4_PROTO_DMA_SPACE</code>)

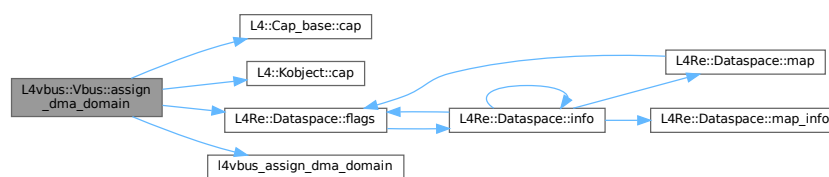
Return values

0	Operation completed successfully.
-L4_ENOENT	The vbus does not support a global DMA domain or no DMA domain could be found.
-L4_EINVAL	Invalid argument used.
-L4_EBUSY	DMA domain is already active, this means another DMA space is already assigned.

Definition at line 382 of file [vbus](#).

References [L4::Cap_base::cap\(\)](#), [L4::Kobject::cap\(\)](#), [L4Re::Dataspace::flags\(\)](#), [l4vbus_assign_dma_domain\(\)](#), [L4VBUS_DMAD_KERNEL_DMA_SPACE](#), and [L4VBUS_DMAD_L4RE_DMA_SPACE](#).

Here is the call graph for this function:



16.386.2.2 assign_dma_domain() [2/2]

```

int L4vbus::Vbus::assign_dma_domain (
    unsigned domain_id,
    unsigned flags,
    L4::Cap< L4Re::Dma_space > dma_space) const [inline]

```

Bind or unbind an [L4Re::Dma_space](#) to a DMA domain.

Parameters

<i>domain_id</i>	DMA domain ID (resource address of DMA domain found on the vBUS). If the value is ~0U the DMA space of the whole vBUS is used.
<i>flags</i>	A combination of L4vbus_dma_domain_assign_flags .
<i>dma_space</i>	The DMA space capability to bind or unbind, this must be an L4Re::Dma_space

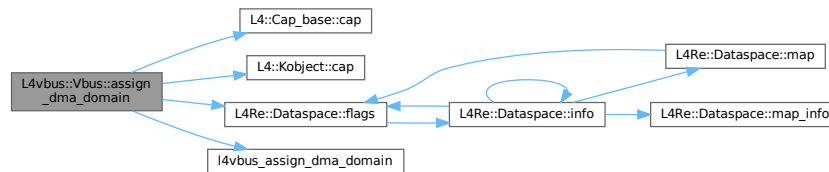
Return values

0	Operation completed successfully.
-L4_ENOENT	The vbus does not support a global DMA domain or no DMA domain could be found.
-L4_EINVAL	Invalid argument used.
-L4_EBUSY	DMA domain is already active, this means another DMA space is already assigned.

Definition at line 357 of file [vbus](#).

References [L4::Cap_base::cap\(\)](#), [L4::Kobject::cap\(\)](#), [L4Re::Dataspace::flags\(\)](#), [l4vbus_assign_dma_domain\(\)](#), [L4VBUS_DMAD_KERNEL_DMA_SPACE](#), and [L4VBUS_DMAD_L4RE_DMA_SPACE](#).

Here is the call graph for this function:



16.386.2.3 release_ioport()

```
int L4vbus::Vbus::release_ioport (
    l4vbus_resource_t * res) const [inline]
```

Release the given IO port resource from the bus.

Parameters

in	res	The IO port resource to be released from the bus.
----	-----	---

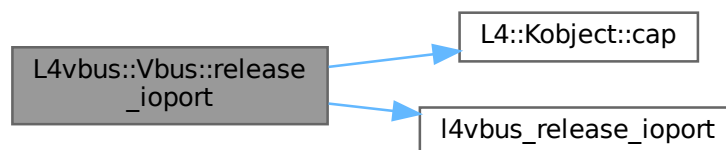
Returns

>=0 on success, <0 on error.

Definition at line 323 of file [vbus](#).

References [L4::Kobject::cap\(\)](#), and [l4vbus_release_ioport\(\)](#).

Here is the call graph for this function:



16.386.2.4 request_ioport()

```
int L4vbus::Vbus::request_ioport (
    l4vbus_resource_t * res) const [inline]
```

Request the given IO port resource from the bus.

Parameters

<i>in</i>	<i>res</i>	The IO port resource to be requested from the bus.
-----------	------------	--

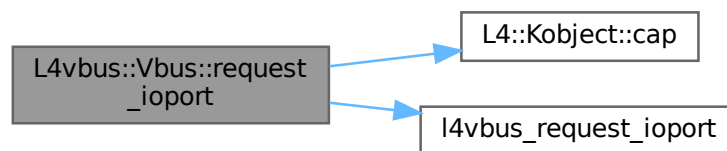
Return values

<i>0</i>	Success.
<i>-L4_EINVAL</i>	Resource is not an IO port resource.
<i>-L4_ENOENT</i>	No matching IO port resource found.

Definition at line 311 of file [vbus](#).

References [L4::Kobject::cap\(\)](#), and [l4vbus_request_ioport\(\)](#).

Here is the call graph for this function:

**16.386.2.5 root()**

```
Device L4vbus::Vbus::root () const [inline]
```

Get the root device of the device tree of this bus.

The root device is usually the starting point for iterating the bus, see [Device::next_device](#).

Returns

A [Vbus](#) device representing the root of the device tree.

Definition at line 336 of file [vbus](#).

References [L4::Kobject::cap\(\)](#), and [L4VBUS_ROOT_BUS](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

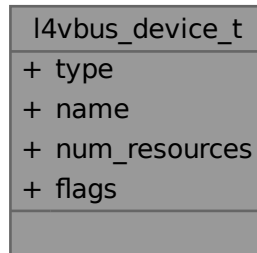
- `l4/vbus/vbus`

16.387 l4vbus_device_t Struct Reference

Detailed information about a vbus device.

```
#include <vbus_types.h>
```

Collaboration diagram for l4vbus_device_t:



Data Fields

- [l4_uint32_t](#) **type**
Bitfield of supported sub-interfaces, see [l4vbus_iface_type_t](#).
- char **name** [L4VBUS_DEV_NAME_LEN]
Name.
- unsigned **num_resources**
Number of resources for this device.
- unsigned **flags**
Flags, see [l4vbus_device_flags_t](#).

16.387.1 Detailed Description

Detailed information about a vbus device.

Definition at line 80 of file [vbus_types.h](#).

The documentation for this struct was generated from the following file:

- [l4/vbus/vbus_types.h](#)

16.388 l4vbus_resource_t Struct Reference

Description of a single vbus resource.

```
#include <vbus_types.h>
```

Collaboration diagram for l4vbus_resource_t:

l4vbus_resource_t	
+	type
+	flags
+	start
+	end
+	provider
+	id

Data Fields

- [l4_uint16_t](#) **type**
Resource type, see [l4vbus_resource_type_t](#).
- [l4_uint16_t](#) **flags**
Flags.
- [l4vbus_paddr_t](#) **start**
Start of resource range.
- [l4vbus_paddr_t](#) **end**
End of resource range (inclusive).
- [l4vbus_device_handle_t](#) **provider**
Device handle of the provider of the resource.
- [l4_uint32_t](#) **id**
Resource ID (4 bytes), usually a 4 letter ASCII name is used.

16.388.1 Detailed Description

Description of a single vbus resource.

Definition at line 23 of file [vbus_types.h](#).

The documentation for this struct was generated from the following file:

- [l4/vbus/vbus_types.h](#)

16.389 L4vcpu::State Class Reference

C++ implementation of state word in the vCPU area.

```
#include <vcpu>
```

Collaboration diagram for L4vcpu::State:

L4vcpu::State	
+	State()
+	add()
+	clear()
+	set()

Public Member Functions

- [State](#) (unsigned v)
Initialize state.
- void [add](#) (unsigned bits) throw ()
Add flags.
- void [clear](#) (unsigned bits) throw ()
Clear flags.
- void [set](#) (unsigned v) throw ()
Set flags.

16.389.1 Detailed Description

C++ implementation of state word in the vCPU area.

Definition at line [24](#) of file [vcpu](#).

16.389.2 Constructor & Destructor Documentation

16.389.2.1 State()

```
L4vcpu::State::State (  
    unsigned v) [inline], [explicit]
```

Initialize state.

Parameters

<i>v</i>	Initial state.
----------	----------------

Definition at line 34 of file [vcpu](#).

16.389.3 Member Function Documentation

16.389.3.1 add()

```
void L4vcpu::State::add (  
    unsigned bits) throw ( )    [inline]
```

Add flags.

Parameters

<i>bits</i>	Bits to add to the word.
-------------	--------------------------

Definition at line 41 of file [vcpu](#).

16.389.3.2 clear()

```
void L4vcpu::State::clear (  
    unsigned bits) throw ( )    [inline]
```

Clear flags.

Parameters

<i>bits</i>	Bits to clear in the word.
-------------	----------------------------

Definition at line 48 of file [vcpu](#).

16.389.3.3 set()

```
void L4vcpu::State::set (  
    unsigned v) throw ( )    [inline]
```

Set flags.

Parameters

<i>v</i>	Set the word to the value of <i>v</i> .
----------	---

Definition at line 55 of file [vcpu](#).

The documentation for this class was generated from the following file:

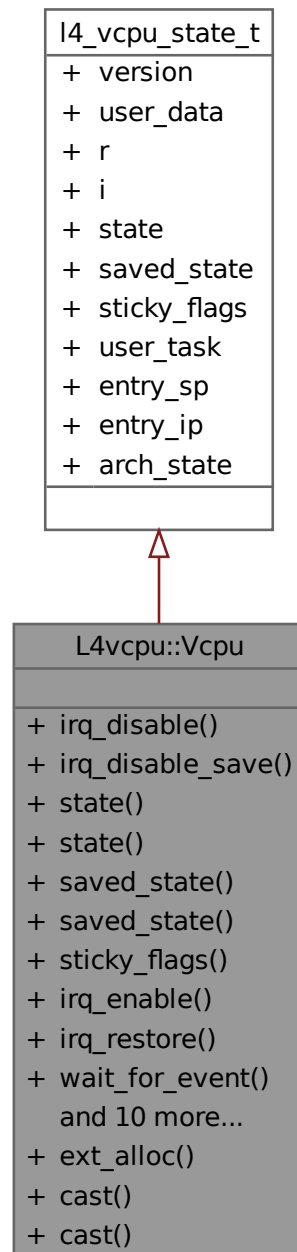
- [I4/vcpu/vcpu](#)

16.390 L4vcpu::Vcpu Class Reference

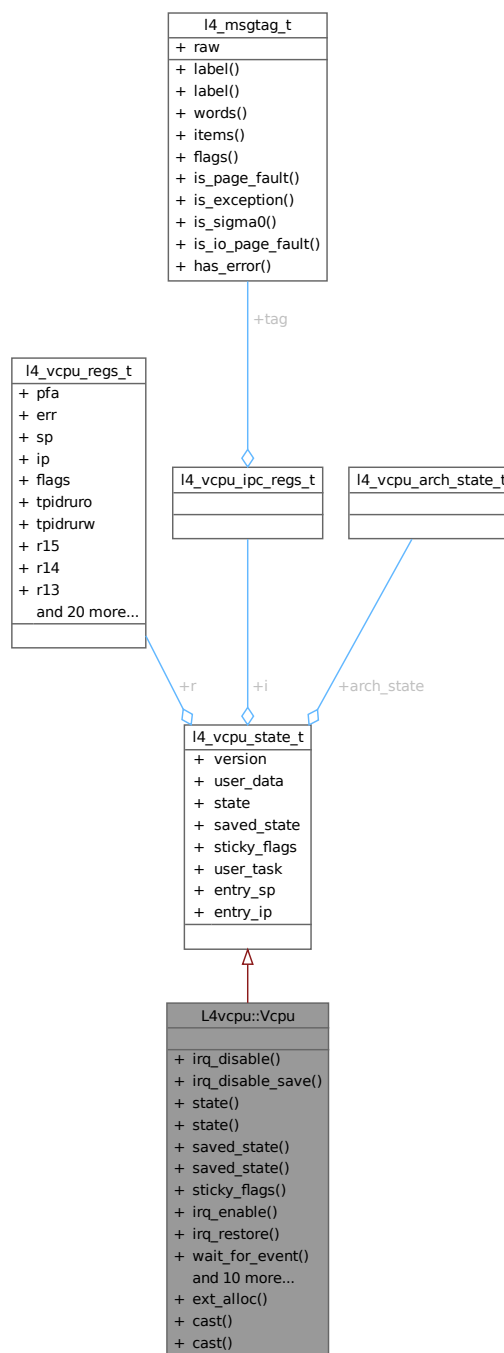
C++ implementation of the vCPU save state area.

```
#include <vcpu>
```

Inheritance diagram for L4vcpu::Vcpu:



Collaboration diagram for L4vcpu::Vcpu:



Public Member Functions

- void **irq_disable** () throw ()
Disable the vCPU for event delivery.
- unsigned **irq_disable_save** () throw ()
Disable the vCPU for event delivery and return previous state.
- **State * state** () throw ()

- Get state word.*

 - `State state () const throw ()`
- Get state word.*

 - `State * saved_state () throw ()`
- Get saved_state word.*

 - `State saved_state () const throw ()`
- Get saved_state word.*

 - `l4_uint16_t sticky_flags () const throw ()`
- Get sticky flags.*

 - `void irq_enable (l4_utcb_t *utcb, l4vcpu_event_hndl_t do_event_work_cb, l4vcpu_setup_ipc_t setup_ipc) throw ()`
- Enable the vCPU for event delivery.*

 - `void irq_restore (unsigned s, l4_utcb_t *utcb, l4vcpu_event_hndl_t do_event_work_cb, l4vcpu_setup_ipc_t setup_ipc) throw ()`
- Restore a previously saved IRQ/event state.*

 - `void wait_for_event (l4_utcb_t *utcb, l4vcpu_event_hndl_t do_event_work_cb, l4vcpu_setup_ipc_t setup_ipc) throw ()`
- Wait for event.*

 - `void task (L4::Cap< L4::Task > const task=L4::Cap< L4::Task >::Invalid) throw ()`
- Set the task of the vCPU.*

 - `int is_page_fault_entry () const`
- Return whether the entry reason was a page fault.*

 - `int is_irq_entry () const`
- Return whether the entry reason was an IRQ/IPC message.*

 - `l4_vcpu_regs_t * r () throw ()`
- Return pointer to register state.*

 - `l4_vcpu_regs_t const * r () const throw ()`
- Return pointer to register state.*

 - `l4_vcpu_ipc_regs_t * i () throw ()`
- Return pointer to IPC state.*

 - `l4_vcpu_ipc_regs_t const * i () const throw ()`
- Return pointer to IPC state.*

 - `void entry_sp (l4_umword_t sp)`
- Set vCPU entry stack pointer.*

 - `void entry_ip (l4_umword_t ip)`
- Set vCPU entry instruction pointer.*

 - `void print_state (const char *prefix="") const throw ()`
- Print the state of the vCPU.*

Static Public Member Functions

- `static int ext_alloc (Vcpu **vcpu, l4_addr_t *ext_state, L4::Cap< L4::Task > task=L4Re::Env::env() ->task(), L4::Cap< L4Re::Rm > rm=L4Re::Env::env() ->rm()) throw ()`

Allocate state area for an extended vCPU.
- `static Vcpu * cast (void *x) throw ()`

Cast a void pointer to a class pointer.
- `static Vcpu * cast (l4_addr_t x) throw ()`

Cast an address to a class pointer.

16.390.1 Detailed Description

C++ implementation of the vCPU save state area.

Definition at line 65 of file [vcpu](#).

16.390.2 Member Function Documentation

16.390.2.1 `cast()` [1/2]

```
Vcpu * L4vcpu::Vcpu::cast (
    l4\_addr\_t x) throw ( )    [inline], [static]
```

Cast an address to a class pointer.

Parameters

<i>x</i>	Pointer.
----------	----------

Returns

Pointer to [Vcpu](#) class.

Definition at line 269 of file [vcpu](#).

16.390.2.2 `cast()` [2/2]

```
Vcpu * L4vcpu::Vcpu::cast (
    void * x) throw ( )    [inline], [static]
```

Cast a void pointer to a class pointer.

Parameters

<i>x</i>	Pointer.
----------	----------

Returns

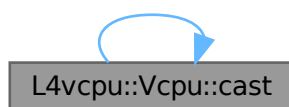
Pointer to [Vcpu](#) class.

Definition at line [259](#) of file [vcpu](#).

References [cast\(\)](#).

Referenced by [cast\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**16.390.2.3 entry_ip()**

```
void L4vcpu::Vcpu::entry_ip (  
    l4\_umword\_t ip) [inline]
```

Set vCPU entry instruction pointer.

Parameters

<i>ip</i>	Instruction pointer address to set.
-----------	-------------------------------------

Definition at line [232](#) of file [vcpu](#).

References [l4_vcpu_state_t::entry_ip](#).

16.390.2.4 entry_sp()

```
void L4vcpu::Vcpu::entry_sp (  
    14_umword_t sp) [inline]
```

Set vCPU entry stack pointer.

Parameters

<i>sp</i>	Stack pointer address to set.
-----------	-------------------------------

Note

The value is only used when entering from a user-task.

Definition at line 225 of file `vcpu`.

References `l4_vcpu_state_t::entry_sp`.

16.390.2.5 ext_alloc()

```
int L4vcpu::Vcpu::ext_alloc (
    Vcpu ** vcpu,
    l4_addr_t * ext_state,
    L4::Cap< L4::Task > task = L4Re::Env::env() ->task(),
    L4::Cap< L4Re::Rm > rm = L4Re::Env::env() ->rm() throw ( )    [static]
```

Allocate state area for an extended vCPU.

Parameters

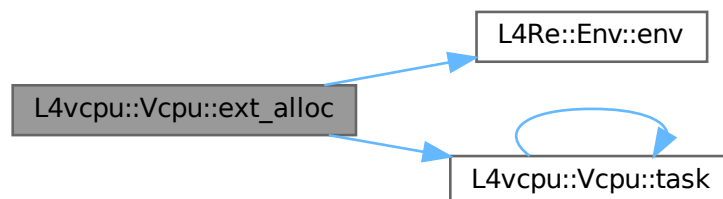
out	<i>vcpu</i>	Allocated vcpu-state area.
out	<i>ext_state</i>	Allocated extended vcpu-state area.
	<i>task</i>	Task to use for allocation, defaults to own task.
	<i>rm</i>	Region manager to use for allocation defaults to standard region manager.

Returns

0 for success, error code otherwise

References `L4Re::Env::env()`, and `task()`.

Here is the call graph for this function:



16.390.2.6 i() [1/2]

```
l4_vcpu_ipc_regs_t * L4vcpu::Vcpu::i () throw ( ) [inline]
```

Return pointer to IPC state.

Returns

Pointer to IPC state.

Definition at line 209 of file [vcpu](#).

References [l4_vcpu_state_t::i](#).

16.390.2.7 i() [2/2]

```
l4_vcpu_ipc_regs_t const * L4vcpu::Vcpu::i () const throw ( ) [inline]
```

Return pointer to IPC state.

Returns

Pointer to IPC state.

Definition at line 216 of file [vcpu](#).

References [l4_vcpu_state_t::i](#).

16.390.2.8 irq_disable_save()

```
unsigned L4vcpu::Vcpu::irq_disable_save () throw ( ) [inline]
```

Disable the vCPU for event delivery and return previous state.

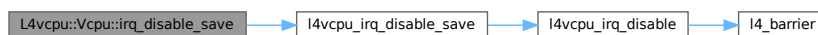
Returns

IRQ state before disabling IRQs.

Definition at line 78 of file [vcpu](#).

References [l4vcpu_irq_disable_save\(\)](#).

Here is the call graph for this function:

**16.390.2.9 irq_enable()**

```
void L4vcpu::Vcpu::irq_enable (
    l4_utcb_t * utcb,
    l4vcpu_event_hndl_t do_event_work_cb,
    l4vcpu_setup_ipc_t setup_ipc) throw ( ) [inline]
```

Enable the vCPU for event delivery.

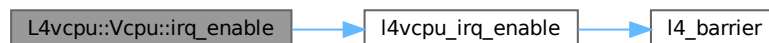
Parameters

<i>utcb</i>	The UTCB to use.
<i>do_event_work_cb</i>	Call-back function that is called in case an event (such as an interrupt) is pending.
<i>setup_ipc</i>	Call-back function that is called before an IPC operation is called, and before event delivery is enabled.

Definition at line 135 of file `vcpu`.

References [l4vcpu_irq_enable\(\)](#).

Here is the call graph for this function:



16.390.2.10 `irq_restore()`

```

void L4vcpu::Vcpu::irq_restore (
    unsigned s,
    l4_utcb_t * utcb,
    l4vcpu_event_hndl_t do_event_work_cb,
    l4vcpu_setup_ipc_t setup_ipc) throw ( )    [inline]
  
```

Restore a previously saved IRQ/event state.

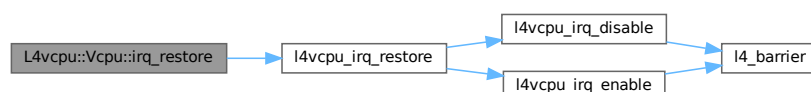
Parameters

<i>s</i>	IRQ state to be restored.
<i>utcb</i>	The UTCB to use.
<i>do_event_work_cb</i>	Call-back function that is called in case an event (such as an interrupt) is pending.
<i>setup_ipc</i>	Call-back function that is called before an IPC operation is called, and before event delivery is enabled.

Definition at line 150 of file `vcpu`.

References [l4vcpu_irq_restore\(\)](#).

Here is the call graph for this function:



16.390.2.11 is_irq_entry()

```
int L4vcpu::Vcpu::is_irq_entry () const [inline]
```

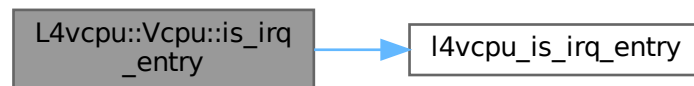
Return whether the entry reason was an IRQ/IPC message.

return 0 if not, !=0 otherwise.

Definition at line 188 of file [vcpu](#).

References [l4vcpu_is_irq_entry\(\)](#).

Here is the call graph for this function:



16.390.2.12 is_page_fault_entry()

```
int L4vcpu::Vcpu::is_page_fault_entry () const [inline]
```

Return whether the entry reason was a page fault.

return 0 if not, !=0 otherwise.

Definition at line 181 of file [vcpu](#).

References [l4vcpu_is_page_fault_entry\(\)](#).

Here is the call graph for this function:



16.390.2.13 `r()` [1/2]

```
l4_vcpu_regs_t * L4vcpu::Vcpu::r () throw ( ) [inline]
```

Return pointer to register state.

Returns

Pointer to register state.

Definition at line 195 of file `vcpu`.

References `l4_vcpu_state_t::r`.

16.390.2.14 `r()` [2/2]

```
l4_vcpu_regs_t const * L4vcpu::Vcpu::r () const throw ( ) [inline]
```

Return pointer to register state.

Returns

Pointer to register state.

Definition at line 202 of file `vcpu`.

References `l4_vcpu_state_t::r`.

16.390.2.15 `saved_state()` [1/2]

```
State * L4vcpu::Vcpu::saved_state () throw ( ) [inline]
```

Get `saved_state` word.

Returns

Pointer to `saved_state` word in the vCPU

Definition at line 106 of file `vcpu`.

References `l4_vcpu_state_t::saved_state`.

16.390.2.16 `saved_state()` [2/2]

```
State L4vcpu::Vcpu::saved_state () const throw ( ) [inline]
```

Get `saved_state` word.

Returns

Pointer to `saved_state` word in the vCPU

Definition at line 116 of file `vcpu`.

References `l4_vcpu_state_t::saved_state`.

16.390.2.17 state() [1/2]

```
State * L4vcpu::Vcpu::state () throw ( )    [inline]
```

Get state word.

Returns

Pointer to state word in the vCPU

Definition at line 88 of file [vcpu](#).

References [l4_vcpu_state_t::state](#).

16.390.2.18 state() [2/2]

```
State L4vcpu::Vcpu::state () const throw ( )    [inline]
```

Get state word.

Returns

Pointer to state word in the vCPU

Definition at line 99 of file [vcpu](#).

References [l4_vcpu_state_t::state](#).

16.390.2.19 task()

```
void L4vcpu::Vcpu::task (
    L4::Cap< L4::Task > const task = L4::Cap<L4::Task>::Invalid) throw ( )    [inline]
```

Set the task of the vCPU.

Parameters

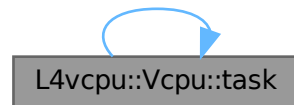
<i>task</i>	Task to set, defaults to invalid task.
-------------	--

Definition at line 174 of file [vcpu](#).

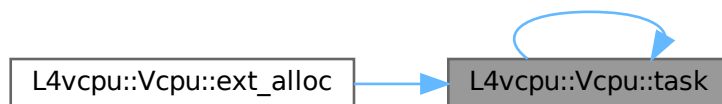
References [L4::Cap_base::Invalid](#), [task\(\)](#), and [l4_vcpu_state_t::user_task](#).

Referenced by [ext_alloc\(\)](#), and [task\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.390.2.20 wait_for_event()

```

void L4vcpu::Vcpu::wait_for_event (
    l4_utcb_t * utcb,
    l4vcpu_event_hndl_t do_event_work_cb,
    l4vcpu_setup_ipc_t setup_ipc) throw ( )    [inline]
  
```

Wait for event.

Parameters

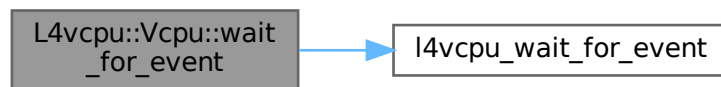
<i>utcb</i>	The UTCB to use.
<i>do_event_work_cb</i>	Call-back function that is called in case an event (such as an interrupt) is pending.
<i>setup_ipc</i>	Call-back function that is called before an IPC operation is called.

Note that event delivery remains disabled after this function returns.

Definition at line 166 of file `vcpu`.

References `l4vcpu_wait_for_event()`.

Here is the call graph for this function:



The documentation for this class was generated from the following file:

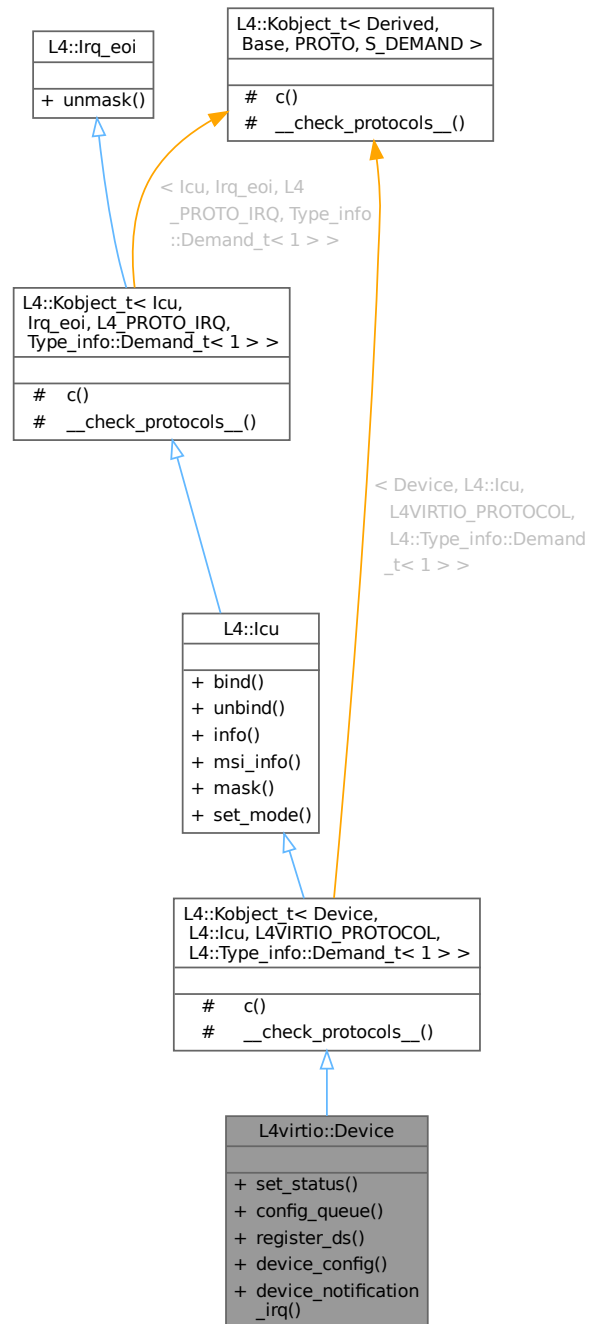
- [l4/vcpu/vcpu](#)

16.391 L4virtio::Device Class Reference

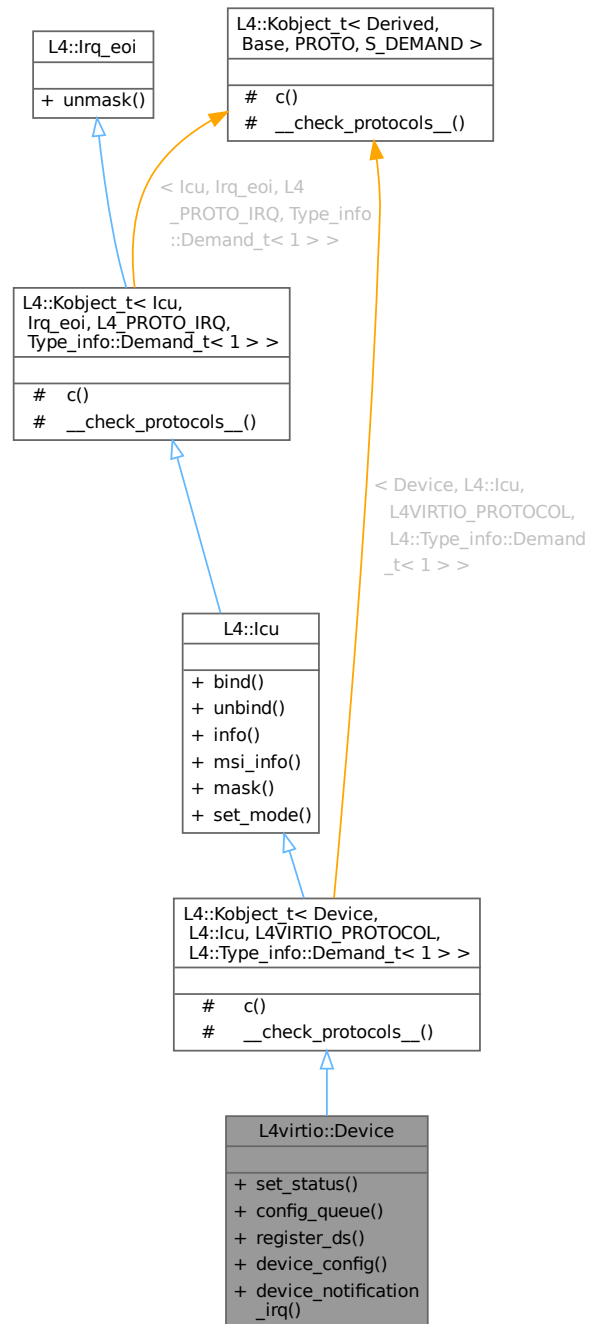
IPC interface for virtio over [L4](#) IPC.

```
#include <l4virtio>
```

Inheritance diagram for L4virtio::Device:



Collaboration diagram for L4virtio::Device:



Public Member Functions

- long `set_status` (unsigned status)
Write the VIRTIO status register.
- long `config_queue` (unsigned queue)
Trigger queue configuration of the given queue.

- long `register_ds` (`L4::lpc::Cap< L4Re::Dataspace >` ds_cap, `l4_uint64_t` base, `l4_umword_t` offset, `l4_umword_t` size)
Register a shared data space with VIRTIO host.
- long `device_config` (`L4::lpc::Out< L4::Cap< L4Re::Dataspace > >` config_ds, `l4_addr_t` *ds_offset)
Get the dataspace with the [L4virtio](#) configuration page.
- long `device_notification_irq` (unsigned index, `L4::lpc::Out< L4::Cap< L4::Triggerable > >` irq)
Get the notification interrupt corresponding to the given index.

Public Member Functions inherited from `L4::lcu`

- `l4_msgtag_t` `bind` (unsigned irqnum, `L4::Cap< Triggerable >` irq, `l4_utcb_t` *utcb=`l4_utcb()`) noexcept
Bind an interrupt line of an interrupt controller to an interrupt object.
- `l4_msgtag_t` `unbind` (unsigned irqnum, `L4::Cap< Triggerable >` irq, `l4_utcb_t` *utcb=`l4_utcb()`) noexcept
Remove binding of an interrupt line from the interrupt controller object.
- `l4_msgtag_t` `info` (`l4_icu_info_t` *info, `l4_utcb_t` *utcb=`l4_utcb()`) noexcept
Get information about the ICU features.
- `l4_msgtag_t` `msi_info` (`l4_umword_t` irqnum, `l4_uint64_t` source, `l4_icu_msi_info_t` *msi_info)
Get MSI info about IRQ.
- `l4_msgtag_t` `mask` (unsigned irqnum, `l4_umword_t` *label=0, `l4_timeout_t` to=`L4_IPC_NEVER`, `l4_utcb_t` *utcb=`l4_utcb()`) noexcept
Mask an IRQ line.
- `l4_msgtag_t` `set_mode` (unsigned irqnum, `l4_umword_t` mode, `l4_utcb_t` *utcb=`l4_utcb()`) noexcept
Set interrupt mode.

Public Member Functions inherited from `L4::lrq_eoi`

- `l4_msgtag_t` `unmask` (unsigned irqnum, `l4_umword_t` *label=0, `l4_timeout_t` to=`L4_IPC_NEVER`, `l4_utcb_t` *utcb=`l4_utcb()`) noexcept
Unmask the given interrupt line.

Additional Inherited Members

Protected Types inherited from

`L4::Kobject_t< Device, L4::lcu, L4VIRTIO_PROTOCOL, L4::Type_info::Demand_t< 1 > >`

- typedef Device **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< PROTO, Device > **__iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__iface** >, typename L4::lcu::__iface_list > **__iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Types inherited from

`L4::Kobject_t< lcu, lrq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > >`

- typedef lcu **Class**
The target interface type (inheriting from [Kobject_t](#)).
- typedef Typeid::Iface< PROTO, lcu > **__iface**
The interface description for the derived class.
- typedef Typeid::Merge_list< Typeid::Iface_list< **__iface** >, typename lrq_eoi::__iface_list > **__iface_list**
The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions inherited from**L4::Kobject_t< Device, L4::lcu, L4VIRTIO_PROTOCOL, L4::Type_info::Demand_t< 1 > >**

- **L4::Cap< Class > c ()** const noexcept
Get the capability to ourselves.

Protected Member Functions inherited from**L4::Kobject_t< lcu, Irq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > >**

- **L4::Cap< Class > c ()** const noexcept
Get the capability to ourselves.

Static Protected Member Functions inherited from**L4::Kobject_t< Device, L4::lcu, L4VIRTIO_PROTOCOL, L4::Type_info::Demand_t< 1 > >**

- static void **__check_protocols__ ()** noexcept
Helper to check for protocol conflicts.

Static Protected Member Functions inherited from**L4::Kobject_t< lcu, Irq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > >**

- static void **__check_protocols__ ()** noexcept
Helper to check for protocol conflicts.

16.391.1 Detailed Description

IPC interface for virtio over [L4](#) IPC.

The [L4virtio](#) protocol is an adaption of the mmio virtio transport 1.0(4). This interface allows to exchange the necessary resources: device configuration page, notification interrupts and dataspace for payload.

Notification interrupts can be configured independently for changes to the configuration space and each queue through special L4virtio-specific `notify_index` fields in the config page and queue configuration. The interface distinguishes between device-to-driver and driver-to-device notification interrupts.

Device-to-driver interrupts are configured via the ICU interface. The device announces the maximum number of supported interrupts via `lcu::info()`. The driver can then bind interrupts using `lcu::bind()`.

Driver-to-device interrupts must be requested from the device through [device_notification_irq\(\)](#).

Definition at line 39 of file [l4virtio](#).

16.391.2 Member Function Documentation**16.391.2.1 config_queue()**

```
long L4virtio::Device::config_queue (
    unsigned queue)
```

Trigger queue configuration of the given queue.

Usually all queues are configured when the status is written to running. However, in some cases queues shall be disabled or enabled dynamically, in this case this function triggers a reconfiguration from the shared memory register of the queue config.

Parameters

<i>queue</i>	Queue index for the queue to be configured.
--------------	---

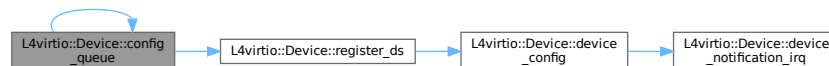
Return values

0	on success.
-L4_EIO	The queue's status is invalid.
-L4_ERANGE	The queue index exceeds the number of queues.
-L4_EINVAL	Otherwise.

References [config_queue\(\)](#), [L4VIRTIO_OP_REGISTER_DS](#), and [register_ds\(\)](#).

Referenced by [config_queue\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.391.2.2 device_config()

```

long L4virtio::Device::device_config (
    L4::Ipc::Out< L4::Cap< L4Re::Dataspace > > config_ds,
    l4_addr_t * ds_offset)

```

Get the dataspace with the [L4virtio](#) configuration page.

Parameters

<i>config_ds</i>	Capability for receiving the dataspace capability for the shared L4-VIRTIO config data space.
------------------	---

<i>ds_offset</i>	Offset into the dataspace where the device configuration structure starts.
------------------	--

References [device_notification_irq\(\)](#), and [L4VIRTIO_OP_GET_DEVICE_IRQ](#).

Referenced by [register_ds\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.391.2.3 device_notification_irq()

```

long L4virtio::Device::device_notification_irq (
    unsigned index,
    L4::Ipc::Out< L4::Cap< L4::Triggerable > > irq)

```

Get the notification interrupt corresponding to the given index.

Parameters

	<i>index</i>	Index of the interrupt.
out	<i>irq</i>	Triggerable for the given index.

Return values

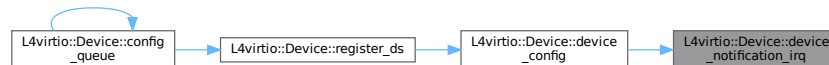
<i>L4_EOK</i>	Success.
<i>L4_ENOSYS</i>	IRQ notification not supported by device.
<0	Other error.

An index is only guaranteed to return an IRQ object when the index is set in one of the device notify index fields. The device must return the same interrupt for a given index as long as the index is in use. If an index disappears as a result of a configuration change and then is reused later, the interrupt is not guaranteed to be the same.

Interrupts must always be rerequested after a device reset.

Referenced by [device_config\(\)](#).

Here is the caller graph for this function:



16.391.2.4 register_ds()

```

long L4virtio::Device::register_ds (
    L4::Ipc::Cap< L4Re::Dataspace > ds_cap,
    l4_uint64_t base,
    l4_umword_t offset,
    l4_umword_t size)

```

Register a shared data space with VIRTIO host.

Parameters

<i>ds_cap</i>	Dataspace capability to register. The lower 8 bits determine the rights mask with which the guest's rights are masked during the registration of the dataspace at the VIRTIO host.
<i>base</i>	VIRTIO guest physical start address of shared memory region
<i>offset</i>	Offset within the data space that is attached to the given <i>base</i> in the guest physical memory.
<i>size</i>	Size of the memory region in the guest

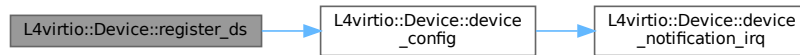
Return values

<i>L4_EOK</i>	Operation successful.
<i>-L4_EINVAL</i>	The <i>ds_cap</i> capability is invalid, does not refer to a valid dataspace, is not a trusted dataspace if trusted dataspace validation is enabled, or <i>size</i> and <i>offset</i> specify an invalid region.
<i>-L4_ENOMEM</i>	The limit of dataspaces that can be registered has been reached or no capability slot could be allocated.
<i>-L4_ERANGE</i>	<i>offset</i> is larger than the size of the dataspace.
<i><0</i>	Any error returned by the dataspace when queried for information during setup or any error returned by the region manager from attaching the dataspace.

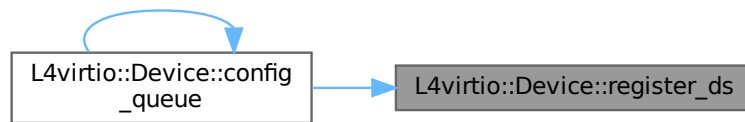
References [device_config\(\)](#), and [L4VIRTIO_OP_DEVICE_CONFIG](#).

Referenced by [config_queue\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.391.2.5 set_status()

```
long L4virtio::Device::set_status (
    unsigned status)
```

Write the VIRTIO status register.

Parameters

<i>status</i>	Status word to write to the VIRTIO status.
---------------	--

Return values

0	on success.
---	-------------

Note

All other registers are accessed via shared memory.

References [L4VIRTIO_OP_CONFIG_QUEUE](#), and [set_status\(\)](#).

Referenced by [set_status\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

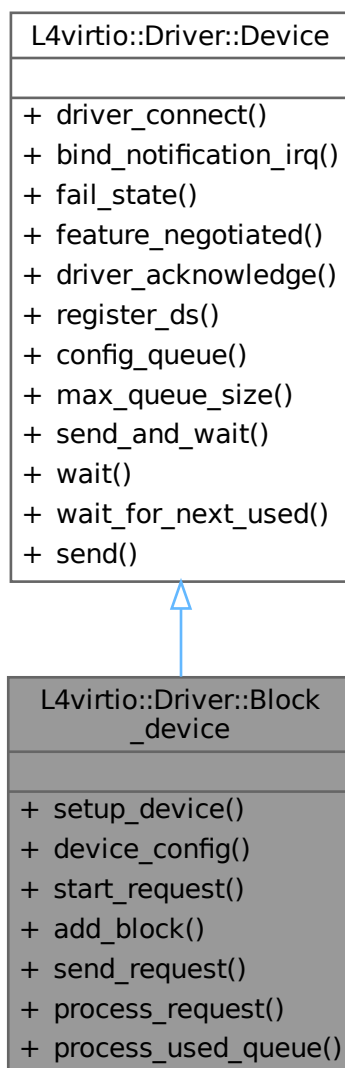
- `I4/I4virtio/I4virtio`

16.392 L4virtio::Driver::Block_device Class Reference

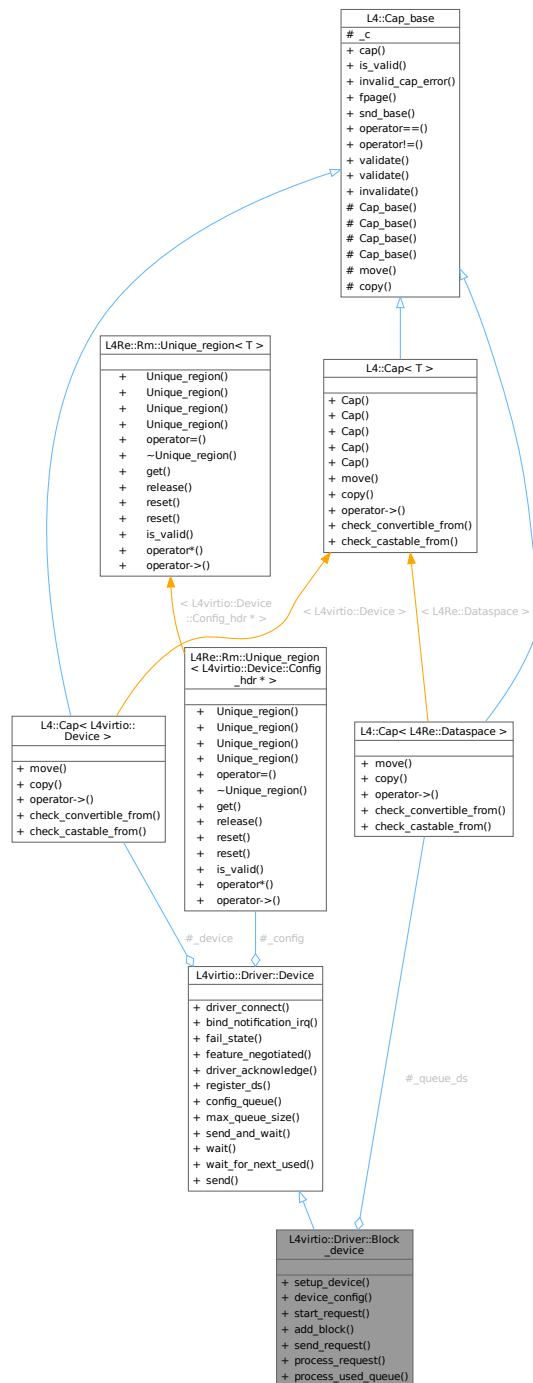
Simple class for accessing a virtio block device synchronously.

```
#include <virtio-block>
```

Inheritance diagram for L4virtio::Driver::Block_device:



Collaboration diagram for L4virtio::Driver::Block_device:



Data Structures

- class `Handle`
Handle to an ongoing request.

Public Member Functions

- void [setup_device](#) ([L4::Cap](#)< [L4virtio::Device](#) > srvcap, [l4_size_t](#) usermem, void **userdata, [Ptr](#)< void > &user_devaddr, [L4::Cap](#)< [L4Re::Dataspace](#) > qds=[L4::Cap](#)< [L4Re::Dataspace](#) >(), [l4_uint32_t](#) fmask0=-1U, [l4_uint32_t](#) fmask1=-1U)
Establish a connection to the device and set up shared memory.
- [l4virtio_block_config_t](#) const & **device_config** () const
Return a reference to the device configuration.
- [Handle](#) start_request ([l4_uint64_t](#) sector, [l4_uint32_t](#) type, Callback callback)
Start the setup of a new request.
- int [add_block](#) ([Handle](#) handle, [Ptr](#)< void > addr, [l4_uint32_t](#) size)
Add a data block to a request that has already been set up.
- int [send_request](#) ([Handle](#) handle)
Process request asynchronously.
- int [process_request](#) ([Handle](#) handle)
Process request synchronously.
- void [process_used_queue](#) ()
Process and free all items in the used queue.

Public Member Functions inherited from [L4virtio::Driver::Device](#)

- void [driver_connect](#) ([L4::Cap](#)< [L4virtio::Device](#) > srvcap, bool manage_notify=true)
Contacts the device and starts the initial handshake.
- int [bind_notification_irq](#) (unsigned index, [L4::Cap](#)< [L4::Triggerable](#) > irq) const
Register a triggerable to receive notifications from the device.
- bool **fail_state** () const
Return true if the device is in a fail state.
- bool [feature_negotiated](#) (unsigned int feat) const
Check if a particular feature bit was negotiated with the device.
- int [driver_acknowledge](#) ()
Finalize handshake with the device.
- int [register_ds](#) ([L4::Cap](#)< [L4Re::Dataspace](#) > ds, [l4_umword_t](#) offset, [l4_umword_t](#) size, [l4_uint64_t](#) *devaddr)
Share a dataspace with the device.
- int [config_queue](#) (int num, unsigned size, [l4_uint64_t](#) desc_addr, [l4_uint64_t](#) avail_addr, [l4_uint64_t](#) used_↔addr)
Send the virtqueue configuration to the device.
- int [max_queue_size](#) (int num) const
Maximum queue size allowed by the device.
- int [send_and_wait](#) ([Virtqueue](#) &queue, [l4_uint16_t](#) descno)
Send a request to the device and wait for it to be processed.
- int [wait](#) (int index) const
Wait for a notification from the device.
- int [wait_for_next_used](#) ([Virtqueue](#) &queue, [l4_uint32_t](#) *len=nullptr) const
Wait for the next item to arrive in the used queue and return it.
- void [send](#) ([Virtqueue](#) &queue, [l4_uint16_t](#) descno)
Send a request to the device.

16.392.1 Detailed Description

Simple class for accessing a virtio block device synchronously.

Definition at line 36 of file [virtio-block](#).

16.392.2 Member Function Documentation

16.392.2.1 add_block()

```
int L4virtio::Driver::Block_device::add_block (
    Handle handle,
    Ptr< void > addr,
    l4_uint32_t size) [inline]
```

Add a data block to a request that has already been set up.

Parameters

<i>handle</i>	Handle to request previously set up with start_request() .
<i>addr</i>	Address of data block in device address space.
<i>size</i>	Size of data block.

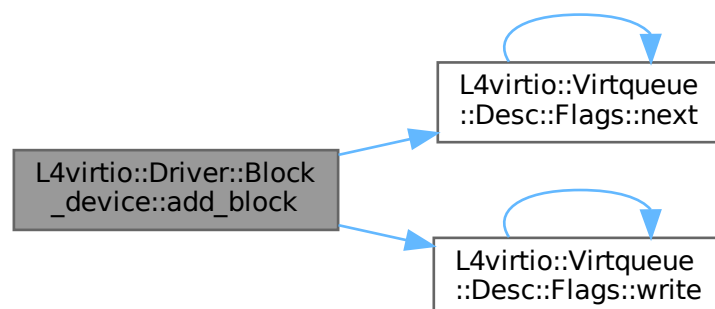
Return values

<i>L4_OK</i>	Block was successfully added.
<i>-L4_EAGAIN</i>	No descriptors available. Try again later.

Definition at line 229 of file [virtio-block](#).

References [L4virtio::Virtqueue::Desc::addr](#), [L4virtio::Virtqueue::Desc::flags](#), [L4_EAGAIN](#), [L4_EOK](#), [L4virtio::Virtqueue::Desc::len](#), [L4virtio::Virtqueue::Desc::Flags::next\(\)](#), [L4virtio::Virtqueue::Desc::next](#), [L4virtio::Virtqueue::Desc::Flags::raw](#), and [L4virtio::Virtqueue::Desc::Flags::write\(\)](#).

Here is the call graph for this function:



16.392.2.2 process_request()

```
int L4virtio::Driver::Block_device::process_request (
    Handle handle) [inline]
```

Process request synchronously.

Parameters

<i>handle</i>	Handle to request to process.
---------------	-------------------------------

Return values

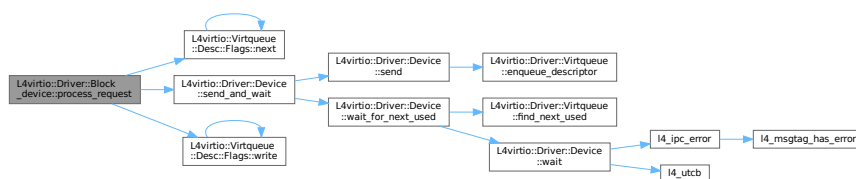
<i>L4_EOK</i>	Request processed successfully.
<i>-L4_EAGAIN</i>	No descriptors available. Try again later.
<i>-L4_EIO</i>	IO error during request processing.
<i>-L4_ENOSYS</i>	Unsupported request.
<i><0</i>	Another unspecified error occurred.

Sends a request to the driver that was previously set up with [start_request\(\)](#) and [add_block\(\)](#) and wait for it to be executed.

Definition at line 306 of file [virtio-block](#).

References [L4virtio::Virtqueue::Desc::addr](#), [L4virtio::Virtqueue::Desc::flags](#), [L4_EAGAIN](#), [L4_EINVAL](#), [L4_EIO](#), [L4_ENOSYS](#), [L4_EOK](#), [L4VIRTIO_BLOCK_S_IOERR](#), [L4VIRTIO_BLOCK_S_OK](#), [L4VIRTIO_BLOCK_S_UNSUPP](#), [L4virtio::Virtqueue::Desc::len](#), [L4virtio::Virtqueue::Desc::Flags::next\(\)](#), [L4virtio::Virtqueue::Desc::next](#), [L4virtio::Virtqueue::Desc::Flags::write\(\)](#), [L4virtio::Driver::Device::send_and_wait\(\)](#), and [L4virtio::Virtqueue::Desc::Flags::write\(\)](#).

Here is the call graph for this function:



16.392.2.3 process_used_queue()

```
void L4virtio::Driver::Block_device::process_used_queue () [inline]
```

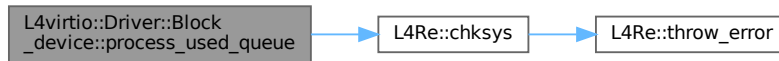
Process and free all items in the used queue.

If the request has a callback registered it is called after the item has been removed from the queue.

Definition at line 357 of file [virtio-block](#).

References [L4Re::chksys\(\)](#), and [L4_ENOSYS](#).

Here is the call graph for this function:



16.392.2.4 send_request()

```
int L4virtio::Driver::Block_device::send_request (
    Handle handle) [inline]
```

Process request asynchronously.

Parameters

<i>handle</i>	Handle to request to send to the device
---------------	---

Return values

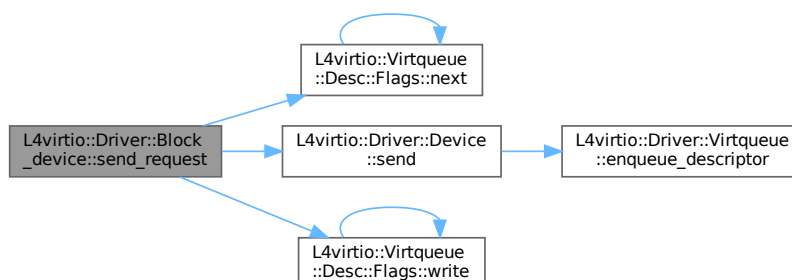
<i>L4_OK</i>	Request was successfully scheduled.
<i>-L4_EAGAIN</i>	No descriptors available. Try again later.

Sends a request to the driver that was previously set up with [start_request\(\)](#) and [add_block\(\)](#) and wait for it to be executed.

Definition at line 265 of file [virtio-block](#).

References [L4virtio::Virtqueue::Desc::addr](#), [L4virtio::Virtqueue::Desc::flags](#), [L4_EAGAIN](#), [L4_EOK](#), [L4virtio::Virtqueue::Desc::len](#), [L4virtio::Virtqueue::Desc::Flags::next\(\)](#), [L4virtio::Virtqueue::Desc::next](#), [L4virtio::Virtqueue::Desc::Flags::raw](#), [L4virtio::Driver::Device::send\(\)](#), and [L4virtio::Virtqueue::Desc::Flags::write\(\)](#).

Here is the call graph for this function:



16.392.2.5 setup_device()

```
void L4virtio::Driver::Block_device::setup_device (
    L4::Cap< L4virtio::Device > srvcap,
    l4_size_t usermem,
    void ** userdata,
    Ptr< void > & user_devaddr,
    L4::Cap< L4Re::Dataspace > qds = L4::Cap<L4Re::Dataspace>(),
    l4_uint32_t fmask0 = -1U,
    l4_uint32_t fmask1 = -1U) [inline]
```

Establish a connection to the device and set up shared memory.

Parameters

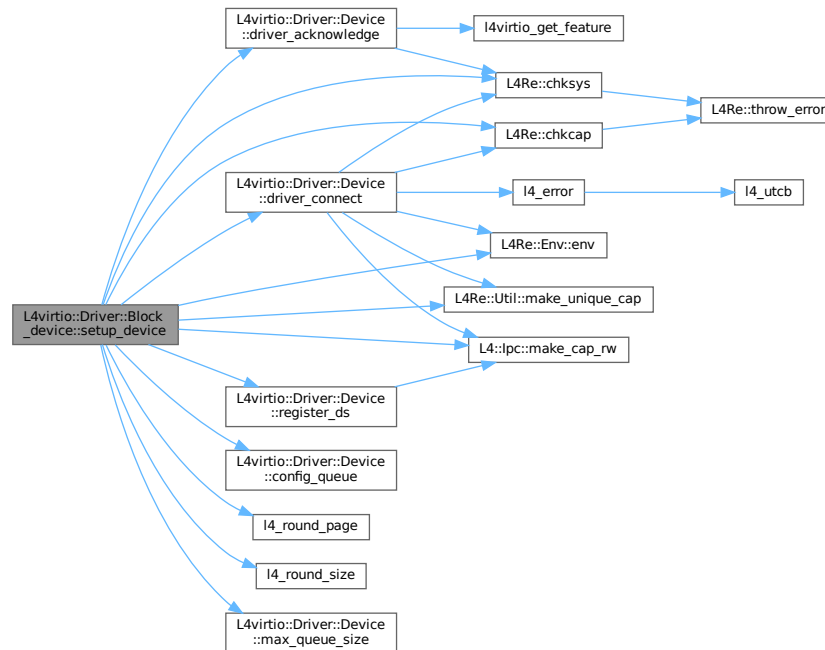
	<i>srvcap</i>	IPC capability of the channel to the server.
	<i>usermem</i>	Size of additional memory to share with device.
out	<i>userdata</i>	Pointer to the region of user-usable memory.
out	<i>user_devaddr</i>	Address of user-usable memory in device address space.
	<i>qds</i>	External queue dataspace. If this capability is invalid, the function will attempt to allocate a dataspace on its own. Note that the external queue dataspace must be large enough.
	<i>fmask0</i>	Feature bits 0..31 that the driver supports.
	<i>fmask1</i>	Feature bits 32..63 that the driver supports.

This function starts a handshake with the device and sets up the virtqueues for communication and the additional data structures for the block device. It will also allocate and share additional memory that the caller then can use freely, i.e. normally this memory would be used as a reception buffer. The caller may also decide to not make use of this convenience function and request 0 bytes in usermem. Then it has to allocate the block buffers for sending/receiving payload manually and share them using [register_ds\(\)](#).

Definition at line 92 of file [virtio-block](#).

References [L4Re::chkcap\(\)](#), [L4Re::chksys\(\)](#), [L4virtio::Driver::Device::config_queue\(\)](#), [L4Re::Mem_alloc::Continuous](#), [L4virtio::Driver::Device::driver_acknowledge\(\)](#), [L4virtio::Driver::Device::driver_connect\(\)](#), [L4Re::Env::env\(\)](#), [L4_EINVAL](#), [L4_ENODEV](#), [L4_PAGESHIFT](#), [l4_round_page\(\)](#), [l4_round_size\(\)](#), [L4VIRTIO_ID_BLOCK](#), [L4::lpc::make_cap_rw\(\)](#), [L4Re::Util::make_unique_cap\(\)](#), [L4virtio::Driver::Device::max_queue_size\(\)](#), [L4Re::Mem_alloc::Pinned](#), [L4virtio::Driver::Device::register_ds\(\)](#), [L4Re::Rm::F::RW](#), and [L4Re::Rm::F::Search_addr](#).

Here is the call graph for this function:



16.392.2.6 start_request()

```

Handle L4virtio::Driver::Block_device::start_request (
    l4_uint64_t sector,
    l4_uint32_t type,
    Callback callback) [inline]

```

Start the setup of a new request.

Parameters

<i>sector</i>	First sector to write to/read from.
<i>type</i>	Request type.
<i>callback</i>	Function to call, when the request is finished. May be 0 for synchronous requests.

Definition at line 191 of file [virtio-block](#).

References [L4virtio::Virtqueue::Desc::addr](#), [L4virtio::Virtqueue::Desc::flags](#), [l4virtio_block_header_t::ioprio](#), [L4virtio::Virtqueue::Desc::len](#), [L4virtio::Virtqueue::Desc::Flags::raw](#), [l4virtio_block_header_t::sector](#), and [l4virtio_block_header_t::type](#)

The documentation for this class was generated from the following file:

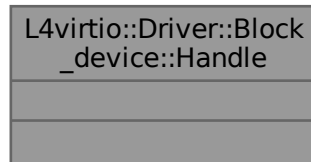
- [I4/I4virtio/client/virtio-block](#)

16.393 L4virtio::Driver::Block_device::Handle Class Reference

[Handle](#) to an ongoing request.

```
#include <virtio-block>
```

Collaboration diagram for L4virtio::Driver::Block_device::Handle:



16.393.1 Detailed Description

[Handle](#) to an ongoing request.

Definition at line 56 of file [virtio-block](#).

The documentation for this class was generated from the following file:

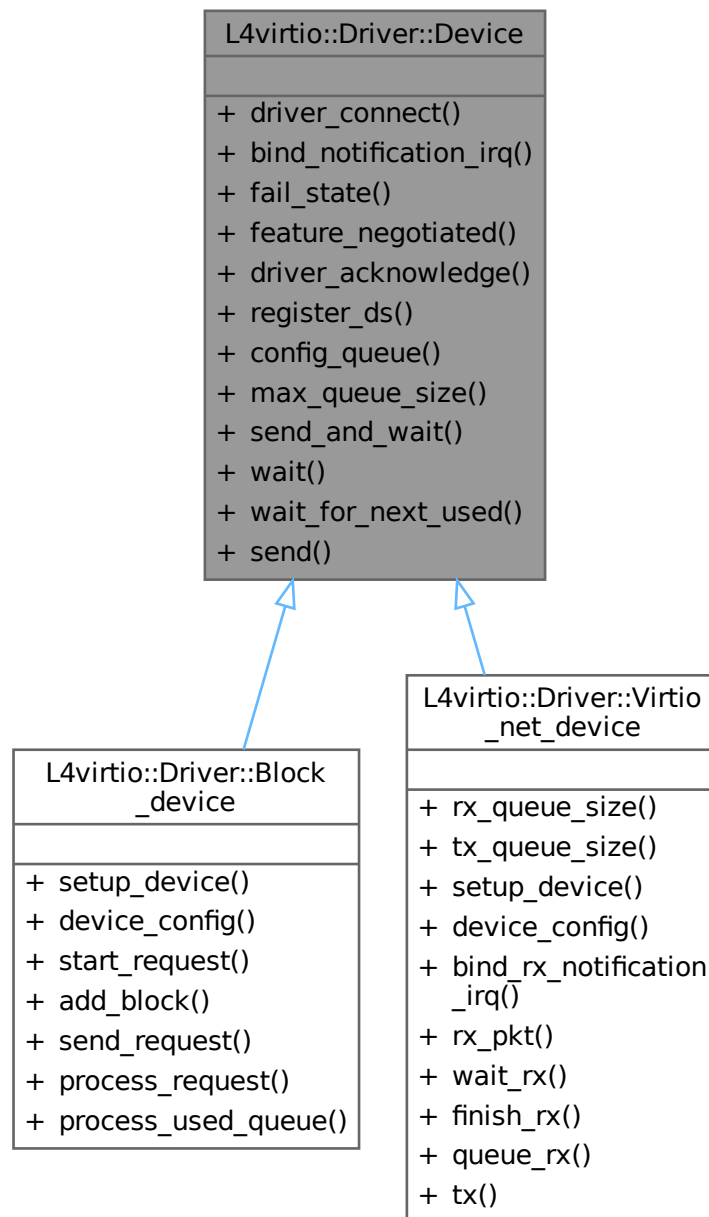
- l4/l4virtio/client/virtio-block

16.394 L4virtio::Driver::Device Class Reference

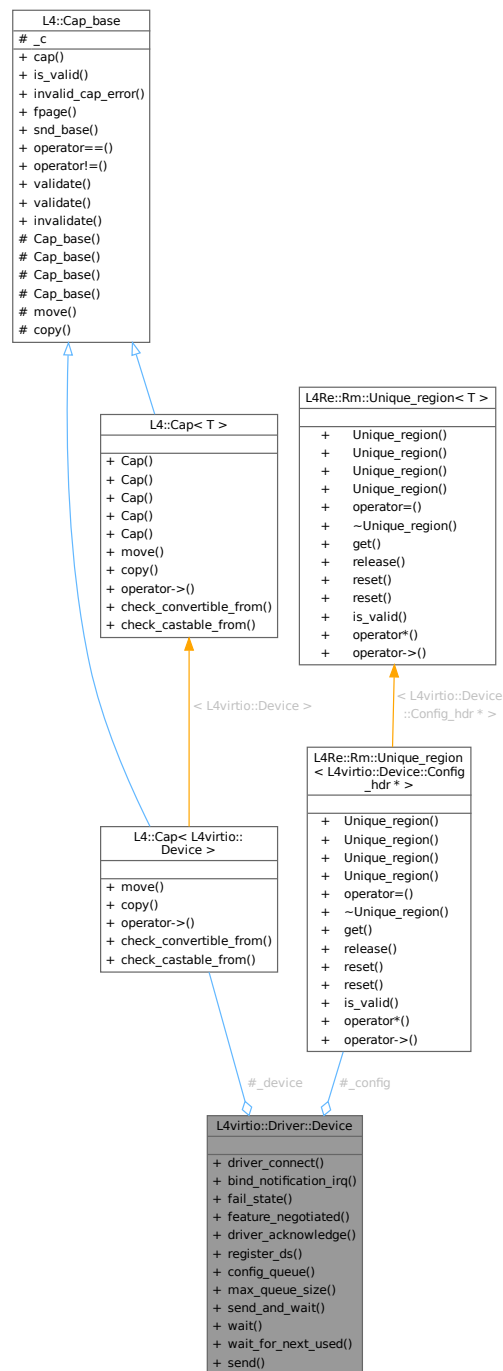
Client-side implementation for a general virtio device.

```
#include <l4virtio>
```

Inheritance diagram for L4virtio::Driver::Device:



Collaboration diagram for L4virtio::Driver::Device:



Public Member Functions

- void `driver_connect` (`L4::Cap< L4virtio::Device >` `src`, bool `manage_notify=true`)
Contacts the device and starts the initial handshake.
- int `bind_notification_irq` (unsigned index, `L4::Cap< L4::Triggerable >` `irq`) const
Register a triggerable to receive notifications from the device.
- bool `fail_state` () const

- Return true if the device is in a fail state.*

 - bool `feature_negotiated` (unsigned int feat) const

Check if a particular feature bit was negotiated with the device.
- int `driver_acknowledge` ()

Finalize handshake with the device.
- int `register_ds` (L4::Cap< L4Re::Dataspace > ds, l4_umword_t offset, l4_umword_t size, l4_uint64_t *devaddr)

Share a dataspace with the device.
- int `config_queue` (int num, unsigned size, l4_uint64_t desc_addr, l4_uint64_t avail_addr, l4_uint64_t used_addr)

Send the virtqueue configuration to the device.
- int `max_queue_size` (int num) const

Maximum queue size allowed by the device.
- int `send_and_wait` (Virtqueue &queue, l4_uint16_t descno)

Send a request to the device and wait for it to be processed.
- int `wait` (int index) const

Wait for a notification from the device.
- int `wait_for_next_used` (Virtqueue &queue, l4_uint32_t *len=NULLPTR) const

Wait for the next item to arrive in the used queue and return it.
- void `send` (Virtqueue &queue, l4_uint16_t descno)

Send a request to the device.

16.394.1 Detailed Description

Client-side implementation for a general virtio device.

Definition at line 31 of file `l4virtio`.

16.394.2 Member Function Documentation

16.394.2.1 bind_notification_irq()

```
int L4virtio::Driver::Device::bind_notification_irq (
    unsigned index,
    L4::Cap< L4::Triggerable > irq) const [inline]
```

Register a triggerable to receive notifications from the device.

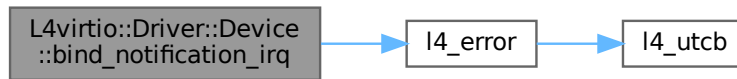
Parameters

	<i>index</i>	Index of the interrupt.
out	<i>irq</i>	Triggerable to register for notifications.

Definition at line 129 of file `l4virtio`.

References `l4_error()`.

Here is the call graph for this function:



16.394.2.2 config_queue()

```

int L4virtio::Driver::Device::config_queue (
    int num,
    unsigned size,
    l4_uint64_t desc_addr,
    l4_uint64_t avail_addr,
    l4_uint64_t used_addr) [inline]
  
```

Send the virtqueue configuration to the device.

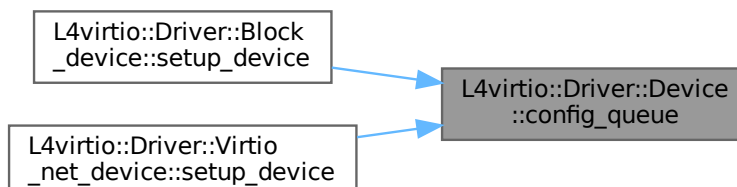
Parameters

<i>num</i>	Number of queue to configure.
<i>size</i>	Size of rings in the queue, must be a power of 2)
<i>desc_addr</i>	Address of descriptor table (device address)
<i>avail_addr</i>	Address of available ring (device address)
<i>used_addr</i>	Address of used ring (device address)

Definition at line 212 of file [l4virtio](#).

Referenced by [L4virtio::Driver::Block_device::setup_device\(\)](#), and [L4virtio::Driver::Virtio_net_device::setup_device\(\)](#).

Here is the caller graph for this function:



16.394.2.3 driver_acknowledge()

```
int L4virtio::Driver::Device::driver_acknowledge () [inline]
```

Finalize handshake with the device.

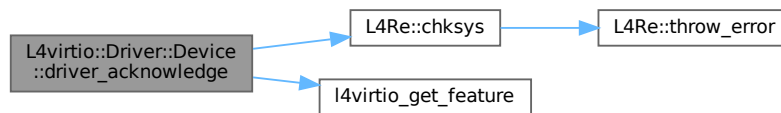
Must be called after all queues have been set up and before the first request is sent. It is still possible to add more shared dataspace after the handshake has been finished.

Definition at line 156 of file [l4virtio](#).

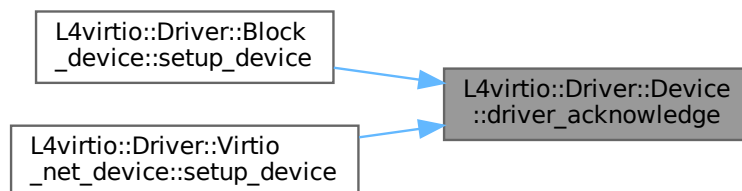
References [L4Re::chksys\(\)](#), [L4_EINVAL](#), [L4_EIO](#), [L4_ENODEV](#), [L4_EOK](#), [L4VIRTIO_FEATURE_VERSION_1](#), [l4virtio_get_feature\(\)](#), [L4VIRTIO_STATUS_DRIVER_OK](#), and [L4VIRTIO_STATUS_FEATURES_OK](#).

Referenced by [L4virtio::Driver::Block_device::setup_device\(\)](#), and [L4virtio::Driver::Virtio_net_device::setup_device\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.394.2.4 driver_connect()

```
void L4virtio::Driver::Device::driver_connect (
    L4::Cap< L4virtio::Device > srvcap,
    bool manage_notify = true) [inline]
```

Contacts the device and starts the initial handshake.

Parameters

<i>svrcap</i>	Capability for device communication.
<i>manage_notify</i>	Set up a semaphore for notifications from the device. See below.

Exceptions

L4::Runtime_error	if the initialisation fails
-----------------------------------	-----------------------------

This function contacts the server, sets up the notification channels and the configuration dataspace. After this is done, the caller can set up any dataspaces it needs. The initialisation then needs to be finished by calling [driver_acknowledge\(\)](#).

Per default this function creates and registers a semaphore for receiving notification from the device. This semaphore is used in the blocking functions [send_and_wait\(\)](#), [wait\(\)](#) and [next_used\(\)](#).

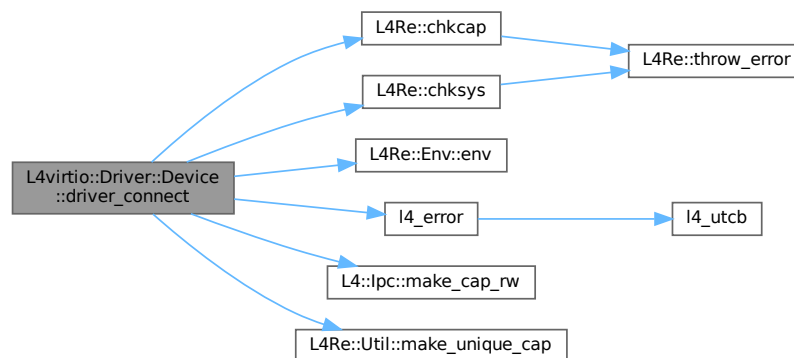
When `manage_notify` is false, then the caller may manually register and handle notification interrupts from the device. This is for example useful, when the client runs in an application with a server loop.

Definition at line 56 of file [l4virtio](#).

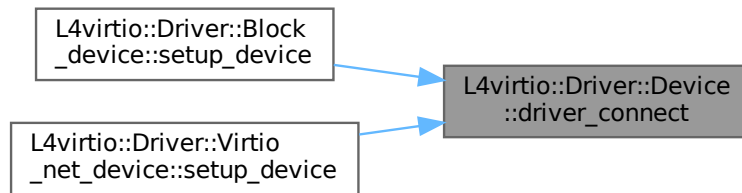
References [L4Re::chkcap\(\)](#), [L4Re::chksys\(\)](#), [L4Re::Env::env\(\)](#), [L4_EINVAL](#), [L4_EIO](#), [L4_ENODEV](#), [l4_error\(\)](#), [L4_PAGEMASK](#), [L4_PAGESHIFT](#), [L4_PAGESIZE](#), [L4_SUPERPAGESIZE](#), [L4VIRTIO_STATUS_ACKNOWLEDGE](#), [L4VIRTIO_STATUS_DRIVER](#), [L4::lpc::make_cap_rw\(\)](#), [L4Re::Util::make_unique_cap\(\)](#), [L4Re::Rm::F::RW](#), and [L4Re::Rm::F::Search_addr](#).

Referenced by [L4virtio::Driver::Block_device::setup_device\(\)](#), and [L4virtio::Driver::Virtio_net_device::setup_device\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.394.2.5 feature_negotiated()

```
bool L4virtio::Driver::Device::feature_negotiated (
    unsigned int feat) const [inline]
```

Check if a particular feature bit was negotiated with the device.

The result is only valid after [driver_acknowledge\(\)](#) was called (when the handshake with the device was completed).

Parameters

<i>feat</i>	The feature bit.
-------------	------------------

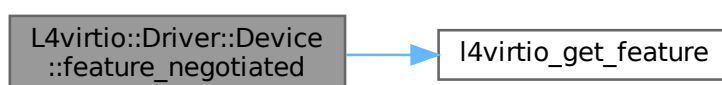
Return values

<i>true</i>	The feature is supported by both driver and device.
<i>false</i>	The feature is not supported by the driver and/or device.

Definition at line [145](#) of file [l4virtio](#).

References [l4virtio_get_feature\(\)](#).

Here is the call graph for this function:



16.394.2.6 max_queue_size()

```
int L4virtio::Driver::Device::max_queue_size (
    int num) const [inline]
```

Maximum queue size allowed by the device.

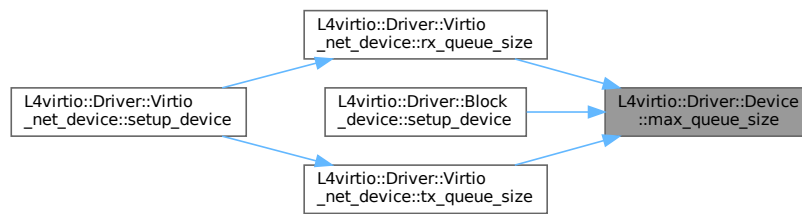
Parameters

<i>num</i>	Number of queue for which to determine the maximum size.
------------	--

Definition at line 230 of file [l4virtio](#).

Referenced by [L4virtio::Driver::Virtio_net_device::rx_queue_size\(\)](#), [L4virtio::Driver::Block_device::setup_device\(\)](#), and [L4virtio::Driver::Virtio_net_device::tx_queue_size\(\)](#).

Here is the caller graph for this function:



16.394.2.7 register_ds()

```
int L4virtio::Driver::Device::register_ds (
    L4::Cap< L4Re::Dataspace > ds,
    l4_umword_t offset,
    l4_umword_t size,
    l4_uint64_t * devaddr) [inline]
```

Share a dataspace with the device.

Parameters

<i>ds</i>	Dataspace to share with the device.
<i>offset</i>	Offset in dataspace where the shared part starts.
<i>size</i>	Total size in bytes of the shared space.
<i>devaddr</i>	Start of shared space in the device address space.

Although this function allows to share only a part of the given dataspace for convenience, the granularity of sharing is always the dataspace level. Thus, the remainder of the dataspace is not protected from the device.

When communicating with the device, addresses must be given with respect to the device address space. This is not the same as the virtual address space of the client in order to not leak information about the address space layout.

Definition at line 196 of file [l4virtio](#).

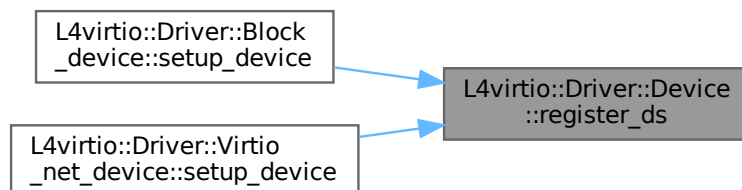
References [L4::lpc::make_cap_rw\(\)](#).

Referenced by [L4virtio::Driver::Block_device::setup_device\(\)](#), and [L4virtio::Driver::Virtio_net_device::setup_device\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.394.2.8 send()

```
void L4virtio::Driver::Device::send (
    Virtqueue & queue,
    l4_uint16_t descno) [inline]
```

Send a request to the device.

Parameters

<i>queue</i>	Queue that contains the request in its descriptor table
<i>descno</i>	Index of first entry in descriptor table where

Definition at line 312 of file [l4virtio](#).

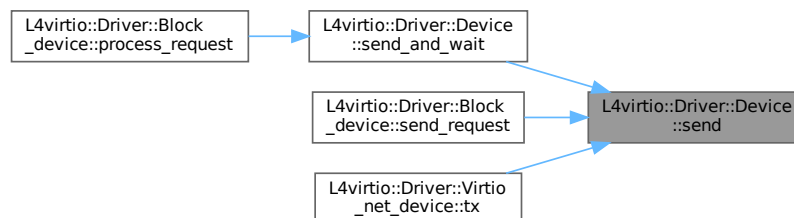
References [L4virtio::Driver::Virtqueue::enqueue_descriptor\(\)](#).

Referenced by [send_and_wait\(\)](#), [L4virtio::Driver::Block_device::send_request\(\)](#), and [L4virtio::Driver::Virtio_net_device::tx\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.394.2.9 send_and_wait()

```

int L4virtio::Driver::Device::send_and_wait (
    Virtqueue & queue,
    l4_uint16_t descno) [inline]
  
```

Send a request to the device and wait for it to be processed.

Parameters

<i>queue</i>	Queue that contains the request in its descriptor table
<i>descno</i>	Index of first entry in descriptor table where

This function provides a simple mechanism to send requests synchronously. It must not be used with other requests at the same time as it directly waits for a notification on the device irq cap.

Precondition

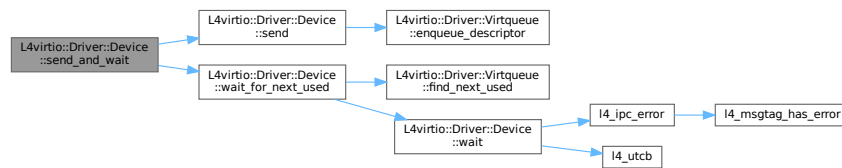
`driver_connect()` was called with `manage_notify`.

Definition at line 247 of file `l4virtio`.

References `L4_EINVAL`, `L4_EOK`, `send()`, and `wait_for_next_used()`.

Referenced by `L4virtio::Driver::Block_device::process_request()`.

Here is the call graph for this function:



Here is the caller graph for this function:

**16.394.2.10 wait()**

```
int L4virtio::Driver::Device::wait (
    int index) const [inline]
```

Wait for a notification from the device.

Parameters

<i>index</i>	Notification slot to wait for.
--------------	--------------------------------

Precondition

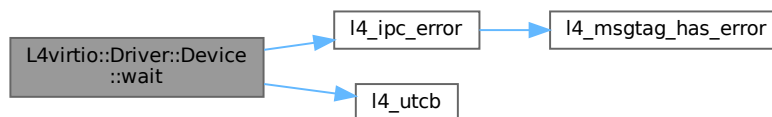
`driver_connect()` was called with `manage_notify`.

Definition at line 268 of file `l4virtio`.

References `L4_EEXIST`, `l4_ipc_error()`, and `l4_utcb()`.

Referenced by `wait_for_next_used()`.

Here is the call graph for this function:



Here is the caller graph for this function:

**16.394.2.11 wait_for_next_used()**

```

int L4virtio::Driver::Device::wait_for_next_used (
    Virtqueue & queue,
    l4_uint32_t * len = nullptr) const [inline]
  
```

Wait for the next item to arrive in the used queue and return it.

Parameters

	<i>queue</i>	A queue.
out	<i>len</i>	(optional) Size of valid data in finished block. Note that this is the value reported by the device, which may set it to a value that is larger than the original buffer size.

Return values

≥ 0	Descriptor number of item removed from used queue.
< 0	IPC error while waiting for notification.

The call blocks until the next item is available in the used queue.

Precondition

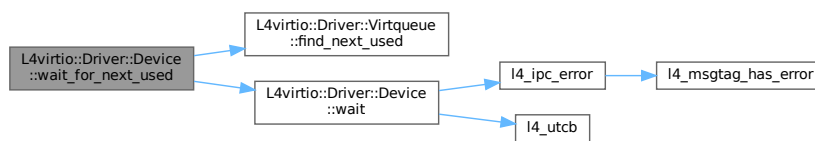
[driver_connect\(\)](#) was called with `manage_notify`.

Definition at line 291 of file [l4virtio](#).

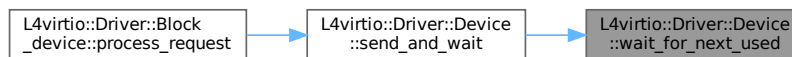
References [L4virtio::Driver::Virtqueue::find_next_used\(\)](#), and [wait\(\)](#).

Referenced by [send_and_wait\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

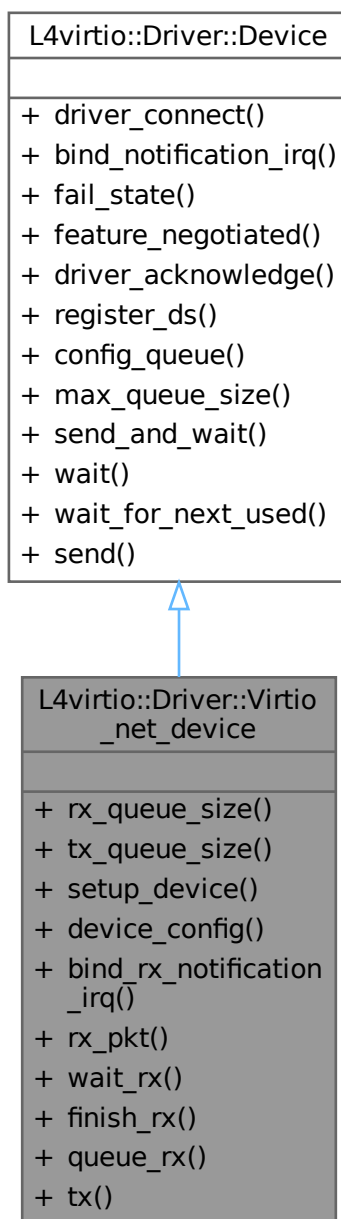
- `I4/I4virtio/client/I4virtio`

16.395 L4virtio::Driver::Virtio_net_device Class Reference

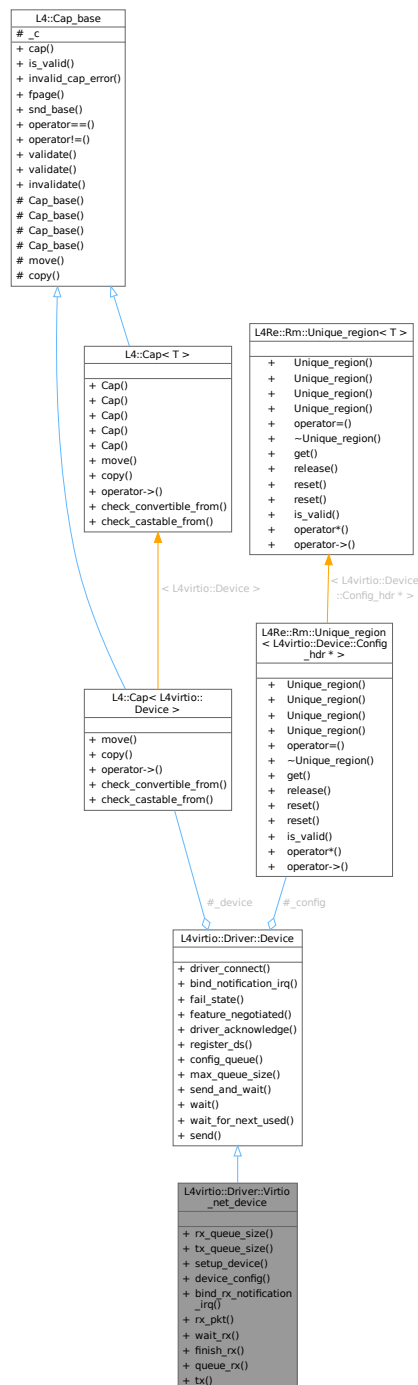
Simple class for accessing a virtio net device.

```
#include <virtio-net>
```

Inheritance diagram for L4virtio::Driver::Virtio_net_device:



Collaboration diagram for L4virtio::Driver::Virtio_net_device:



Data Structures

- struct [Packet](#)

Structure for a network packet (header including data) with maximum size, assuming that no extra features have been negotiated.

Public Member Functions

- int [rx_queue_size](#) () const
Return the maximum receive queue size allowed by the device.
- int [tx_queue_size](#) () const
Return the maximum transmit queue size allowed by the device.
- void [setup_device](#) (L4::Cap< L4virtio::Device > srvcap)
Establish a connection to the device and set up shared memory.
- [l4virtio_net_config_t](#) const & [device_config](#) () const
Return a reference to the device configuration.
- int [bind_rx_notification_irq](#) (L4::Cap< L4::Thread > thread, [l4_umword_t](#) label)
Bind the rx notification IRQ to the specified thread.
- [Packet](#) & [rx_pkt](#) ([l4_uint16_t](#) descno)
Return a reference to the RX packet buffer of the specified descriptor, e.g.
- [l4_uint16_t](#) [wait_rx](#) ([l4_uint32_t](#) *len=nullptr)
Block until a network packet has been received from the device and return the descriptor number.
- void [finish_rx](#) ([l4_uint16_t](#) descno)
Free an RX descriptor number to make it available for the RX queue again.
- void [queue_rx](#) ()
Queue new available descriptors in the RX queue.
- bool [tx](#) (std::function< [l4_uint32_t](#)([Packet](#) &)> prepare)
Attempt to allocate a descriptor in the TX queue and transmit the packet, after calling the prepare callback.

Public Member Functions inherited from [L4virtio::Driver::Device](#)

- void [driver_connect](#) (L4::Cap< L4virtio::Device > srvcap, bool manage_notify=true)
Contacts the device and starts the initial handshake.
- int [bind_notification_irq](#) (unsigned index, L4::Cap< L4::Triggerable > irq) const
Register a triggerable to receive notifications from the device.
- bool [fail_state](#) () const
Return true if the device is in a fail state.
- bool [feature_negotiated](#) (unsigned int feat) const
Check if a particular feature bit was negotiated with the device.
- int [driver_acknowledge](#) ()
Finalize handshake with the device.
- int [register_ds](#) (L4::Cap< L4Re::Dataspace > ds, [l4_umword_t](#) offset, [l4_umword_t](#) size, [l4_uint64_t](#) *devaddr)
Share a dataspace with the device.
- int [config_queue](#) (int num, unsigned size, [l4_uint64_t](#) desc_addr, [l4_uint64_t](#) avail_addr, [l4_uint64_t](#) used_↔ addr)
Send the virtqueue configuration to the device.
- int [max_queue_size](#) (int num) const
Maximum queue size allowed by the device.
- int [send_and_wait](#) ([Virtqueue](#) &queue, [l4_uint16_t](#) descno)
Send a request to the device and wait for it to be processed.
- int [wait](#) (int index) const
Wait for a notification from the device.
- int [wait_for_next_used](#) ([Virtqueue](#) &queue, [l4_uint32_t](#) *len=nullptr) const
Wait for the next item to arrive in the used queue and return it.
- void [send](#) ([Virtqueue](#) &queue, [l4_uint16_t](#) descno)
Send a request to the device.

16.395.1 Detailed Description

Simple class for accessing a virtio net device.

Definition at line 30 of file [virtio-net](#).

16.395.2 Member Function Documentation

16.395.2.1 `bind_rx_notification_irq()`

```
int L4virtio::Driver::Virtio_net_device::bind_rx_notification_irq (
    L4::Cap< L4::Thread > thread,
    l4_umword_t label) [inline]
```

Bind the rx notification IRQ to the specified thread.

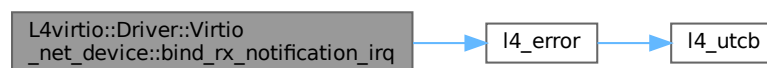
Parameters

<i>thread</i>	Thread to bind the notification IRQ to.
<i>label</i>	Label to assign to the IRQ.

Definition at line 169 of file [virtio-net](#).

References [l4_error\(\)](#).

Here is the call graph for this function:



16.395.2.2 `finish_rx()`

```
void L4virtio::Driver::Virtio_net_device::finish_rx (
    l4_uint16_t descno) [inline]
```

Free an RX descriptor number to make it available for the RX queue again.

Parameters

<i>descno</i>	Descriptor number in the virtio queue.
---------------	--

Usually [queue_rx\(\)](#) should be called afterwards to queue the freed descriptor(s).

Definition at line 230 of file [virtio-net](#).

16.395.2.3 rx_pkt()

```
Packet & L4virtio::Driver::Virtio_net_device::rx_pkt (
    l4_uint16_t descno) [inline]
```

Return a reference to the RX packet buffer of the specified descriptor, e.g.

from [wait_rx\(\)](#).

Parameters

<i>descno</i>	Descriptor number in the virtio queue.
---------------	--

Definition at line 180 of file [virtio-net](#).

16.395.2.4 rx_queue_size()

```
int L4virtio::Driver::Virtio_net_device::rx_queue_size () const [inline]
```

Return the maximum receive queue size allowed by the device.

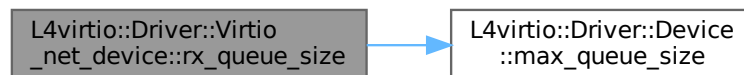
[wait_rx\(\)](#) will return a descriptor number that is smaller than this size.

Definition at line 47 of file [virtio-net](#).

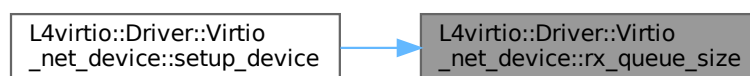
References [L4virtio::Driver::Device::max_queue_size\(\)](#).

Referenced by [setup_device\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.395.2.5 `setup_device()`

```
void L4virtio::Driver::Virtio_net_device::setup_device (  
    L4::Cap< L4virtio::Device > srvcap) [inline]
```

Establish a connection to the device and set up shared memory.

Parameters

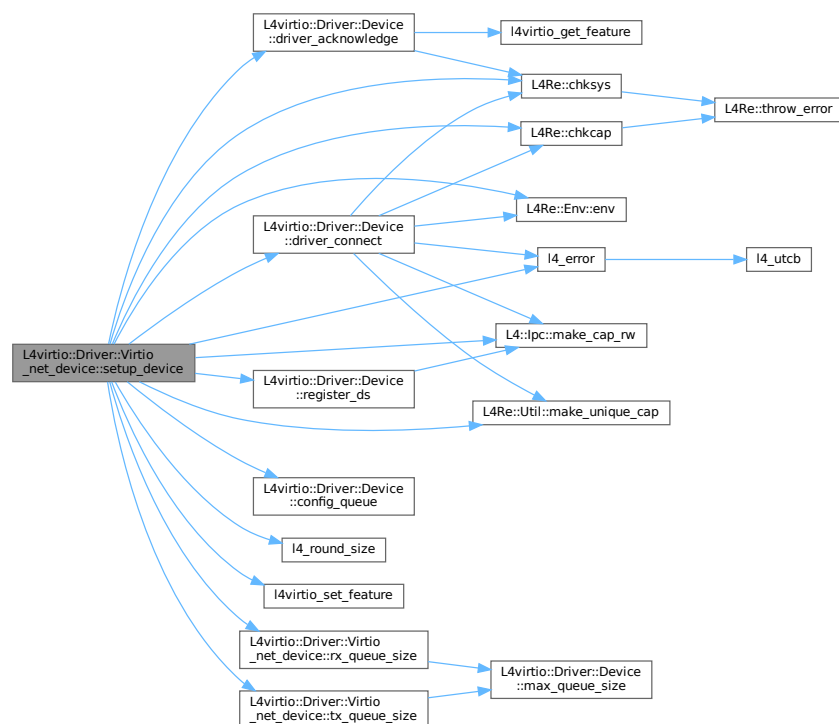
<i>srvcap</i>	IPC capability of the channel to the server.
---------------	--

This function starts a handshake with the device and sets up the virtqueues for communication and the additional data structures for the network device.

Definition at line 66 of file [virtio-net](#).

References [L4Re::chkcap\(\)](#), [L4Re::chksys\(\)](#), [L4virtio::Driver::Device::config_queue\(\)](#), [L4Re::Mem_alloc::Continuous](#), [L4virtio::Driver::Device::driver_acknowledge\(\)](#), [L4virtio::Driver::Device::driver_connect\(\)](#), [L4Re::Env::env\(\)](#), [L4_EINVAL](#), [L4_ENODEV](#), [I4_error\(\)](#), [L4_PAGESHIFT](#), [I4_round_size\(\)](#), [L4VIRTIO_FEATURE_VERSION_1](#), [L4VIRTIO_ID_NET](#), [I4virtio_set_feature\(\)](#), [L4::lpc::make_cap_rw\(\)](#), [L4Re::Util::make_unique_cap\(\)](#), [L4Re::Mem_alloc::Pinned](#), [L4virtio::Driver::Device::register_ds\(\)](#), [L4Re::Rm::F::RW](#), [rx_queue_size\(\)](#), [L4Re::Rm::F::Search_addr](#), and [tx_queue_size\(\)](#).

Here is the call graph for this function:



16.395.2.6 tx()

```
bool L4virtio::Driver::Virtio_net_device::tx (
    std::function< I4_uint32_t(Packet &)> prepare) [inline]
```

Attempt to allocate a descriptor in the TX queue and transmit the packet, after calling the prepare callback.

Parameters

<i>prepare</i>	Function that fills the packet with data, should return the length of the data copied to the packet.
----------------	--

Return values

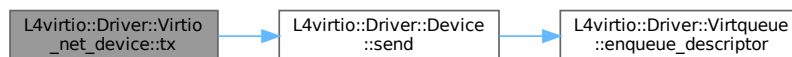
<i>true</i>	The packet was queued.
<i>false</i>	TX queue is full.

The prepare callback should fill the packet with data and return the length of the packet data (without the size of the virtio-net packet header).

Definition at line 260 of file [virtio-net](#).

References [L4virtio::Driver::Device::send\(\)](#).

Here is the call graph for this function:



16.395.2.7 tx_queue_size()

```
int L4virtio::Driver::Virtio_net_device::tx_queue_size () const [inline]
```

Return the maximum transmit queue size allowed by the device.

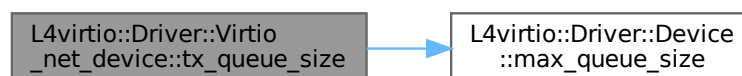
[tx\(\)](#) will fail if the amount of queued packets exceeds this size.

Definition at line 54 of file [virtio-net](#).

References [L4virtio::Driver::Device::max_queue_size\(\)](#).

Referenced by [setup_device\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.395.2.8 wait_rx()

```

14_uint16_t L4virtio::Driver::Virtio_net_device::wait_rx (
    14_uint32_t * len = nullptr) [inline]
  
```

Block until a network packet has been received from the device and return the descriptor number.

Precondition

The calling thread must be bound to the rx notification IRQ via `bind_rx_notification_irq()`.

Parameters

out	len	(optional) Length of valid data in RX packet.
-----	-----	---

Returns

Descriptor number of received packet.

The packet data can be obtained with `rx_pkt()`. `finish_rx()` should be called after the packet buffer can be returned to the RX queue.

Definition at line 202 of file `virtio-net`.

References `L4Re::chksys()`.

Here is the call graph for this function:



The documentation for this class was generated from the following file:

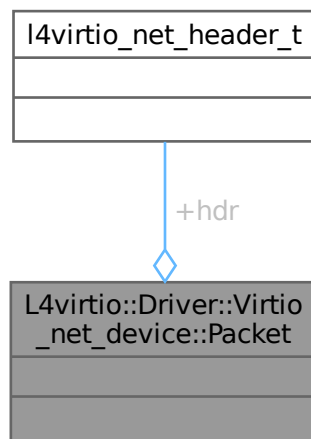
- `l4/l4virtio/client/virtio-net`

16.396 L4virtio::Driver::Virtio_net_device::Packet Struct Reference

Structure for a network packet (header including data) with maximum size, assuming that no extra features have been negotiated.

```
#include <virtio-net>
```

Collaboration diagram for L4virtio::Driver::Virtio_net_device::Packet:



16.396.1 Detailed Description

Structure for a network packet (header including data) with maximum size, assuming that no extra features have been negotiated.

Definition at line 37 of file [virtio-net](#).

The documentation for this struct was generated from the following file:

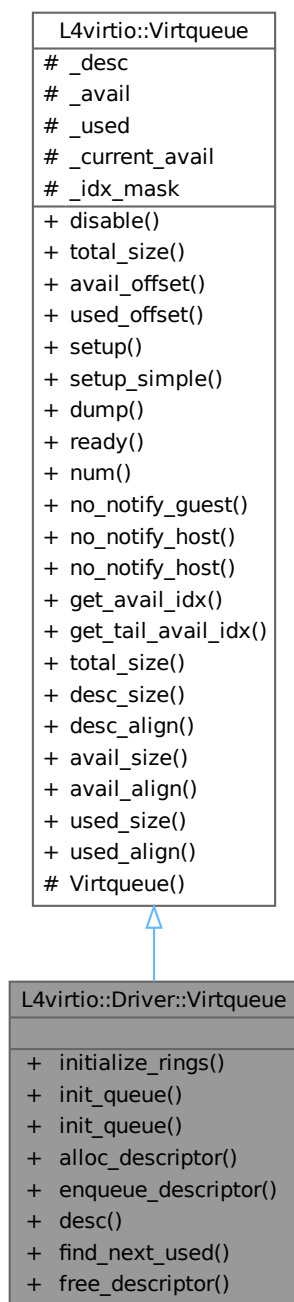
- `I4/I4virtio/client/virtio-net`

16.397 L4virtio::Driver::Virtqueue Class Reference

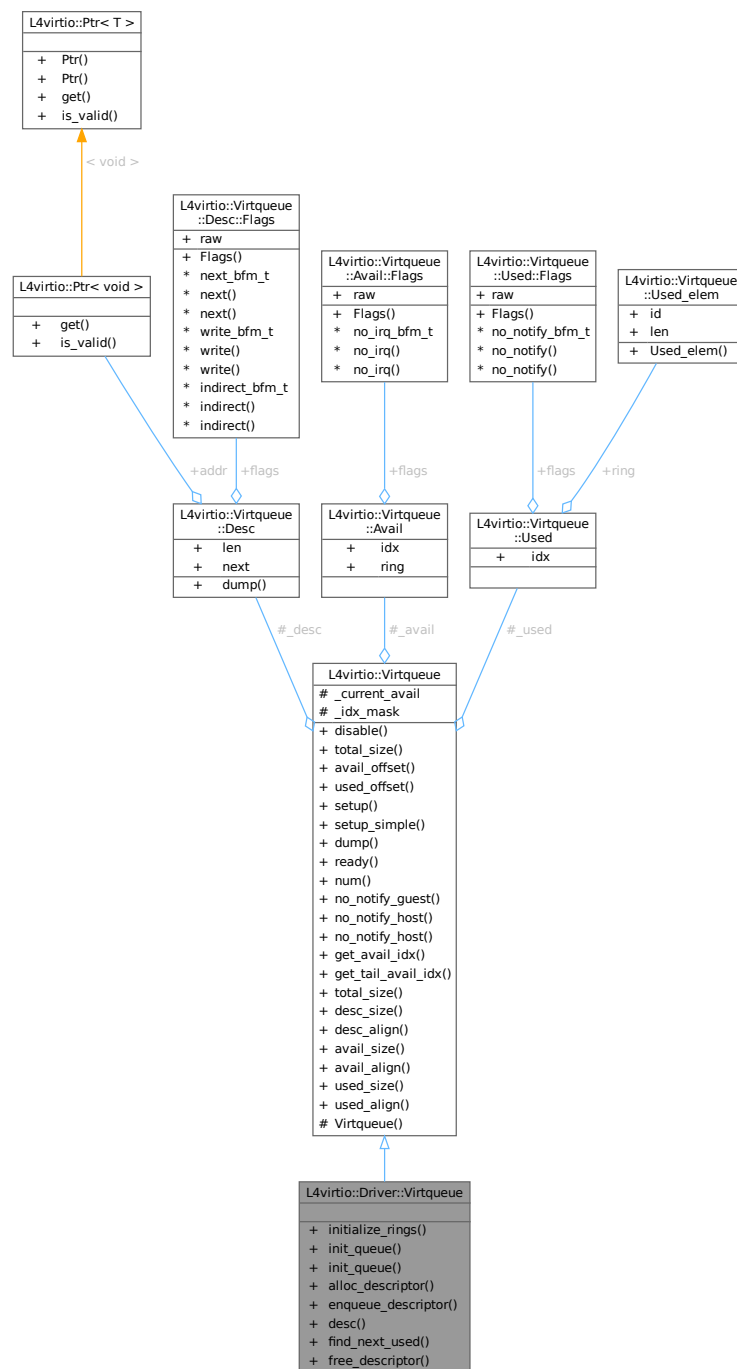
Driver-side implementation of a [Virtqueue](#).

```
#include <virtqueue>
```

Inheritance diagram for L4virtio::Driver::Virtqueue:



Collaboration diagram for L4virtio::Driver::Virtqueue:



Public Member Functions

- void [initialize_rings](#) (unsigned [num](#))
Initialize the descriptor table and the index structures of this queue.
- void [init_queue](#) (unsigned [num](#), void *[desc](#), void *[avail](#), void *[used](#))
Initialize this virtqueue.
- void [init_queue](#) (unsigned [num](#), void *[base](#))

- Initialize this virtqueue.*
- [l4_uint16_t alloc_descriptor](#) ()
Allocate and return an unused descriptor from the descriptor table.
- void [enqueue_descriptor](#) ([l4_uint16_t](#) descno)
Enqueue a descriptor in the available ring.
- [Desc & desc](#) ([l4_uint16_t](#) descno)
Return a reference to a descriptor in the descriptor table.
- [l4_uint16_t find_next_used](#) ([l4_uint32_t](#) *len=nullptr)
Return the next finished block.
- void [free_descriptor](#) ([l4_uint16_t](#) head, [l4_uint16_t](#) tail)
Free a chained list of descriptors in the descriptor queue.

Public Member Functions inherited from [L4virtio::Virtqueue](#)

- void [disable](#) ()
Completely disable the queue.
- unsigned long [total_size](#) () const
Calculate the total size of this virtqueue.
- unsigned long [avail_offset](#) () const
Get the offset of the available ring from the descriptor table.
- unsigned long [used_offset](#) () const
Get the offset of the used ring from the descriptor table.
- void [setup](#) (unsigned [num](#), void *desc, void *avail, void *used)
Enable this queue.
- void [setup_simple](#) (unsigned [num](#), void *ring)
Enable this queue.
- void [dump](#) ([Desc](#) const *d) const
Dump descriptors for this queue.
- bool [ready](#) () const
Test if this queue is in working state.
- unsigned [num](#) () const
- bool [no_notify_guest](#) () const
Get the no IRQ flag of this queue.
- bool [no_notify_host](#) () const
Get the no notify flag of this queue.
- void [no_notify_host](#) (bool value)
Set the no-notify flag for this queue.
- [l4_uint16_t](#) [get_avail_idx](#) () const
Get available index from available ring (for debugging).
- [l4_uint16_t](#) [get_tail_avail_idx](#) () const
Get tail-available index stored in local state (for debugging).

Additional Inherited Members

Public Types inherited from [L4virtio::Virtqueue](#)

- enum
Fixed alignment values for different parts of a virtqueue.

Static Public Member Functions inherited from [L4virtio::Virtqueue](#)

- static unsigned long [total_size](#) (unsigned [num](#))
Calculate the total size for a virtqueue of the given dimensions.
- static unsigned long [desc_size](#) (unsigned [num](#))
Calculate the size of the descriptor table for [num](#) entries.
- static unsigned long [desc_align](#) ()
Get the alignment in zero LSBs needed for the descriptor table.
- static unsigned long [avail_size](#) (unsigned [num](#))
Calculate the size of the available ring for [num](#) entries.
- static unsigned long [avail_align](#) ()
Get the alignment in zero LSBs needed for the available ring.
- static unsigned long [used_size](#) (unsigned [num](#))
Calculate the size of the used ring for [num](#) entries.
- static unsigned long [used_align](#) ()
Get the alignment in zero LSBs needed for the used ring.

Protected Member Functions inherited from [L4virtio::Virtqueue](#)

- [Virtqueue](#) ()=default
Create a disabled virtqueue.

Protected Attributes inherited from [L4virtio::Virtqueue](#)

- [Desc](#) * [_desc](#) = nullptr
pointer to descriptor table, NULL if queue is off.
- [Avail](#) * [_avail](#) = nullptr
pointer to available ring.
- [Used](#) * [_used](#) = nullptr
pointer to used ring.
- [l4_uint16_t](#) [_current_avail](#) = 0
The life counter for the queue.
- [l4_uint16_t](#) [_idx_mask](#) = 0
mask used for indexing into the descriptor table and the rings.

16.397.1 Detailed Description

Driver-side implementation of a [Virtqueue](#).

Adds function for managing the descriptor list, enqueueing new and dequeueing finished requests.

Note

The [Virtqueue](#) implementation is not thread-safe.

Definition at line 470 of file [virtqueue](#).

16.397.2 Member Function Documentation

16.397.2.1 alloc_descriptor()

```
l4_uint16_t L4virtio::Driver::Virtqueue::alloc_descriptor () [inline]
```

Allocate and return an unused descriptor from the descriptor table.

The descriptor will be removed from the free list, the content should be considered undefined. After use, it needs to be freed using [free_descriptor\(\)](#).

Returns

The index of the reserved descriptor or Virtqueue::Eq if no free descriptor is available.

Note: the implementation uses $(2^{16} - 1)$ as the end of queue marker. That means that the final entry in the queue can not be allocated iff the queue size is 2^{16} .

Definition at line 558 of file [virtqueue](#).

References [L4virtio::Virtqueue::_desc](#).

16.397.2.2 desc()

```
Desc & L4virtio::Driver::Virtqueue::desc (
    l4_uint16_t descno) [inline]
```

Return a reference to a descriptor in the descriptor table.

Parameters

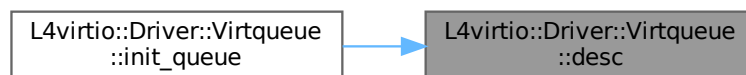
<i>descno</i>	Index of the descriptor, expected to be in correct range.
---------------	---

Definition at line 590 of file [virtqueue](#).

References [L4virtio::Virtqueue::_desc](#), and [L4virtio::Virtqueue::_idx_mask](#).

Referenced by [init_queue\(\)](#).

Here is the caller graph for this function:



16.397.2.3 enqueue_descriptor()

```
void L4virtio::Driver::Virtqueue::enqueue_descriptor (
    l4_uint16_t descno) [inline]
```

Enqueue a descriptor in the available ring.

Parameters

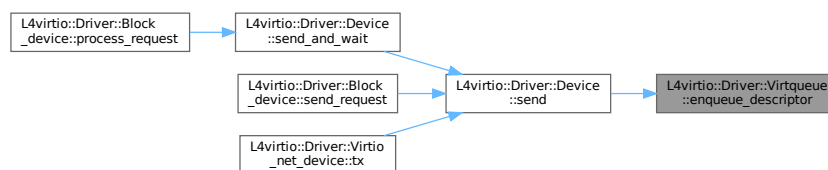
<i>descno</i>	Index of the head descriptor to enqueue.
---------------	--

Definition at line 574 of file [virtqueue](#).

References [L4virtio::Virtqueue::_avail](#), and [L4virtio::Virtqueue::_idx_mask](#).

Referenced by [L4virtio::Driver::Device::send\(\)](#).

Here is the caller graph for this function:



16.397.2.4 find_next_used()

```

14_uint16_t L4virtio::Driver::Virtqueue::find_next_used (
    14_uint32_t * len = nullptr) [inline]
  
```

Return the next finished block.

Parameters

out	<i>len</i>	(optional) Size of valid data in finished block. Note that this is the value reported by the device, which may set it to a value that is larger than the original buffer size.
-----	------------	--

Returns

Index of the head or `Virtqueue::Eoq` if no used element is currently available.

Definition at line 609 of file [virtqueue](#).

References [L4virtio::Virtqueue::_current_avail](#), [L4virtio::Virtqueue::_idx_mask](#), and [L4virtio::Virtqueue::_used](#).

Referenced by [L4virtio::Driver::Device::wait_for_next_used\(\)](#).

Here is the caller graph for this function:



16.397.2.5 free_descriptor()

```
void L4virtio::Driver::Virtqueue::free_descriptor (
    14_uint16_t head,
    14_uint16_t tail) [inline]
```

Free a chained list of descriptors in the descriptor queue.

Parameters

<i>head</i>	Index of the first element in the descriptor chain.
<i>tail</i>	Index of the last element in the descriptor chain.

Simply takes the descriptor chain and prepends it to the beginning of the free list. Assumes that the list has been correctly chained.

Definition at line 631 of file [virtqueue](#).

References [L4virtio::Virtqueue::_desc](#), and [L4virtio::Virtqueue::_idx_mask](#).

16.397.2.6 `init_queue()` [1/2]

```
void L4virtio::Driver::Virtqueue::init_queue (
    unsigned num,
    void * base) [inline]
```

Initialize this virtqueue.

Parameters

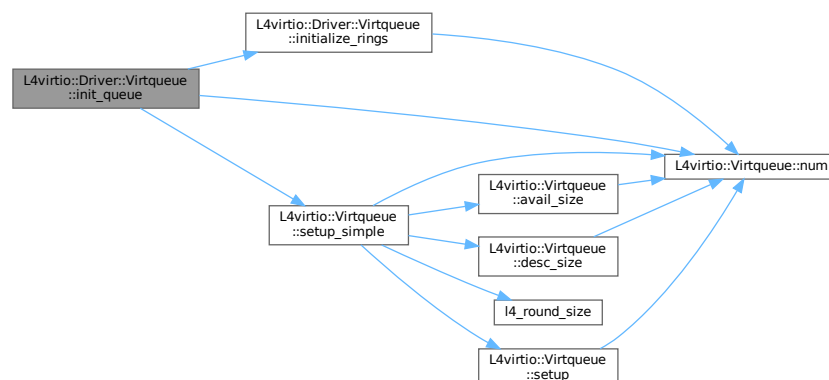
<i>num</i>	The number of entries in the descriptor table, the available ring, and the used ring (must be a power of 2).
<i>base</i>	The base address for the queue data structure.

This function sets up the memory and initializes the freelist.

Definition at line 537 of file [virtqueue](#).

References [initialize_rings\(\)](#), [L4virtio::Virtqueue::num\(\)](#), and [L4virtio::Virtqueue::setup_simple\(\)](#).

Here is the call graph for this function:



16.397.2.7 init_queue() [2/2]

```
void L4virtio::Driver::Virtqueue::init_queue (
    unsigned num,
    void * desc,
    void * avail,
    void * used) [inline]
```

Initialize this virtqueue.

Parameters

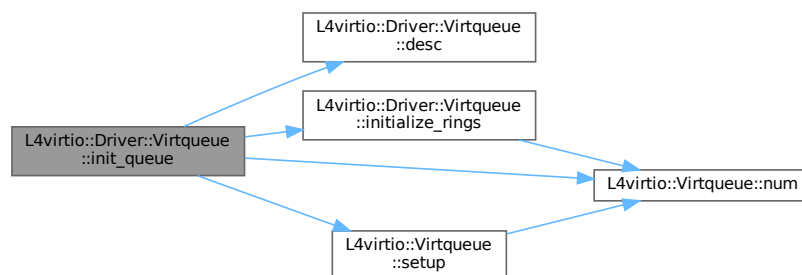
<i>num</i>	The number of entries in the descriptor table, the available ring, and the used ring (must be a power of 2).
<i>desc</i>	The address of the descriptor table. (Must be <code>Desc_align</code> aligned and at least <code>desc_size(num)</code> bytes in size.)
<i>avail</i>	The address of the available ring. (Must be <code>Avail_align</code> aligned and at least <code>avail_size(num)</code> bytes in size.)
<i>used</i>	The address of the used ring. (Must be <code>Used_align</code> aligned and at least <code>used_size(num)</code> bytes in size.)

This function sets up the memory and initializes the freelist.

Definition at line 522 of file [virtqueue](#).

References [desc\(\)](#), [initialize_rings\(\)](#), [L4virtio::Virtqueue::num\(\)](#), and [L4virtio::Virtqueue::setup\(\)](#).

Here is the call graph for this function:



16.397.2.8 initialize_rings()

```
void L4virtio::Driver::Virtqueue::initialize_rings (
    unsigned num) [inline]
```

Initialize the descriptor table and the index structures of this queue.

Parameters

<i>num</i>	The number of entries in the descriptor table, the available ring, and the used ring (must be a power of 2).
------------	--

Precondition

The queue must be set up correctly with [setup\(\)](#) or [setup_simple\(\)](#).

Definition at line [494](#) of file [virtqueue](#).

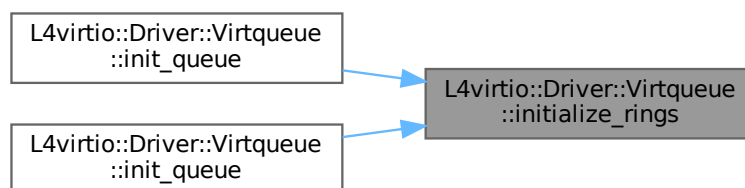
References [L4virtio::Virtqueue::_avail](#), [L4virtio::Virtqueue::_desc](#), [L4virtio::Virtqueue::_used](#), and [L4virtio::Virtqueue::num\(\)](#).

Referenced by [init_queue\(\)](#), and [init_queue\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

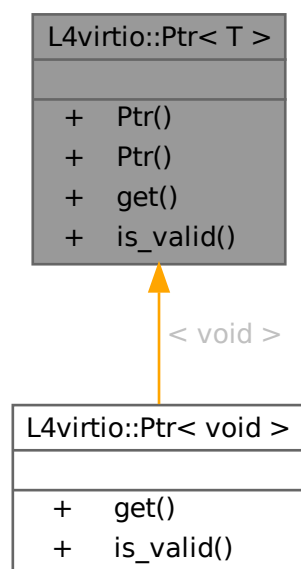
- [I4/I4virtio/virtqueue](#)

16.398 L4virtio::Ptr< T > Class Template Reference

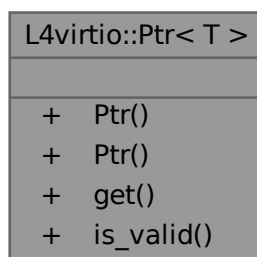
Pointer used in virtio descriptors.

```
#include <virtqueue>
```

Inheritance diagram for L4virtio::Ptr< T >:



Collaboration diagram for L4virtio::Ptr< T >:



Public Types

- enum `Invalid_type` { `Invalid` }
Type for making an invalid (NULL) `Ptr`.

Public Member Functions

- `Ptr` (`Invalid_type`)

Make and invalid [Ptr](#).

- **Ptr** ([l4_uint64_t](#) vm_addr)

Make a [Ptr](#) from a raw 64bit address.

- [l4_uint64_t](#) get () const
- bool [is_valid](#) () const

16.398.1 Detailed Description

```
template<typename T>
class L4virtio::Ptr< T >
```

Pointer used in virtio descriptors.

As the descriptor contain guest addresses these pointers cannot be dereferenced directly.

Definition at line 47 of file [virtqueue](#).

16.398.2 Member Enumeration Documentation

16.398.2.1 Invalid_type

```
template<typename T>
enum L4virtio::Ptr::Invalid_type
```

Type for making an invalid (NULL) [Ptr](#).

Enumerator

Invalid	Use to set a Ptr to invalid (NULL).
---------	---

Definition at line 51 of file [virtqueue](#).

16.398.3 Member Function Documentation

16.398.3.1 get()

```
template<typename T>
l4_uint64_t L4virtio::Ptr< T >::get () const [inline]
```

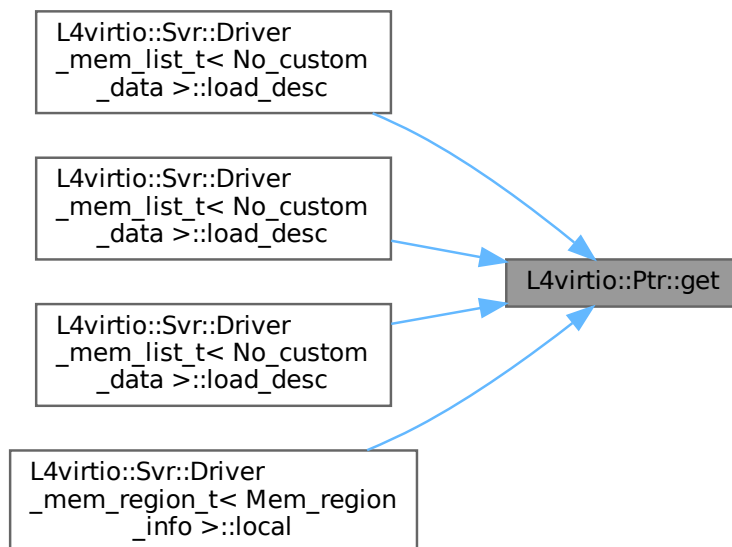
Returns

The raw 64bit address of the stored pointer.

Definition at line 62 of file [virtqueue](#).

Referenced by [L4virtio::Svr::Driver_mem_list_t< No_custom_data >::load_desc\(\)](#), [L4virtio::Svr::Driver_mem_list_t< No_custom_data >::load_desc\(\)](#), [L4virtio::Svr::Driver_mem_list_t< No_custom_data >::load_desc\(\)](#), and [L4virtio::Svr::Driver_mem_region_t< Mem_region_info >::local](#).

Here is the caller graph for this function:

**16.398.3.2 is_valid()**

```
template<typename T>
bool L4virtio::Ptr< T >::is_valid () const [inline]
```

Returns

true if the stored pointer is valid (not NULL).

Definition at line 65 of file [virtqueue](#).

The documentation for this class was generated from the following file:

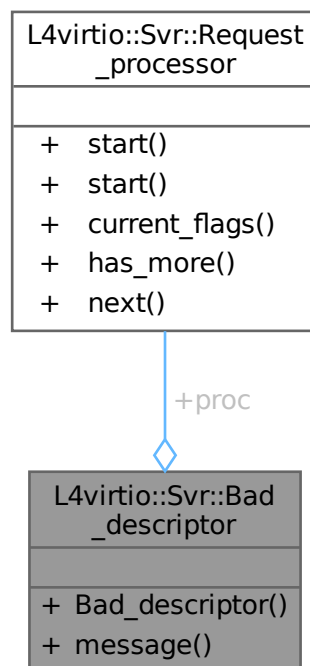
- `I4/I4virtio/virtqueue`

16.399 L4virtio::Svr::Bad_descriptor Struct Reference

Exception used by Queue to indicate descriptor errors.

```
#include <virtio>
```

Collaboration diagram for L4virtio::Svr::Bad_descriptor:



Public Types

- enum [Error](#) {
[Bad_address](#) , [Bad_rights](#) , [Bad_flags](#) , [Bad_next](#) ,
[Bad_size](#) }

The error code.

Public Member Functions

- [Bad_descriptor](#) ([Request_processor](#) const *[proc](#), [Error](#) e)
Make a bad descriptor exception.
- char const * [message](#) () const
Get a human readable description of the error code.

Data Fields

- [Request_processor](#) const * **proc**
The processor that triggered the exception.

16.399.1 Detailed Description

Exception used by Queue to indicate descriptor errors.

Definition at line 397 of file [virtio](#).

16.399.2 Member Enumeration Documentation

16.399.2.1 Error

```
enum L4virtio::Svr::Bad_descriptor::Error
```

The error code.

Enumerator

Bad_address	Address cannot be translated.
Bad_rights	Missing access rights on memory.
Bad_flags	Invalid combination of descriptor flags.
Bad_next	Invalid next index.
Bad_size	Invalid size of memory block.

Definition at line 400 of file [virtio](#).

16.399.3 Constructor & Destructor Documentation

16.399.3.1 Bad_descriptor()

```
L4virtio::Svr::Bad_descriptor::Bad_descriptor (  
    Request_processor const * proc,  
    Error e) [inline]
```

Make a bad descriptor exception.

Parameters

<i>proc</i>	The request processor causing the exception
<i>e</i>	The error code.

Definition at line 421 of file [virtio](#).

References [proc](#).

16.399.4 Member Function Documentation

16.399.4.1 message()

```
char const * L4virtio::Svr::Bad_descriptor::message () const [inline]
```

Get a human readable description of the error code.

Returns

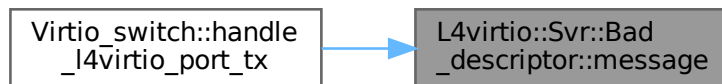
Message describing the error.

Definition at line 430 of file [virtio](#).

References [Bad_address](#), [Bad_flags](#), [Bad_next](#), [Bad_rights](#), and [Bad_size](#).

Referenced by [Virtio_switch::handle_l4virtio_port_tx\(\)](#).

Here is the caller graph for this function:



The documentation for this struct was generated from the following file:

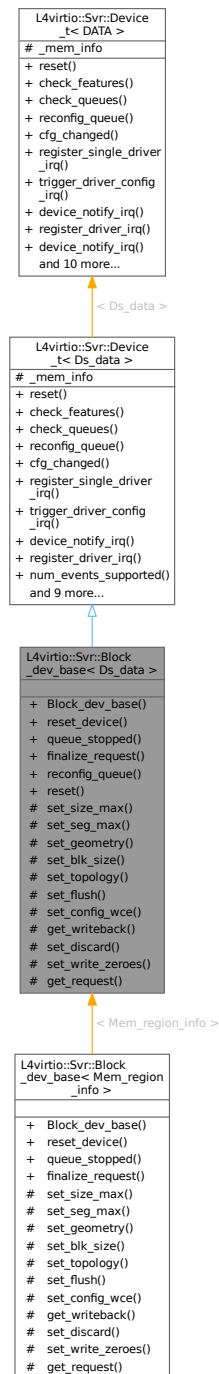
- `I4/I4virtio/server/virtio`

16.400 L4virtio::Svr::Block_dev_base< Ds_data > Class Template Reference

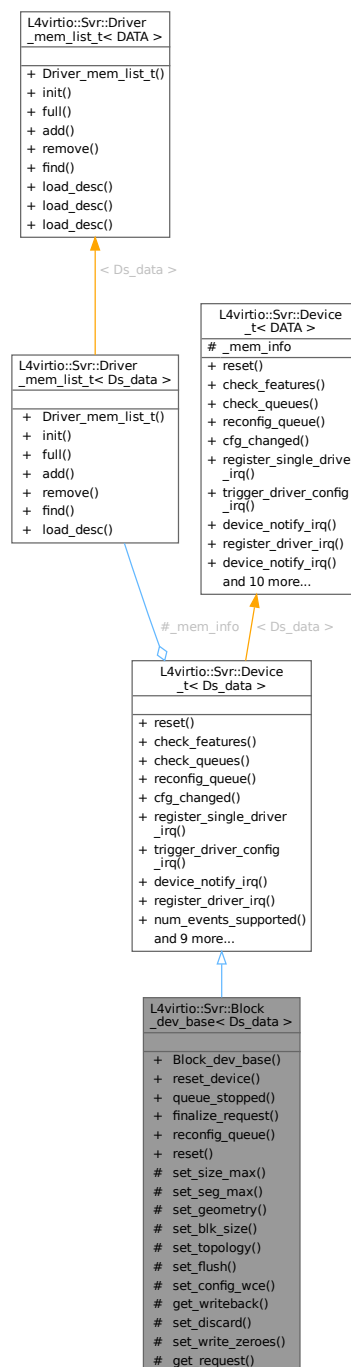
Base class for virtio block devices.

```
#include <virtio-block>
```

Inheritance diagram for L4virtio::Svr::Block_dev_base< Ds_data >:



Collaboration diagram for L4virtio::Svr::Block_dev_base< Ds_data >:



Public Member Functions

- **Block_dev_base** (`l4_uint32_t` vendor, unsigned queue_size, `l4_uint64_t` capacity, bool read_only)
Create a new virtio block device.
- virtual void **reset_device** ()=0
Reset the actual hardware device.
- virtual bool **queue_stopped** ()=0

- *Return true, if the queues should not be processed further.*
- void **finalize_request** (cxx::unique_ptr< [Request](#) > req, unsigned sz, [l4_uint8_t](#) status=L4VIRTIO_BLOCK_S_OK)
Releases resources related to a request and notifies the client.
- int **reconfig_queue** (unsigned idx) override
callback for client queue-config request
- void **reset** () override
reset callback, called for doing a device reset

Public Member Functions inherited from [L4virtio::Svr::Device_t< Ds_data >](#)

- virtual bool **check_features** ()
callback for checking the subset of accepted features
- virtual void **cfg_changed** (unsigned)
callback for client device configuration changes
- virtual [L4::Cap< L4::Irq >](#) **device_notify_irq** () const
callback to gather the device notification IRQ (old-style)
- virtual void **register_driver_irq** (unsigned idx)
Callback for registering an notification IRQ (multi IRQ).
- virtual unsigned **num_events_supported** () const
Return the highest notification index supported.
- [Device_t](#) ([Dev_config](#) *dev_config)
Make a device for the given config.
- [Mem_list](#) const * **mem_info** () const
Get the memory region list used for this device.
- void **reset_queue_config** (unsigned idx, unsigned num_max, bool inc_generation=false)
Trigger reset for the configuration space for queue idx.
- void **init_mem_info** (unsigned num)
Initialize the memory region list to the given maximum.
- void **device_error** ()
Transition device into DEVICE_NEEDS_RESET state.
- bool **setup_queue** ([Virtqueue](#) *q, unsigned qn, unsigned num_max)
Enable/disable the specified queue.
- bool **handle_mem_cmd_write** ()
Check for a value in the cmd register and handle a write.
- void **enable_trusted_ds_validation** ()
Enable trusted dataspace validation.
- void **add_trusted_dataspaces** (std::shared_ptr< [Ds_vector](#) const > ds)
Provide a list of trusted dataspaces that can be used for validation.

Protected Member Functions

- void **set_size_max** ([l4_uint32_t](#) sz)
Sets the maximum size of any single segment reported to client.
- void **set_seg_max** ([l4_uint32_t](#) sz)
Sets the maximum number of segments in a request that is reported to client.
- void **set_geometry** ([l4_uint16_t](#) cylinders, [l4_uint8_t](#) heads, [l4_uint8_t](#) sectors)
Set disk geometry that is reported to the client.
- void **set_blk_size** ([l4_uint32_t](#) sz)
Sets block disk size to be reported to the client.

- void `set_topology` (`l4_uint8_t` physical_block_exp, `l4_uint8_t` alignment_offset, `l4_uint32_t` min_io_size, `l4_uint32_t` opt_io_size)
Sets the I/O alignment information reported back to the client.
- void `set_flush` ()
Enables the flush command.
- void `set_config_wce` (`l4_uint8_t` writeback)
Sets cache mode and enables the writeback toggle.
- `l4_uint8_t` `get_writeback` ()
Get the writeback field from the configuration space.
- void `set_discard` (`l4_uint32_t` max_discard_sectors, `l4_uint32_t` max_discard_seg, `l4_uint32_t` discard_sector_alignment)
Sets constraints for and enables the discard command.
- void `set_write_zeroes` (`l4_uint32_t` max_write_zeroes_sectors, `l4_uint32_t` max_write_zeroes_seg, `l4_uint8_t` write_zeroes_may_unmap)
Sets constraints for and enables the write zeroes command.
- `cxx::unique_ptr`< `Request` > `get_request` ()
Return one request if available.

Additional Inherited Members

Protected Attributes inherited from `L4virtio::Svr::Device_t`< `Ds_data` >

- `Mem_list` `_mem_info`
Memory region list.

16.400.1 Detailed Description

```
template<typename Ds_data>
class L4virtio::Svr::Block_dev_base< Ds_data >
```

Base class for virtio block devices.

Use this class as a base to implement your own specific block device.

Definition at line 259 of file `virtio-block`.

16.400.2 Constructor & Destructor Documentation

16.400.2.1 `Block_dev_base()`

```
template<typename Ds_data>
L4virtio::Svr::Block_dev_base< Ds_data >::Block_dev_base (
    l4_uint32_t vendor,
    unsigned queue_size,
    l4_uint64_t capacity,
    bool read_only) [inline]
```

Create a new virtio block device.

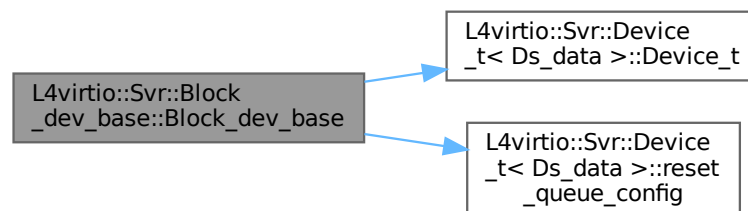
Parameters

<i>vendor</i>	Vendor ID
<i>queue_size</i>	Number of entries to provide in avail and used queue.
<i>capacity</i>	Size of the device in 512-byte sectors.
<i>read_only</i>	True, if the device should not be writable.

Definition at line 444 of file [virtio-block](#).

References [L4virtio::Svr::Device_t< Ds_data >::Device_t\(\)](#), [L4VIRTIO_FEATURE_VERSION_1](#), [L4VIRTIO_ID_BLOCK](#), and [L4virtio::Svr::Device_t< Ds_data >::reset_queue_config\(\)](#).

Here is the call graph for this function:



16.400.3 Member Function Documentation

16.400.3.1 finalize_request()

```

template<typename Ds_data>
void L4virtio::Svr::Block_dev_base< Ds_data >::finalize_request (
    cxx::unique_ptr< Request > req,
    unsigned sz,
    l4_uint8_t status = L4VIRTIO_BLOCK_S_OK) [inline]
  
```

Releases resources related to a request and notifies the client.

Parameters

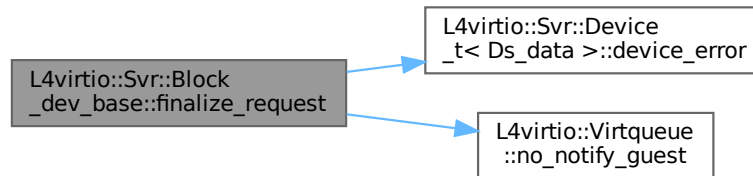
<i>req</i>	Pointer to request that has finished.
<i>sz</i>	Number of bytes consumed.
<i>status</i>	Status of request (see L4virtio_block_status).

This function must be called when an asynchronous request finishes, either successfully or with an error. The status byte in the request must have been set prior to calling it.

Definition at line 483 of file [virtio-block](#).

References [L4virtio::Svr::Device_t< Ds_data >::device_error\(\)](#), [L4VIRTIO_BLOCK_S_OK](#), [L4VIRTIO_IRQ_STATUS_VRING](#), and [L4virtio::Virtqueue::no_notify_guest\(\)](#).

Here is the call graph for this function:



16.400.3.2 get_writeback()

```
template<typename Ds_data>
l4_uint8_t L4virtio::Svr::Block_dev_base< Ds_data >::get_writeback () [inline], [protected]
```

Get the writeback field from the configuration space.

Returns

Value of the writeback field.

Definition at line 388 of file [virtio-block](#).

16.400.3.3 set_blk_size()

```
template<typename Ds_data>
void L4virtio::Svr::Block_dev_base< Ds_data >::set_blk_size (
    l4_uint32_t sz) [inline], [protected]
```

Sets block disk size to be reported to the client.

Setting this does not change the logical sector size used for addressing the device.

Definition at line 332 of file [virtio-block](#).

16.400.3.4 set_config_wce()

```
template<typename Ds_data>
void L4virtio::Svr::Block_dev_base< Ds_data >::set_config_wce (
    l4_uint8_t writeback) [inline], [protected]
```

Sets cache mode and enables the writeback toggle.

Parameters

<i>writeback</i>	Mode of the cache (0 for writethrough, 1 for writeback).
------------------	--

Definition at line 375 of file [virtio-block](#).

16.400.3.5 set_discard()

```
template<typename Ds_data>
void L4virtio::Svr::Block_dev_base< Ds_data >::set_discard (
    14_uint32_t max_discard_sectors,
    14_uint32_t max_discard_seg,
    14_uint32_t discard_sector_alignment) [inline], [protected]
```

Sets constraints for and enables the discard command.

Parameters

<i>max_discard_sectors</i>	Maximum discard sectors size.
<i>max_discard_seg</i>	Maximum discard segment number.
<i>discard_sector_alignment</i>	Can be used by the driver when splitting a request based on alignment.

Definition at line 402 of file [virtio-block](#).

16.400.3.6 set_size_max()

```
template<typename Ds_data>
void L4virtio::Svr::Block_dev_base< Ds_data >::set_size_max (
    14_uint32_t sz) [inline], [protected]
```

Sets the maximum size of any single segment reported to client.

The limit is also applied to any incoming requests. Requests with larger segments result in an IO error being reported to the client. That means that `process_request()` can safely make the assumption that all segments in the received request are smaller.

Definition at line 290 of file [virtio-block](#).

16.400.3.7 set_topology()

```
template<typename Ds_data>
void L4virtio::Svr::Block_dev_base< Ds_data >::set_topology (
    14_uint8_t physical_block_exp,
    14_uint8_t alignment_offset,
    14_uint32_t min_io_size,
    14_uint32_t opt_io_size) [inline], [protected]
```

Sets the I/O alignment information reported back to the client.

Parameters

<i>physical_block_exp</i>	Number of logical blocks per physical block(log2)
<i>alignment_offset</i>	Offset of the first aligned logical block
<i>min_io_size</i>	Suggested minimum I/O size in blocks
<i>opt_io_size</i>	Optimal I/O size in blocks

Definition at line 348 of file [virtio-block](#).

16.400.3.8 set_write_zeroes()

```
template<typename Ds_data>
void L4virtio::Svr::Block_dev_base< Ds_data >::set_write_zeroes (
    14_uint32_t max_write_zeroes_sectors,
    14_uint32_t max_write_zeroes_seg,
    14_uint8_t write_zeroes_may_unmap) [inline], [protected]
```

Sets constraints for and enables the write zeroes command.

Parameters

<i>max_write_zeroes_sectors</i>	Maximum write zeroes sectors size.
<i>max_write_zeroes_seg</i>	maximum write zeroes segment number.
<i>write_zeroes_may_unmap</i>	Set if a write zeroes request can result in deallocating one or more sectors.

Definition at line 422 of file [virtio-block](#).

The documentation for this class was generated from the following file:

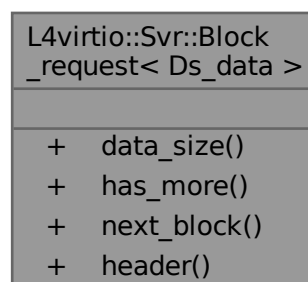
- l4/l4virtio/server/virtio-block

16.401 L4virtio::Svr::Block_request< Ds_data > Class Template Reference

A request to read or write data.

```
#include <virtio-block>
```

Collaboration diagram for L4virtio::Svr::Block_request< Ds_data >:



Public Member Functions

- unsigned [data_size](#) () const
Compute the total size of the data in the request.
- bool **has_more** ()
Check if the request contains more data blocks.
- Data_block [next_block](#) ()
Return next block in scatter-gather list.
- [l4virtio_block_header_t](#) const & **header** () const
Return the block request header.

16.401.1 Detailed Description

```
template<typename Ds_data>
class L4virtio::Svr::Block_request< Ds_data >
```

A request to read or write data.

Definition at line 28 of file [virtio-block](#).

16.401.2 Member Function Documentation

16.401.2.1 data_size()

```
template<typename Ds_data>
unsigned L4virtio::Svr::Block_request< Ds_data >::data_size () const [inline]
```

Compute the total size of the data in the request.

Return values

Size	in bytes or 0 if there was an error.
------	--------------------------------------

Exceptions

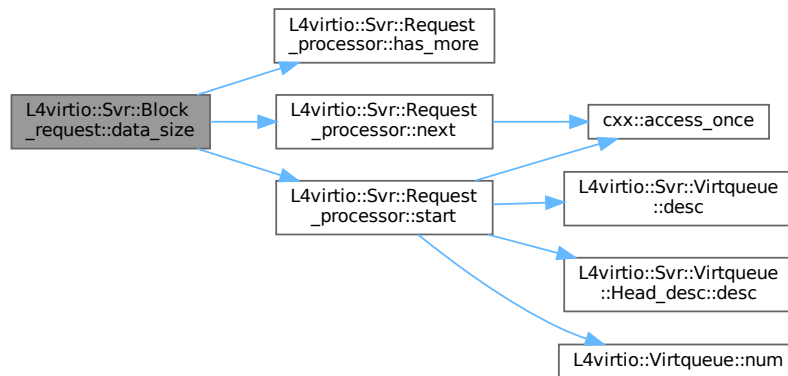
L4::Runtime_error(-L4_EIO)	Request has a bad format.
--	---------------------------

Note that this operation is relatively expensive as it has to iterate over the complete list of blocks.

Definition at line 63 of file [virtio-block](#).

References [L4virtio::Svr::Request_processor::has_more\(\)](#), [L4_EIO](#), [L4virtio::Svr::Request_processor::next\(\)](#), and [L4virtio::Svr::Request_processor::start\(\)](#).

Here is the call graph for this function:



16.401.2.2 next_block()

```
template<typename Ds_data>
Data_block L4virtio::Svr::Block_request< Ds_data >::next_block () [inline]
```

Return next block in scatter-gather list.

Returns

Information about the next data block.

Exceptions

<i>L4::Runtime_error</i>	No more data block is available.
<i>Bad_descriptor</i>	Virtio request is corrupted.

Definition at line 113 of file [virtio-block](#).

References [L4virtio::Svr::Bad_descriptor::Bad_size](#), and [L4_EEXIST](#).

The documentation for this class was generated from the following file:

- [l4/l4virtio/server/virtio-block](#)

16.402 L4virtio::Svr::Console::Control_message Struct Reference

Virtio console control message.

```
#include <virtio-console>
```

Collaboration diagram for L4virtio::Svr::Console::Control_message:

L4virtio::Svr::Console ::Control_message	
+	id
+	event
+	value

Public Types

- enum [Events](#) {
[Device_ready](#) = 0 , [Device_add](#) = 1 , [Device_remove](#) = 2 , [Port_ready](#) = 3 ,
[Console_port](#) = 4 , [Resize](#) = 5 , [Port_open](#) = 6 , [Port_name](#) = 7 }

Possible control events.

Data Fields

- [l4_uint32_t](#) **id**
Port number.
- [l4_uint16_t](#) **event**
Control event, see [Events](#).
- [l4_uint16_t](#) **value**
Extra information.

16.402.1 Detailed Description

Virtio console control message.

Definition at line [31](#) of file [virtio-console](#).

16.402.2 Member Enumeration Documentation

16.402.2.1 Events

```
enum L4virtio::Svr::Console::Control_message::Events
```

Possible control events.

Enumerator

Device_ready	Sent by driver at initialization.
Device_add	Sent by device to create new ports.
Device_remove	Sent by device to remove added ports.
Port_ready	Sent by driver as response to Device_add .
Console_port	Sent by device to nominate port as console port.
Resize	Sent by device to indicate a console size change.
Port_open	Sent by device and driver to indicate whether a port is open.
Port_name	Sent by device to tag a port.

Definition at line 34 of file [virtio-console](#).

The documentation for this struct was generated from the following file:

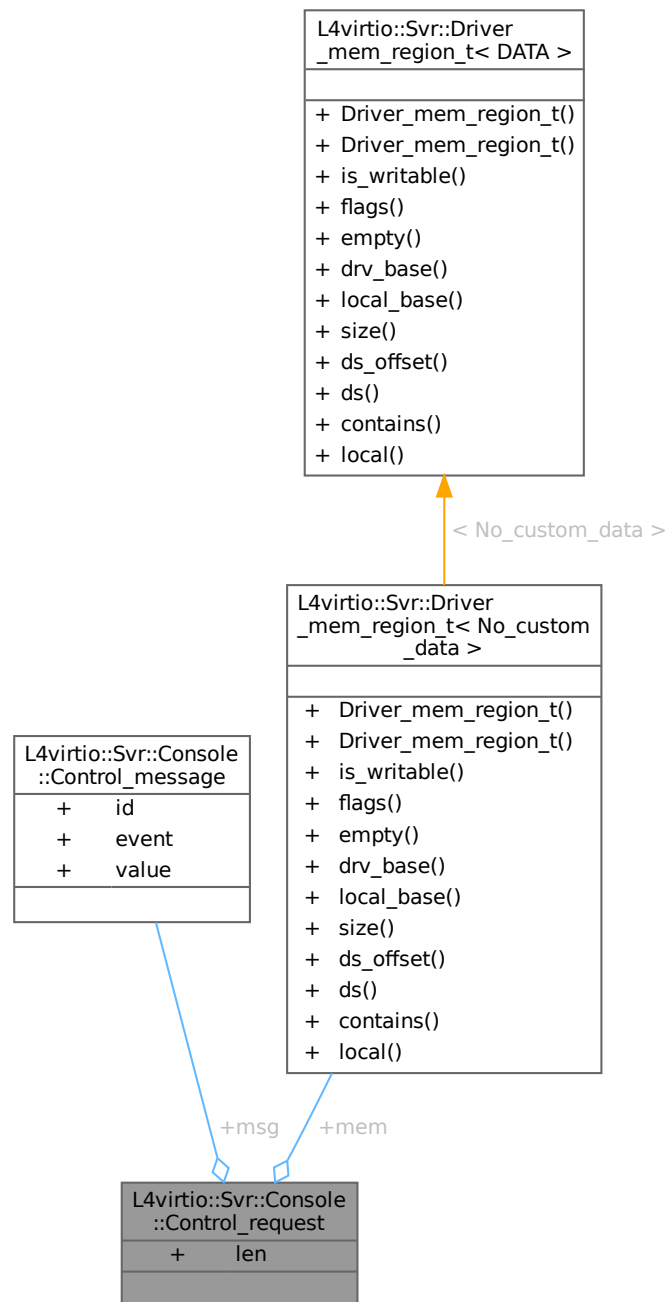
- l4/l4virtio/server/virtio-console

16.403 L4virtio::Svr::Console::Control_request Struct Reference

Specialised `Virtqueue::Request` providing access to control message payload.

```
#include <virtio-console>
```

Collaboration diagram for L4virtio::Svr::Console::Control_request:



Data Fields

- **Control_message * msg**
Virtual address of the data block (in device space).
- **l4_uint32_t len**
Length of datablock in bytes.
- **Driver_mem_region * mem**
Pointer to driver memory region.

16.403.1 Detailed Description

Specialised `Virtqueue::Request` providing access to control message payload.

Definition at line 65 of file [virtio-console](#).

The documentation for this struct was generated from the following file:

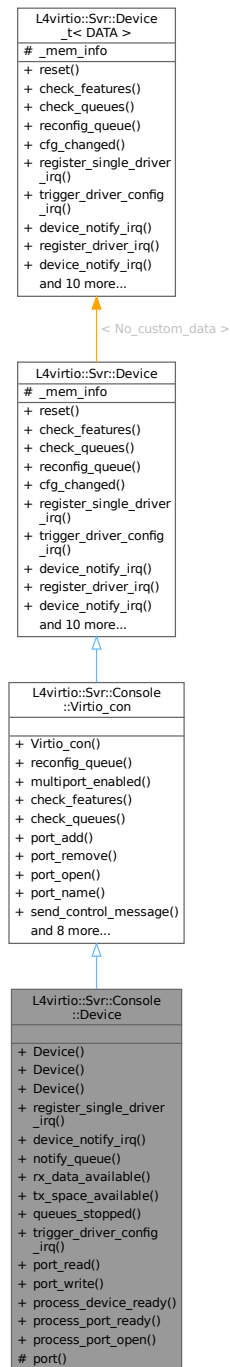
- `l4/l4virtio/server/virtio-console`

16.404 L4virtio::Svr::Console::Device Class Reference

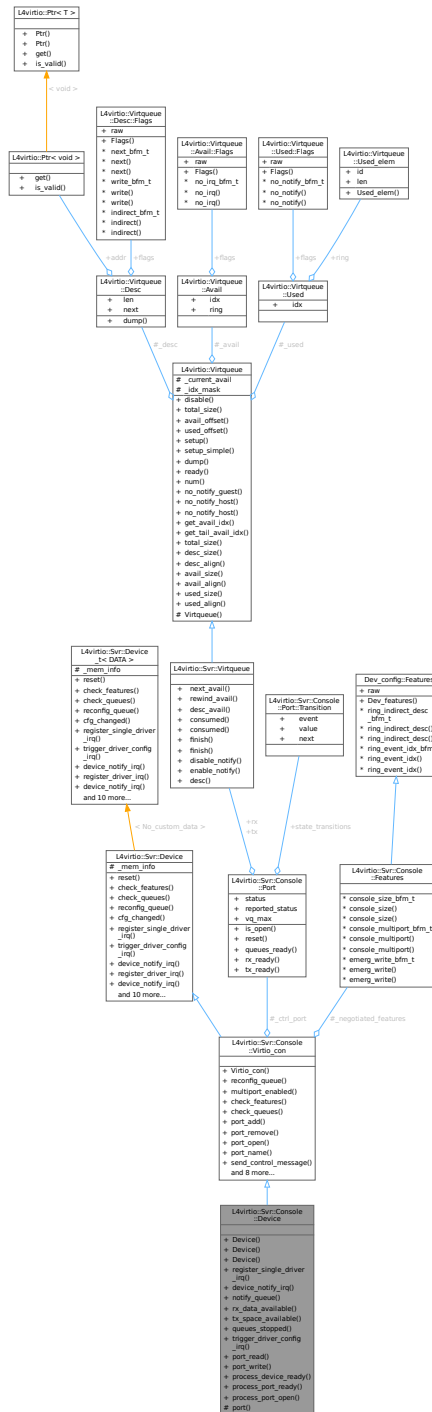
Base class implementing a virtio console device with L4Re-based notification handling.

```
#include <virtio-console-device>
```

Inheritance diagram for L4virtio::Svr::Console::Device:



Collaboration diagram for L4virtio::Svr::Console::Device:



Public Member Functions

- **Device** (unsigned vq_max)
Create a new console device.
- **Device** (unsigned vq_max, unsigned ports)
Create a new console device.
- **Device** (cxx::static_vector< unsigned > const &vq_max_nums)

- Create a new console [Device](#).
- void **register_single_driver_irq** () override
callback for registering a single guest IRQ for all queues (old-style)
- [L4::Cap](#)< [L4::Irq](#) > **device_notify_irq** () const override
callback to gather the device notification IRQ (old-style)
- void **notify_queue** ([Virtqueue](#) *queue) override
Notify queue of available data.
- virtual void **rx_data_available** (unsigned [port](#))=0
Callback to notify that new data is available to be read from port.
- virtual void **tx_space_available** (unsigned [port](#))=0
Callback to notify that data can be written to port.
- virtual bool **queues_stopped** ()
Return true, if the queues should not be processed further.
- void **trigger_driver_config_irq** () override
callback for triggering configuration change notification IRQ
- unsigned **port_read** (char *buf, unsigned len, unsigned [port](#)=0)
Read data from port.
- unsigned **port_write** (char const *buf, unsigned len, unsigned [port](#)=0)
Write data to port.
- void **process_device_ready** ([l4_uint16_t](#) value) override
Callback called on `DEVICE_READY` event.
- void **process_port_ready** ([l4_uint32_t](#) id, [l4_uint16_t](#) value) override
Callback called on `PORT_READY` event.
- virtual void **process_port_open** ([l4_uint32_t](#) id, [l4_uint16_t](#) value)
Callback called on `PORT_OPEN` event.

Public Member Functions inherited from [L4virtio::Svr::Console::Virtio_con](#)

- [Virtio_con](#) (unsigned max_ports, bool enable_multiport)
Create a new multiport console device.
- int **reconfig_queue** (unsigned index) override
callback for client queue-config request
- bool **multiport_enabled** () const
Return true if the multiport feature is enabled and control queues are available.
- bool **check_features** (void) override
callback for checking the subset of accepted features
- bool **check_queues** () override
callback for checking if the queues at `DRIVER_OK` transition
- int **port_add** (unsigned idx)
Send a `DEVICE_ADD` message and update the internal state.
- int **port_remove** (unsigned idx)
Send a `DEVICE_REMOVE` message and update the internal state.
- int **port_open** (unsigned idx, bool open)
Send a `PORT_OPEN` message and update the internal state.
- int **port_name** (unsigned idx, char const *name)
Send a `PORT_NAME` message to announce the port name.
- int **send_control_message** ([l4_uint32_t](#) idx, [l4_uint16_t](#) event, [l4_uint16_t](#) value=0, const char *name=0)
Send control message to driver.
- int **handle_control_message** ()
Handle control message received from the driver.
- void **reset** () override
reset callback, called for doing a device reset
- virtual void **reset_device** ()
Reset the state of the actual console device.

Public Member Functions inherited from L4virtio::Svr::Device_t< No_custom_data >

- virtual void **cfg_changed** (unsigned)
callback for client device configuration changes
- virtual void **register_driver_irq** (unsigned idx)
Callback for registering an notification IRQ (multi IRQ).
- virtual L4::Cap< L4::Irq > **device_notify_irq** (unsigned idx)
Callback to gather the device notification IRQ (multi IRQ).
- virtual unsigned **num_events_supported** () const
Return the highest notification index supported.
- **Device_t** (Dev_config *dev_config)
Make a device for the given config.
- **Mem_list** const * **mem_info** () const
Get the memory region list used for this device.
- void **reset_queue_config** (unsigned idx, unsigned num_max, bool inc_generation=false)
Trigger reset for the configuration space for queue idx.
- void **init_mem_info** (unsigned num)
Initialize the memory region list to the given maximum.
- void **device_error** ()
Transition device into DEVICE_NEEDS_RESET state.
- bool **setup_queue** (Virtqueue *q, unsigned qn, unsigned num_max)
Enable/disable the specified queue.
- bool **handle_mem_cmd_write** ()
Check for a value in the cmd register and handle a write.
- void **enable_trusted_ds_validation** ()
Enable trusted dataspace validation.
- void **add_trusted_dataspaces** (std::shared_ptr< Ds_vector const > ds)
Provide a list of trusted dataspaces that can be used for validation.

Protected Member Functions

- **Port** * **port** (unsigned idx) override
Return the specified port.

Additional Inherited Members

Protected Attributes inherited from L4virtio::Svr::Device_t< No_custom_data >

- **Mem_list** **mem_info**
Memory region list.

16.404.1 Detailed Description

Base class implementing a virtio console device with L4Re-based notification handling.

This console device is derived from [Virtio_con](#) and already includes functionality to handle interrupts and notify drivers. If an interrupt is received, all the necessary interaction with the virtqueues is performed and only the actual data processing has to be done by the derived class. By default all available ports are added and an "open"-request of a port by the driver is automatically acknowledged. The derived class can optionally change this behaviour by overriding [process_device_ready\(\)](#), [process_port_ready\(\)](#) and [process_port_open\(\)](#).

This class provides a stream-based interface to access the port data with edge-triggered notification callbacks. If a port receives data from the driver the derived class is notified with the [rx_data_available\(\)](#) callback. The actual data can be retrieved by [port_read\(\)](#). If there was not enough data to be read, the call will return the available partial data. Only then will the [rx_data_available\(\)](#) callback be triggered again.

Data on a port may be transmitted by [port_write\(\)](#). If there were not enough buffers available, only a part of the data will be transmitted. Once there are new buffers available, the [tx_space_available\(\)](#) callback will be invoked. This callback will be called again only after a previous [port_write\(\)](#) was not able to send all requested data.

Use this class as a base to provide your own high-level console device. You must derive from this class as well as [L4::Epiface_t<..., L4virtio::Device>](#). For a working device the [irq_iface\(\)](#) must be registered too. A typical implementation might look like the following:

```
class My_console
: public L4virtio::Svr::Console::Device,
  public L4::Epiface_t<My_console, L4virtio::Device>
{
public:
    My_console(L4Re::Util::Object_registry *r)
        : L4virtio::Svr::Console::Device(0x100)
        {
            init_mem_info(4);
            L4Re::chkcap(r->register_irq_obj(irq_iface()), "virtio notification IRQ");
        }

    void rx_data_available(unsigned port) override
    {
        // call port_read() to fetch available data
    }

    void tx_space_available(unsigned port) override
    {
        // can call port_write() to send (pending) data
    }
};

My_console console(registry);
registry->register_obj(&console, ...);
```

The maximum number of memory regions ([init_mem_info\(\)](#)) should correlate with the number of supported ports.

Definition at line 118 of file [virtio-console-device](#).

16.404.2 Constructor & Destructor Documentation

16.404.2.1 Device() [1/3]

```
L4virtio::Svr::Console::Device::Device (
    unsigned vq_max) [inline], [explicit]
```

Create a new console device.

Parameters

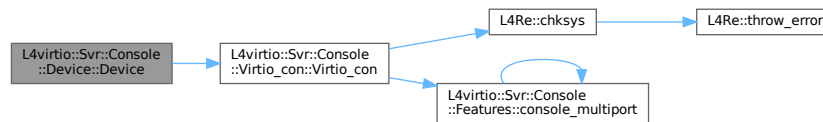
<i>vq_max</i>	Maximum number of buffers in data queues.
---------------	---

Create a console device with no multiport support, i.e. control queues are disabled.

Definition at line 145 of file [virtio-console-device](#).

References [L4virtio::Svr::Console::Virtio_con::Virtio_con\(\)](#).

Here is the call graph for this function:



16.404.2.2 Device() [2/3]

```
L4virtio::Svr::Console::Device::Device (
    unsigned vq_max,
    unsigned ports) [inline], [explicit]
```

Create a new console device.

Parameters

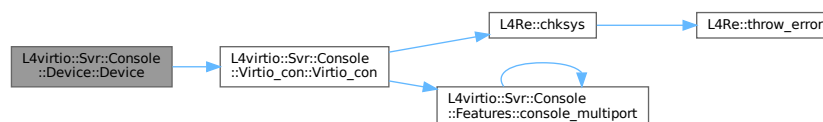
<i>vq_max</i>	Maximum number of buffers in data queues.
<i>ports</i>	Number of ports (maximum 32).

Create a console device with multiport support, i.e. control queues are enabled.

Definition at line 163 of file [virtio-console-device](#).

References [L4virtio::Svr::Console::Virtio_con::Virtio_con\(\)](#).

Here is the call graph for this function:



16.404.2.3 Device() [3/3]

```
L4virtio::Svr::Console::Device::Device (
    cxx::static_vector< unsigned > const & vq_max_nums) [inline], [explicit]
```

Create a new console [Device](#).

Parameters

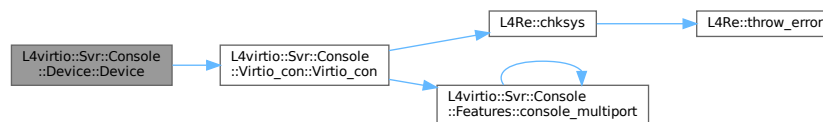
<code>vq_max_nums</code>	Maximum number of buffers in data queues, given as a <code>cx::static_vector</code> with one entry per port.
--------------------------	--

Create a console device with multiport support, i.e. control queues are enabled.

Definition at line 182 of file `virtio-console-device`.

References `L4virtio::Svr::Console::Virtio_con::Virtio_con()`.

Here is the call graph for this function:



16.404.3 Member Function Documentation

16.404.3.1 notify_queue()

```
void L4virtio::Svr::Console::Device::notify_queue (
    Virtqueue * queue) [inline], [override], [virtual]
```

Notify queue of available data.

Parameters

<code>queue</code>	<code>Virtqueue</code> to notify.
--------------------	-----------------------------------

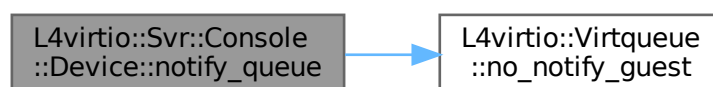
This callback is called whenever data is sent to `queue`. It is the responsibility of the derived class to perform all necessary notification actions, e.g. triggering guest interrupts.

Implements `L4virtio::Svr::Console::Virtio_con`.

Definition at line 202 of file `virtio-console-device`.

References `L4VIRTIO_IRQ_STATUS_VRING`, and `L4virtio::Virtqueue::no_notify_guest()`.

Here is the call graph for this function:



16.404.3.2 port()

```
Port * L4virtio::Svr::Console::Device::port (
    unsigned port) [inline], [override], [protected], [virtual]
```

Return the specified port.

Parameters

<i>port</i>	Port number.
-------------	--------------

Precondition

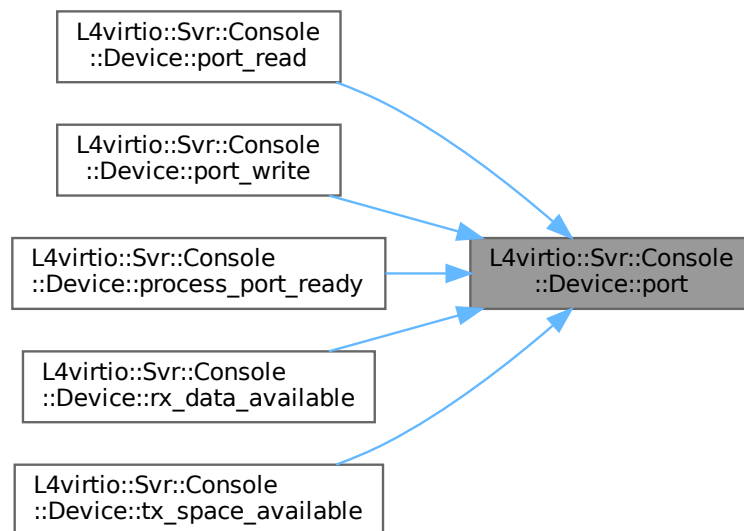
Port number must be lower than the configured maximum number of ports.

Implements [L4virtio::Svr::Console::Virtio_con](#).

Definition at line 450 of file [virtio-console-device](#).

Referenced by [port_read\(\)](#), [port_write\(\)](#), [process_port_ready\(\)](#), [rx_data_available\(\)](#), and [tx_space_available\(\)](#).

Here is the caller graph for this function:



16.404.3.3 port_read()

```
unsigned L4virtio::Svr::Console::Device::port_read (  
    char * buf,  
    unsigned len,  
    unsigned port = 0) [inline]
```

Read data from port.

Will read up to *len* bytes from *port* into *buf*. Returns the number of bytes read, which may be less if not enough data was available. If all data was read, the [rx_data_available\(\)](#) callback will be invoked the next time the driver queues new data for the port. The callback won't be called again until all data was consumed again.

Parameters

<i>buf</i>	The destination buffer
<i>len</i>	Size of the buffer
<i>port</i>	Port index to read data from

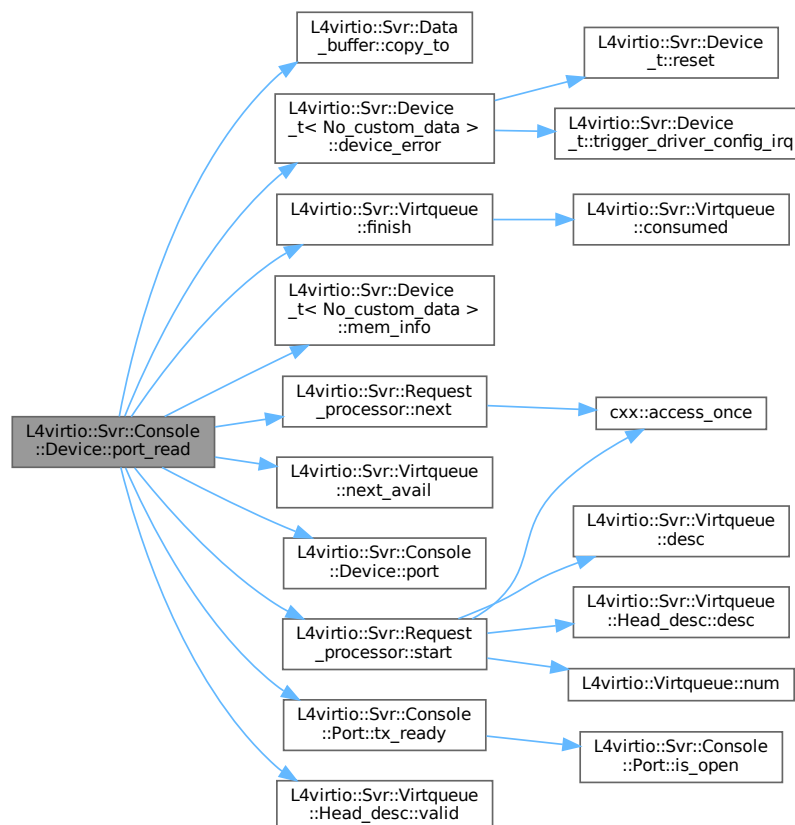
Returns

Number of bytes read

Definition at line 272 of file [virtio-console-device](#).

References [L4virtio::Svr::Data_buffer::copy_to\(\)](#), [L4virtio::Svr::Device_t< No_custom_data >::device_error\(\)](#), [L4virtio::Svr::Virtqueue::finish\(\)](#), [L4virtio::Svr::Data_buffer::left](#), [L4virtio::Svr::Device_t< No_custom_data >::mem_info\(\)](#), [L4virtio::Svr::Request_processor::next\(\)](#), [L4virtio::Svr::Virtqueue::next_avail\(\)](#), [port\(\)](#), [L4virtio::Svr::Data_buffer::pos](#), [L4virtio::Svr::Console::Device_port::request](#), [L4virtio::Svr::Console::Device_port::rp](#), [L4virtio::Svr::Console::Device_port::src](#), [L4virtio::Svr::Request_processor::start\(\)](#), [L4virtio::Svr::Console::Port::tx](#), [L4virtio::Svr::Console::Port::tx_ready\(\)](#), and [L4virtio::Svr::Virtqueue::Head_desc::valid\(\)](#).

Here is the call graph for this function:



16.404.3.4 port_write()

```
unsigned L4virtio::Svr::Console::Device::port_write (
    char const * buf,
    unsigned len,
    unsigned port = 0) [inline]
```

Write data to port.

Will write up to *len* bytes to *port* from *buf*. Returns the number of bytes written, which may be less if not enough virtio buffers were available. If not all data could be written, the [tx_space_available\(\)](#) callback will be invoked the next time the driver queues new receive buffers for the port. The callback won't be called again until all receive buffers were filled again.

Parameters

<i>buf</i>	The source buffer
<i>len</i>	Size of the buffer
<i>port</i>	Port index to write data to

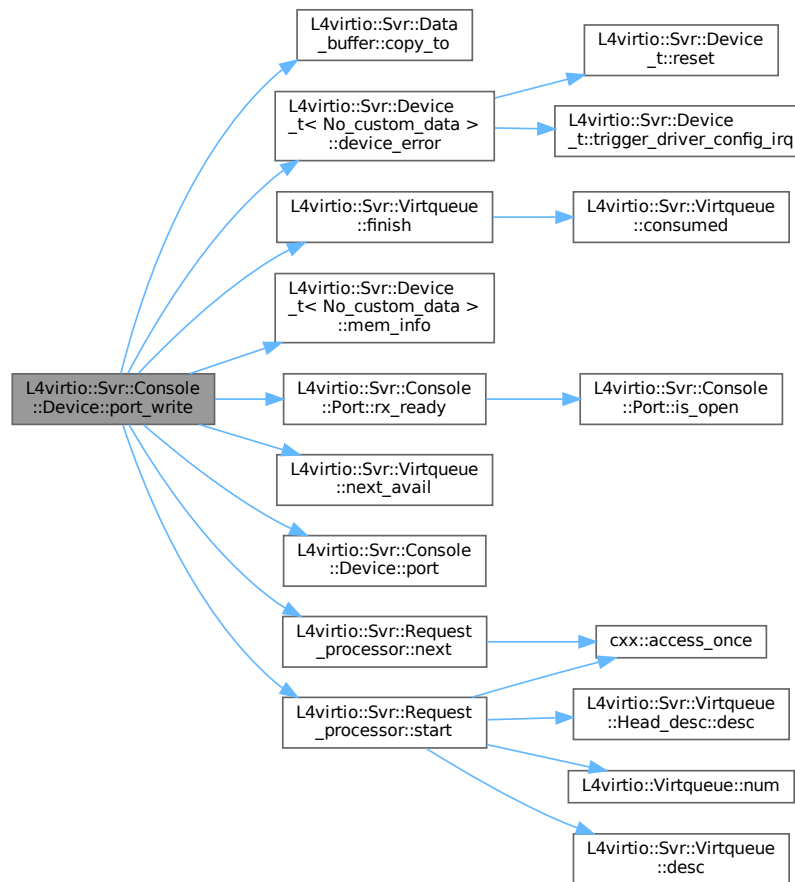
Returns

Number of bytes written

Definition at line [341](#) of file [virtio-console-device](#).

References [L4virtio::Svr::Data_buffer::copy_to\(\)](#), [L4virtio::Svr::Device_t< No_custom_data >::device_error\(\)](#), [L4virtio::Svr::Virtqueue::finish\(\)](#), [L4virtio::Svr::Data_buffer::left](#), [L4virtio::Svr::Device_t< No_custom_data >::mem_info\(\)](#), [L4virtio::Svr::Request_processor::next\(\)](#), [L4virtio::Svr::Virtqueue::next_avail\(\)](#), [port\(\)](#), [L4virtio::Svr::Data_buffer::pos](#), [L4virtio::Svr::Console::Port::rx](#), [L4virtio::Svr::Console::Port::rx_ready\(\)](#), and [L4virtio::Svr::Request_processor::start\(\)](#).

Here is the call graph for this function:



16.404.3.5 process_device_ready()

```
void L4virtio::Svr::Console::Device::process_device_ready (
    uint16_t value) [inline], [override], [virtual]
```

Callback called on DEVICE_READY event.

Parameters

<i>value</i>	The value field of the control message, indicating if the initialization was successful.
--------------	--

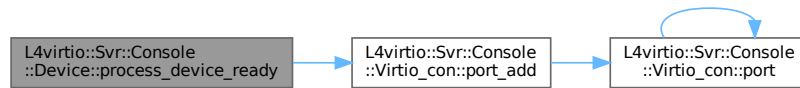
By default, this function adds all ports if the driver indicates successful initialization. Override this function to perform custom actions for a DEVICE_READY event. It is then your responsibility to inform the driver about usable ports, see [port_add\(\)](#).

Implements [L4virtio::Svr::Console::Virtio_con](#).

Definition at line 399 of file [virtio-console-device](#).

References [L4virtio::Svr::Console::Virtio_con::port_add\(\)](#).

Here is the call graph for this function:



16.404.3.6 process_port_open()

```
virtual void L4virtio::Svr::Console::Device::process_port_open (
    14_uint32_t id,
    14_uint16_t value) [inline], [virtual]
```

Callback called on PORT_OPEN event.

Parameters

<i>id</i>	The id field of the control message, i.e. the port number.
<i>value</i>	The value field of the control message, indicating if the port was opened or closed.

The default implementation does nothing. Override it to implement some custom logic to respond to open/close events of the driver.

Implements [L4virtio::Svr::Console::Virtio_con](#).

Definition at line 443 of file [virtio-console-device](#).

16.404.3.7 process_port_ready()

```
void L4virtio::Svr::Console::Device::process_port_ready (
    14_uint32_t id,
    14_uint16_t value) [inline], [override], [virtual]
```

Callback called on PORT_READY event.

Parameters

<i>id</i>	The id field of the control message, i.e. the port number.
<i>value</i>	The value field of the control message, indicating if the initialization was successful.

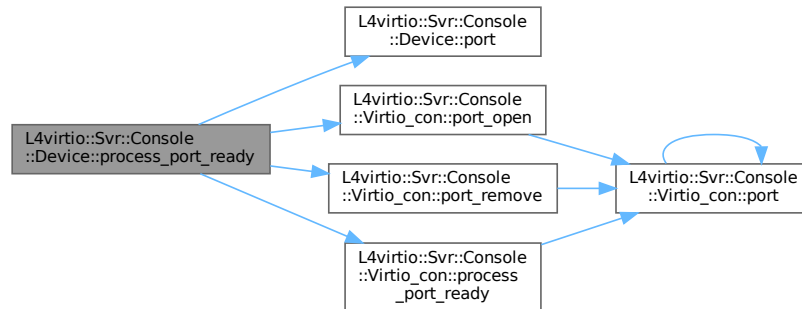
By default, this function opens the port if the driver is ready. Otherwise, the port is removed if the driver failed to set it up correctly. Override this function to perform custom actions for a PORT_READY event, *after* the generic management of the base class. It is then your responsibility to inform the driver about connected or unusable ports. See [port_open\(\)](#) and [port_remove\(\)](#).

Reimplemented from [L4virtio::Svr::Console::Virtio_con](#).

Definition at line 422 of file [virtio-console-device](#).

References [port\(\)](#), [L4virtio::Svr::Console::Port::Port_failed](#), [L4virtio::Svr::Console::Virtio_con::port_open\(\)](#), [L4virtio::Svr::Console::Port::Port_ready](#), [L4virtio::Svr::Console::Virtio_con::port_remove\(\)](#), [L4virtio::Svr::Console::Virtio_con::process_port_ready](#) and [L4virtio::Svr::Console::Port::status](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

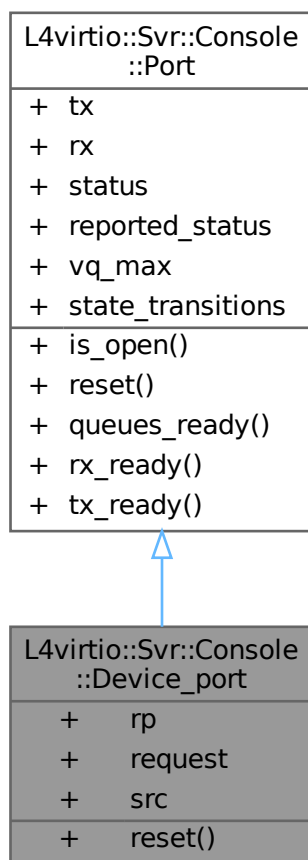
- `l4/l4virtio/server/virtio-console-device`

16.405 L4virtio::Svr::Console::Device_port Struct Reference

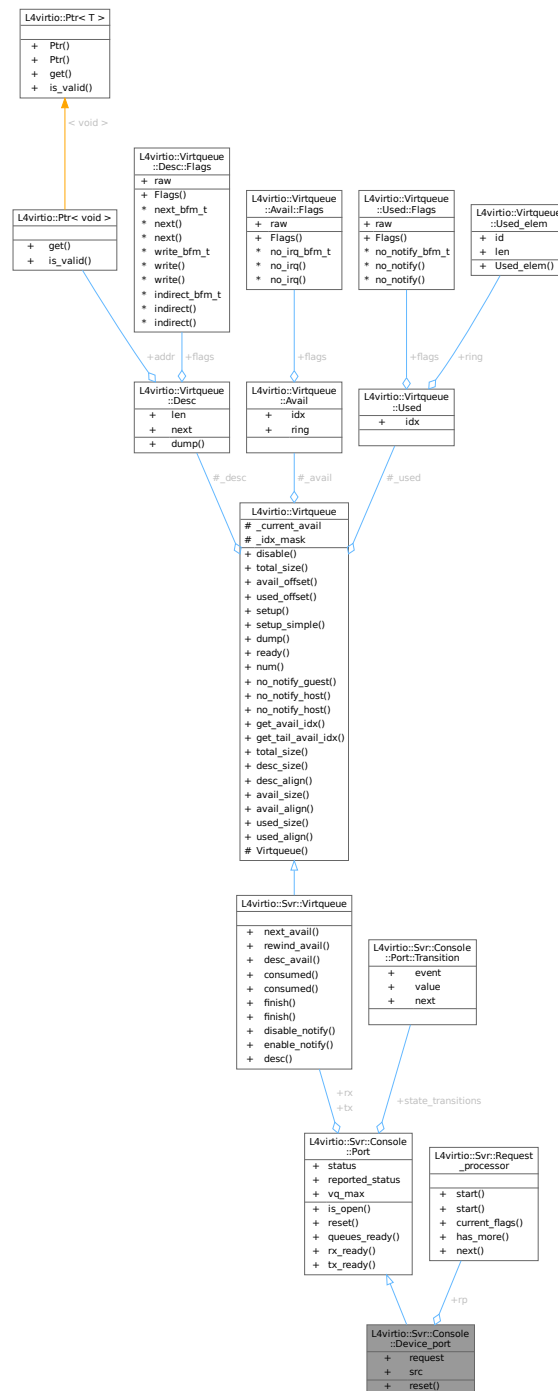
A console port with associated read/write state.

```
#include <virtio-console-device>
```

Inheritance diagram for L4virtio::Svr::Console::Device_port:



Collaboration diagram for L4virtio::Svr::Console::Device_port:



Public Member Functions

- void **reset** () override
Reset the port to the initial state and disable its virtqueues.

Public Member Functions inherited from L4virtio::Svr::Console::Port

- bool **is_open** () const

- Check that the port is open.*

 - bool **queues_ready** () const

Check that both virtqueues are set up correctly.
- bool **rx_ready** () const

Check that device implementation may write to receive queues.
- bool **tx_ready** () const

Check that device implementation may read from transmit queues.

Data Fields

- [Request_processor](#) **rp**
- Request processor associated with current request.*
- Virtqueue::Request **request**
- Current virtio tx queue request.*
- [Buffer](#) **src**
- Source data block to process.*

Data Fields inherited from [L4virtio::Svr::Console::Port](#)

- [Virtqueue](#) **tx**
- Receiveq of the port.*
- [Virtqueue](#) **rx**
- Transmitq of the port.*
- [Port_status](#) **status**
- State the port is in.*
- [Port_status](#) **reported_status**
- State the port was last reported.*
- unsigned **vq_max**
- Maximum queue sizes for this port.*

Additional Inherited Members

Public Types inherited from [L4virtio::Svr::Console::Port](#)

- enum [Port_status](#) {
[Port_disabled](#) = 0 , [Port_added](#) , [Port_ready](#) , [Port_open](#) ,
[Port_failed](#) , [Port_num_states](#) }
- Possible states of a virtio console port.*
- enum
- Size of control queues, also used as default size.*

Static Public Attributes inherited from [L4virtio::Svr::Console::Port](#)

- static constexpr [Transition](#) **state_transitions** [[Port_num_states](#)][[Port_num_states](#)]
- State transition table from last report state to current state.*

16.405.1 Detailed Description

A console port with associated read/write state.

Tracks the notification of the device implementation and holds the state when receiving data from the driver.

Definition at line 26 of file [virtio-console-device](#).

The documentation for this struct was generated from the following file:

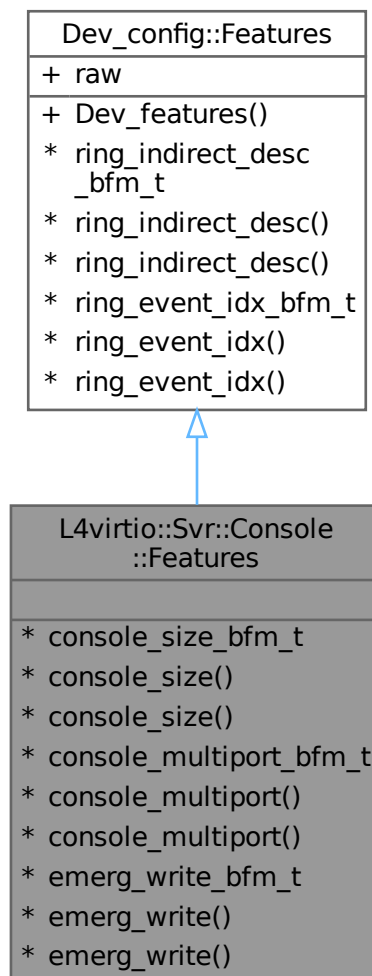
- l4/l4virtio/server/virtio-console-device

16.406 L4virtio::Svr::Console::Features Struct Reference

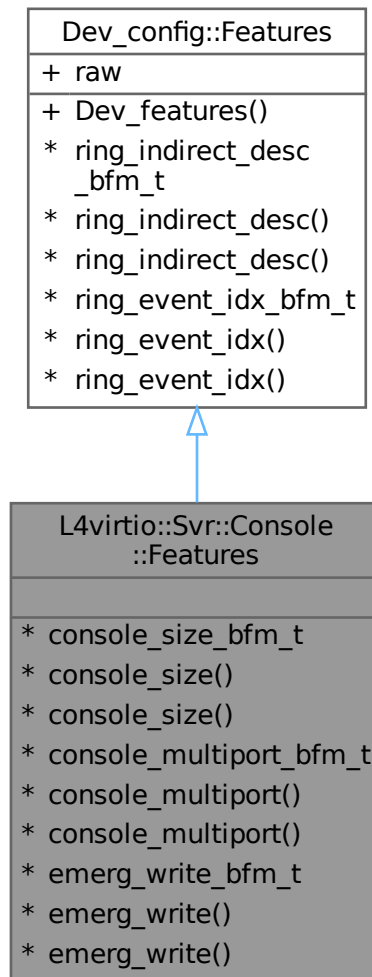
Virtio console specific feature bits.

```
#include <virtio-console>
```

Inheritance diagram for L4virtio::Svr::Console::Features:



Collaboration diagram for L4virtio::Svr::Console::Features:



- typedef `cxx::Bitfield< decltype(raw), 0, 0 >` `console_size_bfm_t`
Configuration `cols` and `rows` are valid.
- constexpr `console_size_bfm_t::Val console_size ()` const
Get the `console_size` bits (0 to 0) of `raw`.
- constexpr `console_size_bfm_t::Ref console_size ()`
Get a reference to the `console_size` bits (0 to 0) of `raw`.
- typedef `cxx::Bitfield< decltype(raw), 1, 1 >` `console_multiport_bfm_t`
`Device` has support for multiple ports.
- constexpr `console_multiport_bfm_t::Val console_multiport ()` const
Get the `console_multiport` bits (1 to 1) of `raw`.
- constexpr `console_multiport_bfm_t::Ref console_multiport ()`
Get a reference to the `console_multiport` bits (1 to 1) of `raw`.

- typedef [cxx::Bitfield](#)< decltype([raw](#)), 2, 2 > [emerg_write_bfm_t](#)
Device has support for emergency write.
- constexpr [emerg_write_bfm_t::Val](#) [emerg_write](#) () const
Get the [emerg_write](#) bits (2 to 2) of [raw](#).
- constexpr [emerg_write_bfm_t::Ref](#) [emerg_write](#) ()
Get a reference to the [emerg_write](#) bits (2 to 2) of [raw](#).

Additional Inherited Members

- typedef [cxx::Bitfield](#)< decltype([raw](#)), 28, 28 > [ring_indirect_desc_bfm_t](#)
Type to access the [ring_indirect_desc](#) bits (28 to 28) of [raw](#).
- typedef [cxx::Bitfield](#)< decltype([raw](#)), 29, 29 > [ring_event_idx_bfm_t](#)
Type to access the [ring_event_idx](#) bits (29 to 29) of [raw](#).

Public Member Functions inherited from [L4virtio::Svr::Dev_features](#)

- [Dev_features](#) ([l4_uint32_t](#) v)
Make Features from a raw bitmap.
- constexpr [ring_indirect_desc_bfm_t::Val](#) [ring_indirect_desc](#) () const
Get the [ring_indirect_desc](#) bits (28 to 28) of [raw](#).
- constexpr [ring_indirect_desc_bfm_t::Ref](#) [ring_indirect_desc](#) ()
Get a reference to the [ring_indirect_desc](#) bits (28 to 28) of [raw](#).
- constexpr [ring_event_idx_bfm_t::Val](#) [ring_event_idx](#) () const
Get the [ring_event_idx](#) bits (29 to 29) of [raw](#).
- constexpr [ring_event_idx_bfm_t::Ref](#) [ring_event_idx](#) ()
Get a reference to the [ring_event_idx](#) bits (29 to 29) of [raw](#).

Data Fields inherited from [L4virtio::Svr::Dev_features](#)

- [l4_uint32_t](#) [raw](#)
The raw value of the features bitmap.

16.406.1 Detailed Description

Virtio console specific feature bits.

Definition at line 18 of file [virtio-console](#).

16.406.2 Member Typedef Documentation

16.406.2.1 console_multiport_bfm_t

```
typedef cxx::Bitfield<decltype(raw), 1, 1> L4virtio::Svr::Console::Features::console_multiport_bfm_t
```

[Device](#) has support for multiple ports.

Type to access the [console_multiport](#) bits (1 to 1) of [raw](#).

Definition at line [25](#) of file [virtio-console](#).

16.406.2.2 console_size_bfm_t

```
typedef cxx::Bitfield<decltype(raw), 0, 0> L4virtio::Svr::Console::Features::console_size_bfm_t
```

Configuration [cols](#) and [rows](#) are valid.

Type to access the [console_size](#) bits (0 to 0) of [raw](#).

Definition at line [23](#) of file [virtio-console](#).

16.406.2.3 emerg_write_bfm_t

```
typedef cxx::Bitfield<decltype(raw), 2, 2> L4virtio::Svr::Console::Features::emerg_write_bfm_t
```

[Device](#) has support for emergency write.

Type to access the [emerg_write](#) bits (2 to 2) of [raw](#).

Definition at line [27](#) of file [virtio-console](#).

The documentation for this struct was generated from the following file:

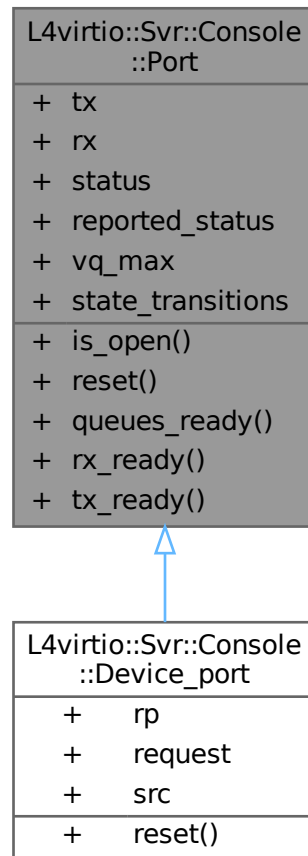
- [l4/l4virtio/server/virtio-console](#)

16.407 L4virtio::Svr::Console::Port Struct Reference

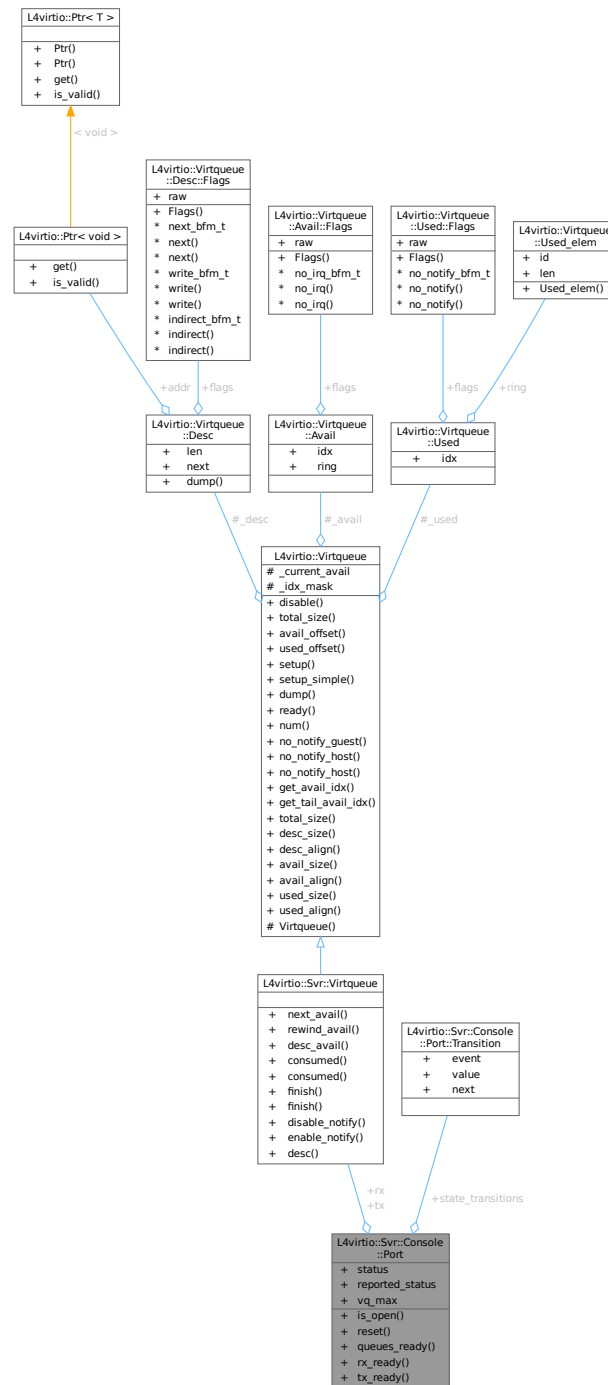
Representation of a Virtio console port.

```
#include <virtio-console>
```

Inheritance diagram for L4virtio::Svr::Console::Port:



Collaboration diagram for L4virtio::Svr::Console::Port:



Data Structures

- struct [Transition](#)

State transition from last report state to current state.

Public Types

- enum [Port_status](#) {
 [Port_disabled](#) = 0 , [Port_added](#) , [Port_ready](#) , [Port_open](#) ,
 [Port_failed](#) , [Port_num_states](#) }

Possible states of a virtio console port.

- enum
Size of control queues, also used as default size.

Public Member Functions

- bool **is_open** () const
Check that the port is open.
- virtual void **reset** ()
Reset the port to the initial state and disable its virtqueues.
- bool **queues_ready** () const
Check that both virtqueues are set up correctly.
- bool **rx_ready** () const
Check that device implementation may write to receive queues.
- bool **tx_ready** () const
Check that device implementation may read from transmit queues.

Data Fields

- [Virtqueue](#) **tx**
Receiveq of the port.
- [Virtqueue](#) **rx**
Transmitq of the port.
- [Port_status](#) **status**
State the port is in.
- [Port_status](#) **reported_status**
State the port was last reported.
- unsigned **vq_max**
Maximum queue sizes for this port.

Static Public Attributes

- static constexpr [Transition](#) **state_transitions** [[Port_num_states](#)][[Port_num_states](#)]
State transition table from last report state to current state.

16.407.3 Field Documentation

16.407.3.1 state_transitions

```
Transition L4virtio::Svr::Console::Port::state_transitions[Port_num_states][Port_num_states]  
[static], [constexpr]
```

State transition table from last report state to current state.

Not all transitions can be made directly. For example, if the last reported state was `Port_disabled` and the current state is `Port_open`, the device has to send two messages: `Control_message::Device_add` and `Control_message::Port_open`. This is expressed by going through an intermediate state (`Port_ready`) on the reporting side.

For the purpose of the driver there are only three coarse states:

1. The port does not exist (`Port_disabled`).
2. The port exists but is closed on the device side (`Port_added`, `Port_ready`, `Port_failed`).
3. The port exists and is open on the device side (`Port_open`).

The state transition table with `Port_added`, `Port_ready` and `Port_failed` as current state are thus identical.

Definition at line 195 of file `virtio-console`.

The documentation for this struct was generated from the following file:

- `l4/l4virtio/server/virtio-console`

16.408 L4virtio::Svr::Console::Port::Transition Struct Reference

State transition from last report state to current state.

```
#include <virtio-console>
```

Collaboration diagram for L4virtio::Svr::Console::Port::Transition:

L4virtio::Svr::Console ::Port::Transition	
+	event
+	value
+	next

Data Fields

- [l4_int16_t](#) **event**
[Control_message::Events](#) or < 0 if no event is sent.
- [l4_uint16_t](#) **value**
Extra information.
- [Port_status](#) **next**
Next [Port_status](#) state.

16.408.1 Detailed Description

State transition from last report state to current state.

Definition at line 169 of file [virtio-console](#).

The documentation for this struct was generated from the following file:

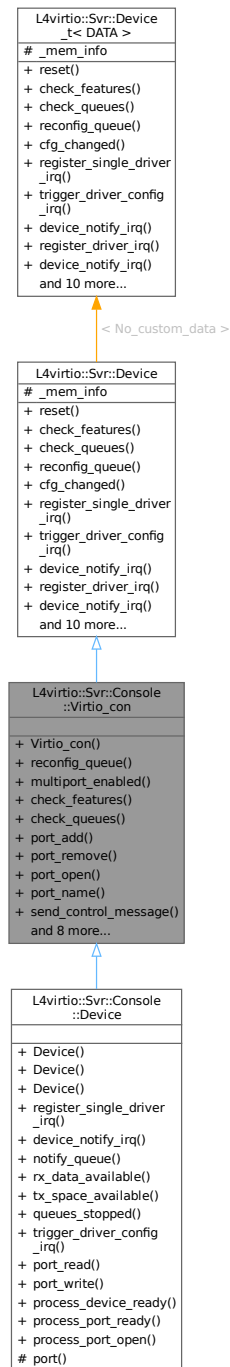
- l4/l4virtio/server/virtio-console

16.409 L4virtio::Svr::Console::Virtio_con Class Reference

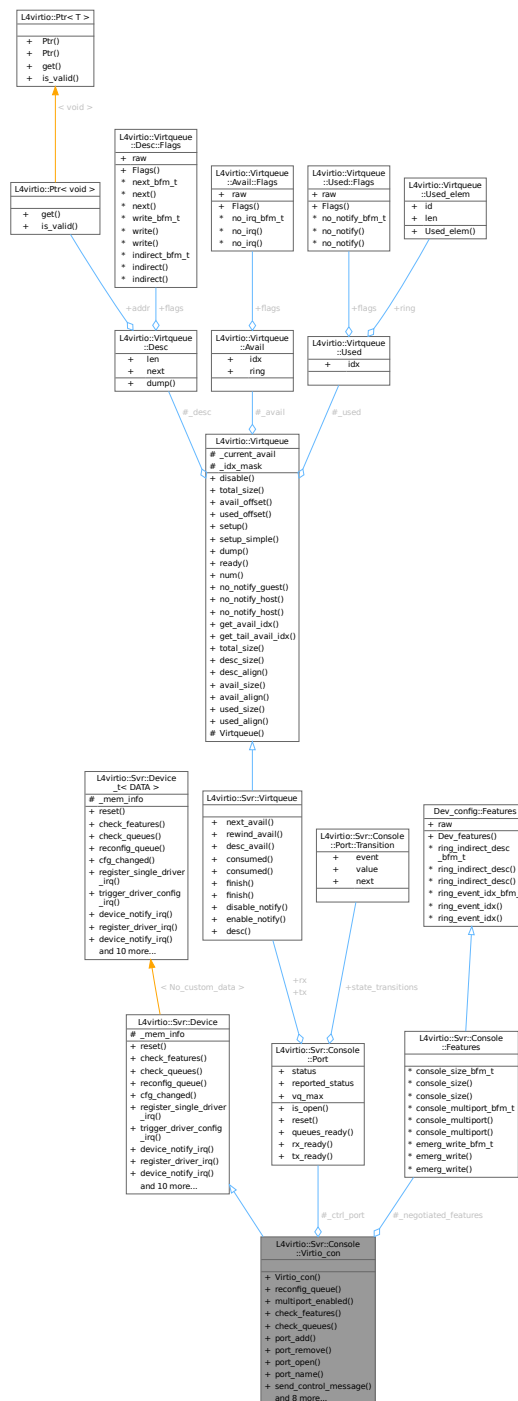
Base class implementing a virtio console functionality.

```
#include <virtio-console>
```


Inheritance diagram for L4virtio::Svr::Console::Virtio_con:



Collaboration diagram for L4virtio::Svr::Console::Virtio_con:



Public Member Functions

- **Virtio_con** (unsigned max_ports, bool enable_multiport)
Create a new multiport console device.
- int **reconfig_queue** (unsigned index) override
callback for client queue-config request
- bool **multiport_enabled** () const

- Return true if the multiport feature is enabled and control queues are available.*

 - bool **check_features** (void) override
callback for checking the subset of accepted features
 - bool **check_queues** () override
callback for checking if the queues at DRIVER_OK transition
 - int **port_add** (unsigned idx)
Send a DEVICE_ADD message and update the internal state.
 - int **port_remove** (unsigned idx)
Send a DEVICE_REMOVE message and update the internal state.
 - int **port_open** (unsigned idx, bool open)
Send a PORT_OPEN message and update the internal state.
 - int **port_name** (unsigned idx, char const *name)
Send a PORT_NAME message to announce the port name.
 - int **send_control_message** (l4_uint32_t idx, l4_uint16_t event, l4_uint16_t value=0, const char *name=0)
Send control message to driver.
 - int **handle_control_message** ()
Handle control message received from the driver.
 - void **reset** () override
reset callback, called for doing a device reset
 - virtual void **reset_device** ()
Reset the state of the actual console device.
 - virtual void **notify_queue** (Virtqueue *queue)=0
Notify queue of available data.
 - virtual Port * **port** (unsigned port)=0
Return the specified port.
 - virtual void **process_device_ready** (l4_uint16_t value)=0
Callback called on DEVICE_READY event.
 - virtual void **process_port_ready** (l4_uint32_t id, l4_uint16_t value)
Callback called on PORT_READY event.
 - virtual void **process_port_open** (l4_uint32_t id, l4_uint16_t value)=0
Callback called on PORT_OPEN event.

Public Member Functions inherited from L4virtio::Svr::Device_t< No_custom_data >

- virtual void **cfg_changed** (unsigned)
callback for client device configuration changes
- virtual void **register_single_driver_irq** ()
callback for registering a single guest IRQ for all queues (old-style)
- virtual void **trigger_driver_config_irq** ()=0
callback for triggering configuration change notification IRQ
- virtual L4::Cap< L4::Irq > **device_notify_irq** () const
callback to gather the device notification IRQ (old-style)
- virtual void **register_driver_irq** (unsigned idx)
Callback for registering an notification IRQ (multi IRQ).
- virtual L4::Cap< L4::Irq > **device_notify_irq** (unsigned idx)
Callback to gather the device notification IRQ (multi IRQ).
- virtual unsigned **num_events_supported** () const
Return the highest notification index supported.
- **Device_t** (Dev_config *dev_config)
Make a device for the given config.

- `Mem_list` const * `mem_info` () const
Get the memory region list used for this device.
- void `reset_queue_config` (unsigned idx, unsigned num_max, bool inc_generation=false)
Trigger reset for the configuration space for queue idx.
- void `init_mem_info` (unsigned num)
Initialize the memory region list to the given maximum.
- void `device_error` ()
Transition device into `DEVICE_NEEDS_RESET` state.
- bool `setup_queue` (`Virtqueue` *q, unsigned qn, unsigned num_max)
Enable/disable the specified queue.
- bool `handle_mem_cmd_write` ()
Check for a value in the `cmd` register and handle a write.
- void `enable_trusted_ds_validation` ()
Enable trusted dataspace validation.
- void `add_trusted_dataspaces` (std::shared_ptr< Ds_vector const > ds)
Provide a list of trusted dataspaces that can be used for validation.

Additional Inherited Members

Protected Attributes inherited from `L4virtio::Svr::Device_t< No_custom_data >`

- `Mem_list` _mem_info
Memory region list.

16.409.1 Detailed Description

Base class implementing a virtio console functionality.

It is possible to activate the MULTIPORT feature, in which case incoming control messages need to be dispatched by calling `handle_control_message()`. The derived class must additionally override `process_device_ready()`, `process_port_ready()` and `process_port_open()` to implement the actual behaviour. The derived class has the following responsibilities:

- inform the driver about usable ports once the device is ready as signaled in `process_device_ready()`, see the wrapper `port_add()`.
- inform the driver about unusable ports, see the wrapper `port_remove()`.
- react to open/close events, see the wrapper `port_open()`.

This implementation provides no means to handle interrupts or notify guests, therefore derived classes have to provide this functionality, see `notify_queue()` and `handle_control_message()`. Similarly, all interaction with data queues has to be implemented. Memory for port structures must be managed by the implementor as well.

Use this class as a base to implement your own specific console device.

Definition at line 267 of file `virtio-console`.

16.409.2 Constructor & Destructor Documentation

16.409.2.1 Virtio_con()

```
L4virtio::Svr::Console::Virtio_con::Virtio_con (
    unsigned max_ports,
    bool enable_multiport) [inline], [explicit]
```

Create a new multiport console device.

Parameters

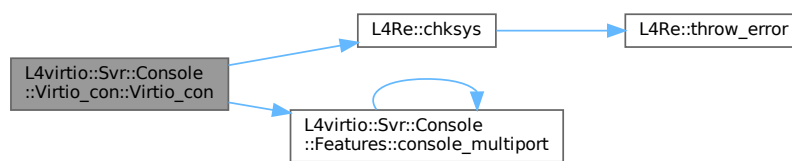
<i>max_ports</i>	Maximum number of ports the device should be able to handle (ignored when <code>enable_multiport</code> is false).
<i>enable_multiport</i>	Enable the control queue for dynamic handling of ports.

Definition at line 293 of file [virtio-console](#).

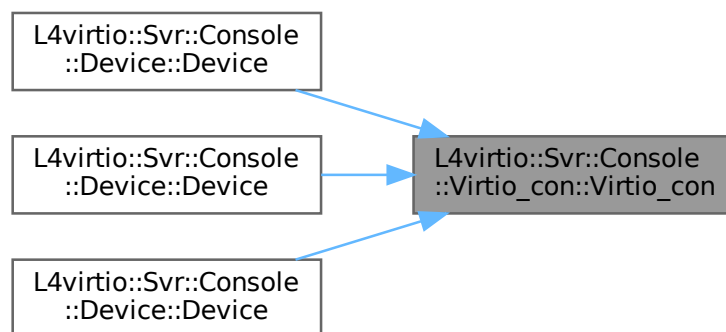
References [L4Re::chksys\(\)](#), [L4virtio::Svr::Console::Features::console_multiport\(\)](#), [L4_EINVAL](#), [L4VIRTIO_ID_CONSOLE](#), and [L4virtio::Svr::Dev_features::raw](#).

Referenced by [L4virtio::Svr::Console::Device::Device\(\)](#), [L4virtio::Svr::Console::Device::Device\(\)](#), and [L4virtio::Svr::Console::Device::Device\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.409.3 Member Function Documentation

16.409.3.1 `handle_control_message()`

```
int L4virtio::Svr::Console::Virtio_con::handle_control_message () [inline]
```

Handle control message received from the driver.

Return values

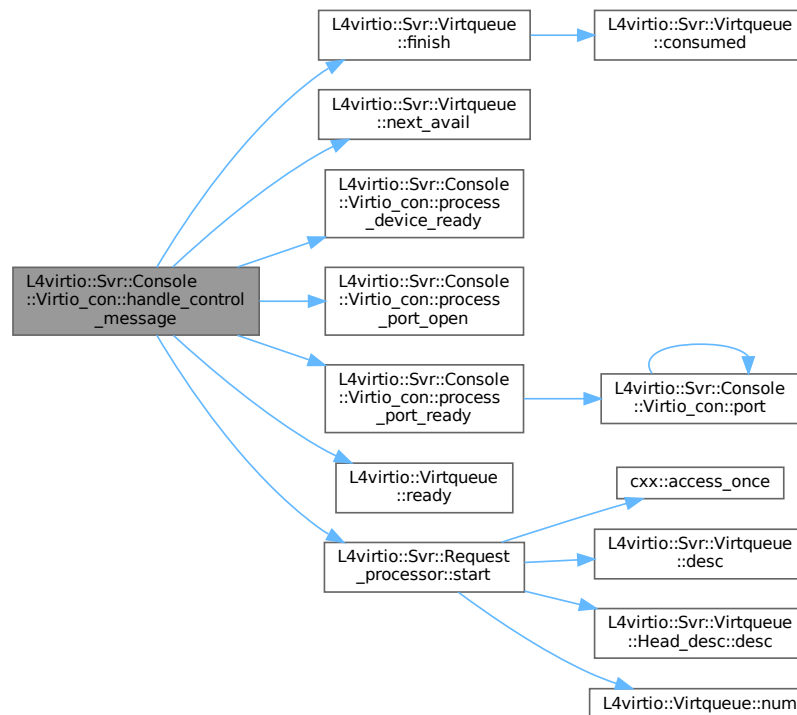
<code>L4_EOK</code>	Message has been handled.
<code>-L4_ENODEV</code>	Control queue is not ready.
<code>-L4_EINVAL</code>	Received an unexpected control event.

This function performs the basic handling of control messages from the driver. It does all necessary work with the control queues and performs some sanity checks. All other work is deferred to the derived class, see `process_device_ready()`, `process_port_ready()` and `process_port_open()`.

Definition at line 536 of file `virtio-console`.

References `L4virtio::Svr::Console::Control_message::Device_ready`, `L4virtio::Svr::Console::Control_message::event`, `L4virtio::Svr::Virtqueue::finish()`, `L4virtio::Svr::Console::Control_message::id`, `L4_EINVAL`, `L4_ENODEV`, `L4_EOK`, `L4virtio::Svr::Console::Control_request::len`, `L4virtio::Svr::Console::Control_request::msg`, `L4virtio::Svr::Virtqueue::next_avail()`, `L4virtio::Svr::Console::Port::Port_disabled`, `L4virtio::Svr::Console::Control_message::Port_open`, `L4virtio::Svr::Console::Port::Port_op`, `L4virtio::Svr::Console::Control_message::Port_ready`, `process_device_ready()`, `process_port_open()`, `process_port_ready()`, `L4virtio::Virtqueue::ready()`, `L4virtio::Svr::Request_processor::start()`, and `L4virtio::Svr::Console::Control_message::value`.

Here is the call graph for this function:



16.409.3.2 notify_queue()

```
virtual void L4virtio::Svr::Console::Virtio_con::notify_queue (
    Virtqueue * queue) [pure virtual]
```

Notify queue of available data.

Parameters

<i>queue</i>	Virtqueue to notify.
--------------	--------------------------------------

This callback is called whenever data is sent to `queue`. It is the responsibility of the derived class to perform all necessary notification actions, e.g. triggering guest interrupts.

Implemented in [L4virtio::Svr::Console::Device](#).

16.409.3.3 port()

```
virtual Port * L4virtio::Svr::Console::Virtio_con::port (
    unsigned port) [pure virtual]
```

Return the specified port.

Parameters

<i>port</i>	Port number.
-------------	------------------------------

Precondition

[Port](#) number must be lower than the configured maximum number of ports.

Implemented in [L4virtio::Svr::Console::Device](#).

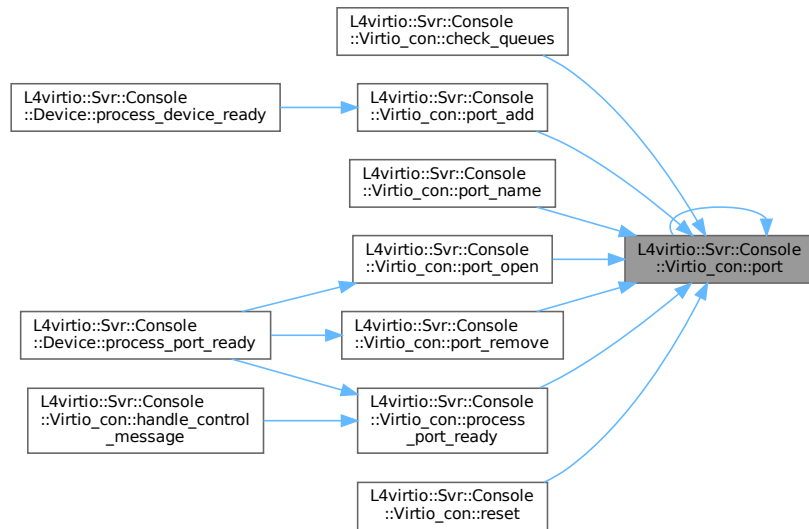
References [port\(\)](#).

Referenced by [check_queues\(\)](#), [port\(\)](#), [port_add\(\)](#), [port_name\(\)](#), [port_open\(\)](#), [port_remove\(\)](#), [process_port_ready\(\)](#), and [reset\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.409.3.4 port_add()

```
int L4virtio::Svr::Console::Virtio_con::port_add (
    unsigned idx) [inline]
```

Send a DEVICE_ADD message and update the internal state.

Parameters

<i>idx</i>	Port that should be added.
------------	----------------------------

Return values

<i>L4_EOK</i>	Message has been sent.
<i>-L4_EPERM</i>	Invalid state transition.

Precondition

idx must be smaller than the configured number of ports.

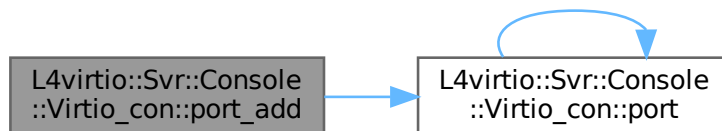
Port must not already exist.

Definition at line 379 of file [virtio-console](#).

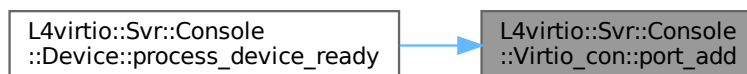
References [L4_EOK](#), [L4_EPERM](#), [port\(\)](#), [L4virtio::Svr::Console::Port::Port_added](#), [L4virtio::Svr::Console::Port::Port_disabled](#), and [L4virtio::Svr::Console::Port::status](#).

Referenced by [L4virtio::Svr::Console::Device::process_device_ready\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.409.3.5 port_name()

```
int L4virtio::Svr::Console::Virtio_con::port_name (
    unsigned idx,
    char const * name) [inline]
```

Send a PORT_NAME message to announce the port name.

Parameters

<i>idx</i>	Port that should be opened or closed.
<i>name</i>	The port name

Return values

<i>L4_EOK</i>	Message has been sent.
<i>-L4_EPERM</i>	Control message is not allowed in the current state.

Returns

Errors from [send_control_message\(\)](#)

Precondition

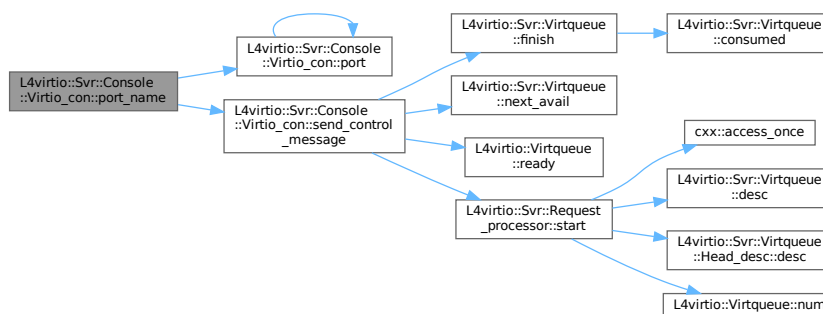
`idx` must be smaller than the configured number of ports.

[Port](#) must already exist.

Definition at line 455 of file [virtio-console](#).

References [L4_EPERM](#), [port\(\)](#), [L4virtio::Svr::Console::Port::Port_disabled](#), [L4virtio::Svr::Console::Control_message::Port_name](#), [send_control_message\(\)](#), and [L4virtio::Svr::Console::Port::status](#).

Here is the call graph for this function:

**16.409.3.6 port_open()**

```
int L4virtio::Svr::Console::Virtio_con::port_open (
    unsigned idx,
    bool open) [inline]
```

Send a PORT_OPEN message and update the internal state.

Parameters

<i>idx</i>	Port that should be opened or closed.
<i>open</i>	Open or close port.

Return values

<i>L4_EOK</i>	Message has been sent.
<i>-L4_EPERM</i>	Invalid state transition.

Precondition

`idx` must be smaller than the configured number of ports.

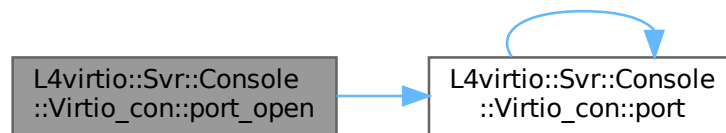
[Port](#) must be ready when opening or open when closing.

Definition at line 428 of file [virtio-console](#).

References [L4_EOK](#), [L4_EPERM](#), [port\(\)](#), [L4virtio::Svr::Console::Port::Port_open](#), [L4virtio::Svr::Console::Port::Port_ready](#), and [L4virtio::Svr::Console::Port::status](#).

Referenced by [L4virtio::Svr::Console::Device::process_port_ready\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**16.409.3.7 port_remove()**

```
int L4virtio::Svr::Console::Virtio_con::port_remove (
    unsigned idx) [inline]
```

Send a `DEVICE_REMOVE` message and update the internal state.

Parameters

<code>idx</code>	Port that should be removed.
------------------	--

Return values

<code>L4_EOK</code>	Message has been sent.
<code>-L4_EPERM</code>	Invalid state transition.

Precondition

`idx` must be smaller than the configured number of ports.

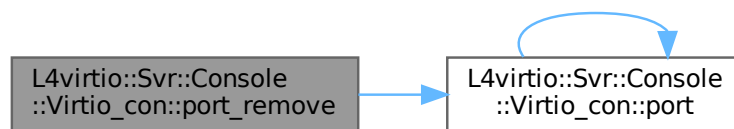
`Port` must already exist.

Definition at line 403 of file `virtio-console`.

References `L4_EOK`, `L4_EPERM`, `port()`, `L4virtio::Svr::Console::Port::Port_disabled`, and `L4virtio::Svr::Console::Port::status`.

Referenced by `L4virtio::Svr::Console::Device::process_port_ready()`.

Here is the call graph for this function:



Here is the caller graph for this function:

**16.409.3.8 process_device_ready()**

```
virtual void L4virtio::Svr::Console::Virtio_con::process_device_ready (
    l4_uint16_t value) [pure virtual]
```

Callback called on `DEVICE_READY` event.

Parameters

<i>value</i>	The value field of the control message, indicating if the initialization was successful.
--------------	--

Needs to be overridden by the derived class if the MULTIPOINT feature is enabled. Control messages may be sent only after the driver has successfully initialized the device.

Implemented in [L4virtio::Svr::Console::Device](#).

Referenced by [handle_control_message\(\)](#).

Here is the caller graph for this function:



16.409.3.9 process_port_open()

```
virtual void L4virtio::Svr::Console::Virtio_con::process_port_open (
    l4_uint32_t id,
    l4_uint16_t value) [pure virtual]
```

Callback called on PORT_OPEN event.

Parameters

<i>id</i>	The id field of the control message, i.e. the port number.
<i>value</i>	The value field of the control message, indicating if the port was opened or closed.

Signal that an application has opened the port. Can to be overridden by the derived class if the MULTIPOINT feature is enabled.

Implemented in [L4virtio::Svr::Console::Device](#).

Referenced by [handle_control_message\(\)](#).

Here is the caller graph for this function:



16.409.3.10 process_port_ready()

```
virtual void L4virtio::Svr::Console::Virtio_con::process_port_ready (
    14_uint32_t id,
    14_uint16_t value) [inline], [virtual]
```

Callback called on PORT_READY event.

Parameters

<i>id</i>	The id field of the control message, i.e. the port number.
<i>value</i>	The value field of the control message, indicating if the initialization was successful.

May be overridden by the derived class if the MULTIPORT feature is enabled. This default implementation just sets the status of the port according to the driver message.

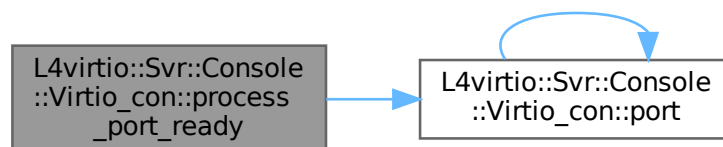
Reimplemented in [L4virtio::Svr::Console::Device](#).

Definition at line 698 of file [virtio-console](#).

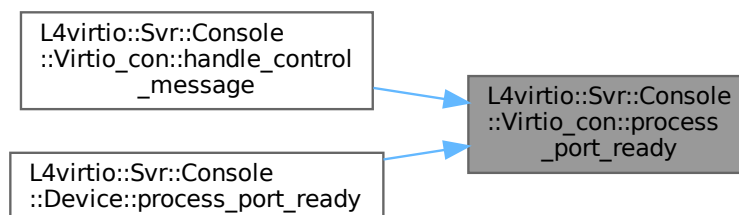
References [port\(\)](#), [L4virtio::Svr::Console::Port::Port_added](#), [L4virtio::Svr::Console::Port::Port_failed](#), [L4virtio::Svr::Console::Port::Port_ready](#), and [L4virtio::Svr::Console::Port::status](#).

Referenced by [handle_control_message\(\)](#), and [L4virtio::Svr::Console::Device::process_port_ready\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.409.3.11 reset_device()

```
virtual void L4virtio::Svr::Console::Virtio_con::reset_device () [inline], [virtual]
```

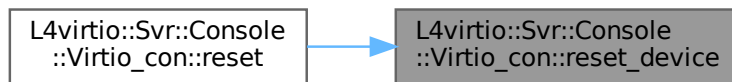
Reset the state of the actual console device.

This callback is called at the end of `reset ()`, allowing the derived class to reset internal state.

Definition at line 652 of file `virtio-console`.

Referenced by `reset()`.

Here is the caller graph for this function:

**16.409.3.12 send_control_message()**

```
int L4virtio::Svr::Console::Virtio_con::send_control_message (
    14_uint32_t idx,
    14_uint16_t event,
    14_uint16_t value = 0,
    const char * name = 0) [inline]
```

Send control message to driver.

Parameters

<i>idx</i>	Port number.
<i>event</i>	Kind of control event.
<i>value</i>	Extra information for the event.
<i>name</i>	Name to be used for Port_name message

Return values

<i>L4_EOK</i>	Message has been sent.
<i>-L4_ENODEV</i>	Control queue is not ready.
<i>-L4_EBUSY</i>	Currently no descriptor available in the control queue.
<i>-L4_ENOMEM</i>	Client-issued descriptor too small. Device will be set to failed state.

Precondition

`port` must be smaller than the configured number of ports.

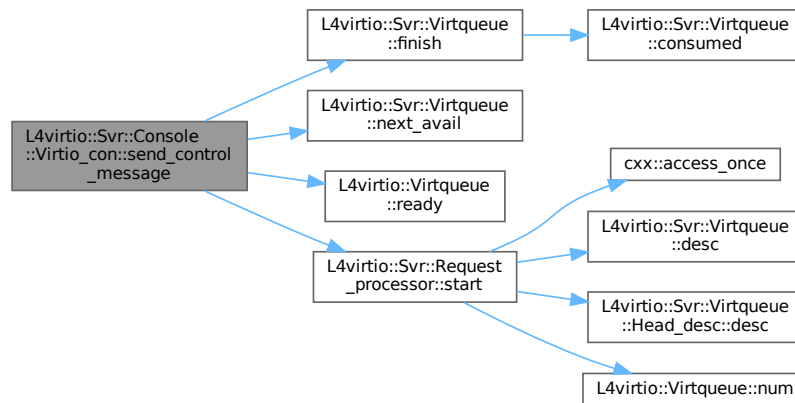
The convenience functions `port_add()`, `port_remove()` and `port_open()` should cover the most use cases and are the preferred way of communication with the driver. If you use this function directly, it is your responsibility to guarantee no invalid control messages are sent to the driver.

Definition at line 487 of file `virtio-console`.

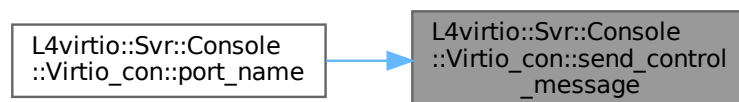
References `L4virtio::Svr::Virtqueue::finish()`, `L4_EBUSY`, `L4_ENODEV`, `L4_ENOMEM`, `L4_EOK`, `L4virtio::Svr::Console::Control_request::msg`, `L4virtio::Svr::Virtqueue::next_avail()`, `L4virtio::Svr::Console::Control_message::Port_name`, `L4virtio::Virtqueue::ready()`, and `L4virtio::Svr::Request_processor::start()`.

Referenced by `port_name()`.

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

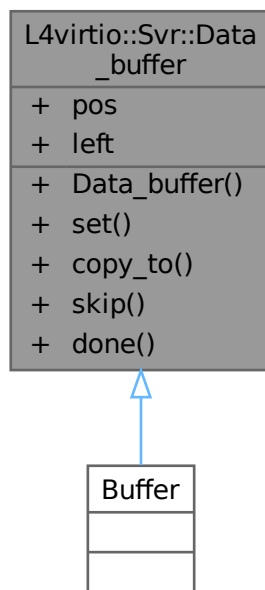
- `l4/l4virtio/server/virtio-console`

16.410 L4virtio::Svr::Data_buffer Struct Reference

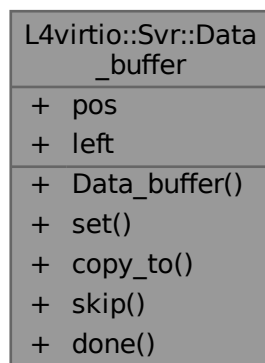
Abstract data buffer.

```
#include <virtio>
```

Inheritance diagram for L4virtio::Svr::Data_buffer:



Collaboration diagram for L4virtio::Svr::Data_buffer:



Public Member Functions

- `template<typename T>`
`Data_buffer` (T *p)
Create buffer for object p.
- `template<typename T>`
`void set` (T *p)
Set buffer for object p.
- `l4_uint32_t copy_to` (`Data_buffer` *dst, `l4_uint32_t` max=UINT_MAX)
Copy contents from this buffer to the destination buffer.
- `l4_uint32_t skip` (`l4_uint32_t` bytes)
Skip given number of bytes in this buffer.
- `bool done` () const
Check if there are no more bytes left in the buffer.

Data Fields

- `char * pos`
Current buffer position.
- `l4_uint32_t left`
Bytes left in buffer.

16.410.1 Detailed Description

Abstract data buffer.

Definition at line 306 of file [virtio](#).

16.410.2 Constructor & Destructor Documentation

16.410.2.1 Data_buffer()

```
template<typename T>
L4virtio::Svr::Data_buffer::Data_buffer (
    T * p) [inline], [explicit]
```

Create buffer for object p.

Template Parameters

<i>T</i>	Type of object (implicit)
----------	---------------------------

Parameters

<i>p</i>	Pointer to object.
----------	--------------------

The buffer shall point to the start of the object p and the size left is sizeof(T).

Definition at line 323 of file [virtio](#).

References [left](#), and [pos](#).

16.410.3 Member Function Documentation

16.410.3.1 copy_to()

```
l4_uint32_t L4virtio::Svr::Data_buffer::copy_to (
    Data_buffer * dst,
    l4_uint32_t max = UINT_MAX) [inline]
```

Copy contents from this buffer to the destination buffer.

Parameters

<i>dst</i>	Destination buffer.
<i>max</i>	(optional) Maximum number of bytes to copy.

Returns

the number of bytes copied.

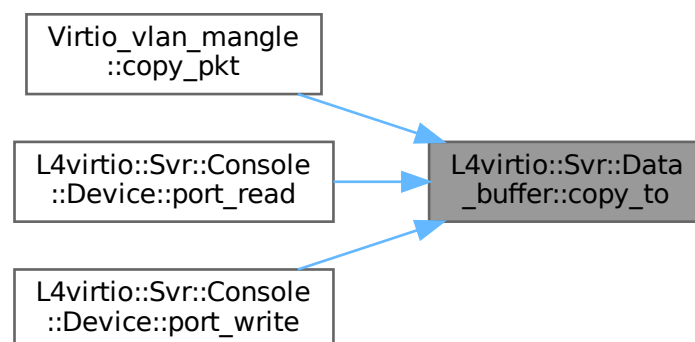
This function copies at most `max` bytes from this to `dst`. If `max` is omitted, copies the maximum number of bytes available that fit `dst`.

Definition at line 354 of file [virtio](#).

References [left](#), and [pos](#).

Referenced by [Virtio_vlan_mangle::copy_pkt\(\)](#), [L4virtio::Svr::Console::Device::port_read\(\)](#), and [L4virtio::Svr::Console::Device::port_w](#)

Here is the caller graph for this function:



16.410.3.2 done()

```
bool L4virtio::Svr::Data_buffer::done () const [inline]
```

Check if there are no more bytes left in the buffer.

Returns

true if there are no more bytes left in the buffer.

Definition at line 388 of file [virtio](#).

References [left](#).

16.410.3.3 set()

```
template<typename T>
void L4virtio::Svr::Data_buffer::set (
    T * p) [inline]
```

Set buffer for object p.

Template Parameters

<i>T</i>	Type of object (implicit)
----------	---------------------------

Parameters

<i>p</i>	Pointer to object.
----------	--------------------

The buffer shall point to the start of the object p and the size left is sizeof(T).

Definition at line 337 of file [virtio](#).

References [left](#), and [pos](#).

16.410.3.4 skip()

```
l4_uint32_t L4virtio::Svr::Data_buffer::skip (
    l4_uint32_t bytes) [inline]
```

Skip given number of bytes in this buffer.

Parameters

<i>bytes</i>	Number of bytes that shall be skipped.
--------------	--

Returns

The number of bytes skipped.

Try to skip the given number of bytes in this buffer, if there are less bytes left in the buffer that given then at most left bytes are skipped and the amount is returned.

Definition at line 375 of file [virtio](#).

References [left](#), and [pos](#).

Referenced by [Virtio_vlan_mangle::copy_pkt\(\)](#).

Here is the caller graph for this function:



The documentation for this struct was generated from the following file:

- l4/l4virtio/server/virtio

16.411 L4virtio::Svr::Dev_config Class Reference

Abstraction for L4-Virtio device config memory.

```
#include <l4virtio>
```

Collaboration diagram for L4virtio::Svr::Dev_config:



Public Member Functions

- [Dev_config](#) ([l4_uint32_t](#) vendor, [l4_uint32_t](#) device, unsigned cfg_size, [l4_uint32_t](#) num_queues=0)
Create a L4-Virtio config data space.
- [Dev_config](#) (Cfg_cap const &cfg, [l4_addr_t](#) cfg_offset, [l4_uint32_t](#) vendor, [l4_uint32_t](#) device, unsigned cfg_size, [l4_uint32_t](#) num_queues=0)
Setup an L4-Virtio config space in an existing data space.
- [l4_uint32_t num_queues](#) () const
Return the number of queues currently usable.
- [l4_uint32_t guest_features](#) (unsigned idx) const
Return a specific set of guest features.
- [l4_uint32_t negotiated_features](#) (unsigned idx) const
Compute a specific set of negotiated features.
- [Status status](#) () const
Get current device status (trusted).
- [l4_uint32_t get_cmd](#) () const
Get the value from the cmd register.
- void [reset_cmd](#) ()
Reset the cmd register after execution of a command.
- void [set_status](#) ([Status status](#))
Set device status register.
- void [add_irq_status](#) ([l4_uint32_t status](#))
Adds irq status bit.
- void [set_device_needs_reset](#) ()
Set DEVICE_NEEDS_RESET bit in device status register.

- bool [change_queue_config](#) ([l4_uint32_t](#) num_queues)
Setup new queue configuration.
- [l4virtio_config_queue_t](#) volatile const * [qconfig](#) (unsigned index) const
Get queue read-only config data for queue with the given index.
- void [reset_hdr](#) (bool inc_generation=false) const
Reset the config header to the initial contents.
- bool [reset_queue](#) (unsigned index, unsigned num_max, bool inc_generation=false) const
Reset queue config for the given queue.
- [l4virtio_config_hdr_t](#) const volatile * [hdr](#) () const
Get a read-only pointer to the config header.
- [L4::Cap](#)< [L4Re::Dataspace](#) > [ds](#) () const
Get data-space capability for the shared config data space.
- [l4_addr_t](#) [ds_offset](#) () const
Return the offset into the config dataspace where the device configuration starts.

16.411.1 Detailed Description

Abstraction for L4-Virtio device config memory.

Virtio defines a device configuration mechanism, L4-Virtio implements this mechanism based on shared memory a [set_status\(\)](#) and a [config_queue\(\)](#) call. This class provides an abstraction for L4-Virtio host implementations to establish such a shared memory data space and providing the necessary contents and access functions.

Definition at line 52 of file [l4virtio](#).

16.411.2 Constructor & Destructor Documentation

16.411.2.1 Dev_config() [1/2]

```
L4virtio::Svr::Dev_config::Dev_config (
    l4\_uint32\_t vendor,
    l4\_uint32\_t device,
    unsigned cfg_size,
    l4\_uint32\_t num_queues = 0) [inline]
```

Create a L4-Virtio config data space.

Parameters

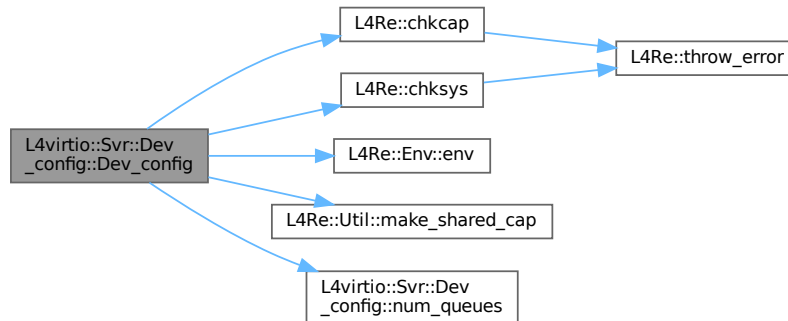
<i>vendor</i>	The vendor ID to store in config header.
<i>device</i>	The device ID to store in config header.
<i>cfg_size</i>	The size of the device-specific config data in bytes.
<i>num_queues</i>	The number of queues provided by the device.

This constructor allocates a data space used for L4-virtio config attaches the data space to the local address space and writes the initial contents to the config header.

Definition at line 112 of file [l4virtio](#).

References [L4Re::chkcap\(\)](#), [L4Re::chksys\(\)](#), [L4Re::Env::env\(\)](#), [L4_PAGESIZE](#), [L4Re::Util::make_shared_cap\(\)](#), and [num_queues\(\)](#).

Here is the call graph for this function:



16.411.2.2 Dev_config() [2/2]

```

L4virtio::Svr::Dev_config::Dev_config (
    Cfg_cap const & cfg,
    14_addr_t cfg_offset,
    14_uint32_t vendor,
    14_uint32_t device,
    unsigned cfg_size,
    14_uint32_t num_queues = 0) [inline]
  
```

Setup an L4-Virtio config space in an existing data space.

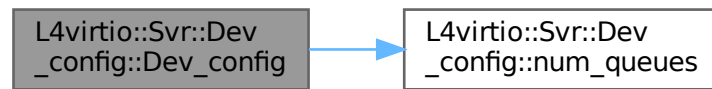
Parameters

<i>cfg</i>	Dataspace that should hold the L4-Virtio configuration.
<i>cfg_offset</i>	Offset into the dataspace where the configuration starts.
<i>vendor</i>	The vendor ID to store in config header.
<i>device</i>	The device ID to store in config header.
<i>cfg_size</i>	The size of the device-specific config data in bytes.
<i>num_queues</i>	The number of queues provided by the device.

Definition at line 146 of file [l4virtio](#).

References [L4_PAGESIZE](#), and [num_queues\(\)](#).

Here is the call graph for this function:



16.411.3 Member Function Documentation

16.411.3.1 add_irq_status()

```
void L4virtio::Svr::Dev_config::add_irq_status (
    14_uint32_t status) [inline]
```

Adds irq status bit.

Parameters

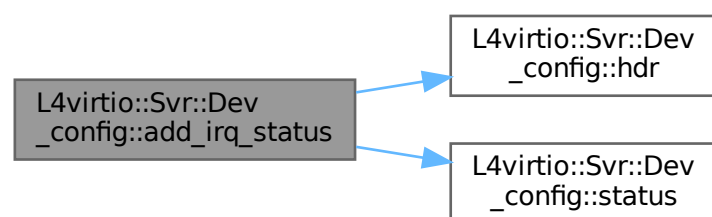
<i>status</i>	The value to add to the irq status register.
---------------	--

This function adds the status bit to the irq status register.

Definition at line 265 of file [l4virtio](#).

References [hdr\(\)](#), and [status\(\)](#).

Here is the call graph for this function:



16.411.3.2 change_queue_config()

```
bool L4virtio::Svr::Dev_config::change_queue_config (
    14_uint32_t num_queues) [inline]
```

Setup new queue configuration.

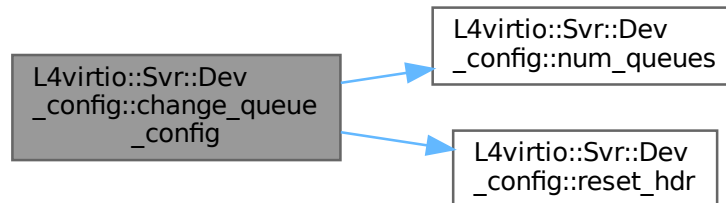
Parameters

<code>num_queues</code>	The number of queues provided by the device.
-------------------------	--

Definition at line 286 of file [l4virtio](#).

References [L4_PAGESIZE](#), [num_queues\(\)](#), and [reset_hdr\(\)](#).

Here is the call graph for this function:



16.411.3.3 ds()

```
L4::Cap< L4Re::Dataspace > L4virtio::Svr::Dev_config::ds () const [inline]
```

Get data-space capability for the shared config data space.

Returns

Capability for the shared config data space.

Definition at line 375 of file [l4virtio](#).

16.411.3.4 get_cmd()

```
l4_uint32_t L4virtio::Svr::Dev_config::get_cmd () const [inline]
```

Get the value from the `cmd` register.

Note, the most significant eight bits are the command (0 is nothing to do). The upper eight bit are reset to zero after the command was handled.

Definition at line 230 of file [l4virtio](#).

References [hdr\(\)](#).

Here is the call graph for this function:



16.411.3.5 guest_features()

```
l4_uint32_t L4virtio::Svr::Dev_config::guest_features (
    unsigned idx) const [inline]
```

Return a specific set of guest features.

Parameters

<i>idx</i>	Index into the guest features array.
------------	--------------------------------------

Return values

<i>The</i>	selected set of guest features.
------------	---------------------------------

This function returns a specific 32bit set of features enabled by the guest/driver. `idx` is the index in the guest features array, resp. the 32 bit set to return.

Definition at line 198 of file [l4virtio](#).

16.411.3.6 hdr()

```
l4virtio_config_hdr_t const volatile * L4virtio::Svr::Dev_config::hdr () const [inline]
```

Get a read-only pointer to the config header.

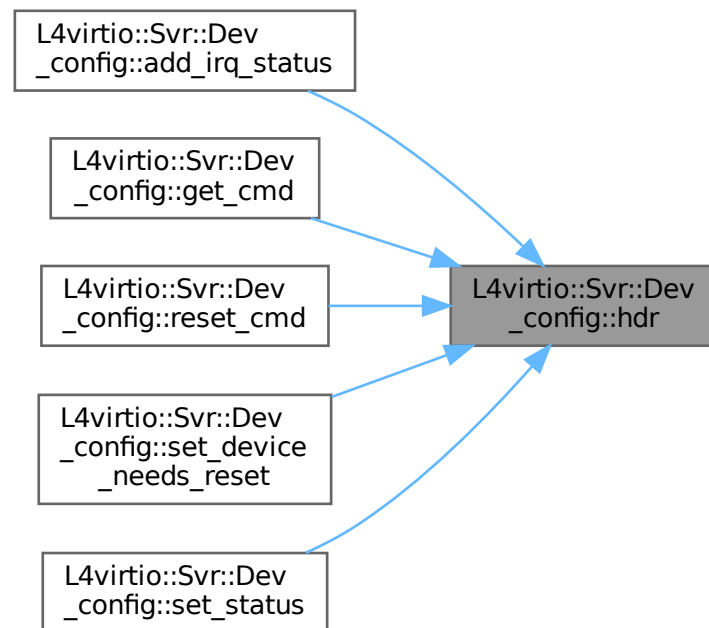
Returns

Read-only pointer to the shared config header.

Definition at line 368 of file [l4virtio](#).

Referenced by [add_irq_status\(\)](#), [get_cmd\(\)](#), [reset_cmd\(\)](#), [set_device_needs_reset\(\)](#), and [set_status\(\)](#).

Here is the caller graph for this function:



16.411.3.7 negotiated_features()

```
l4_uint32_t L4virtio::Svr::Dev_config::negotiated_features (
    unsigned idx) const [inline]
```

Compute a specific set of negotiated features.

Parameters

<i>idx</i>	Index into the guest/host features array.
------------	---

Return values

<i>The</i>	selected set of negotiated features.
------------	--------------------------------------

This function returns a specific 32-bit set of features negotiated by the guest/driver and host/device. *idx* is the index in the guest/host features array, resp. the 32-bit set to return.

Definition at line 212 of file [l4virtio](#).

16.411.3.8 qconfig()

```
l4virtio_config_queue_t volatile const * L4virtio::Svr::Dev_config::qconfig (
    unsigned index) const [inline]
```

Get queue read-only config data for queue with the given *index*.

Parameters

<i>index</i>	The index of the queue.
--------------	-------------------------

Returns

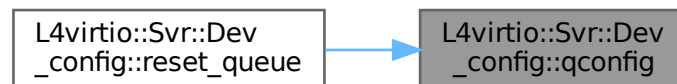
Read-only pointer to the config of the queue with the given *index*, or NULL if *index* is out of range.

Definition at line 303 of file [l4virtio](#).

References [L4_UNLIKELY](#).

Referenced by [reset_queue\(\)](#).

Here is the caller graph for this function:



16.411.3.9 reset_cmd()

```
void L4virtio::Svr::Dev_config::reset_cmd () [inline]
```

Reset the `cmd` register after execution of a command.

This function resets the `cmd` register in order for the client to detect that the command was executed by the device.

Definition at line 241 of file [l4virtio](#).

References [hdr\(\)](#).

Here is the call graph for this function:



16.411.3.10 reset_queue()

```
bool L4virtio::Svr::Dev_config::reset_queue (
    unsigned index,
    unsigned num_max,
    bool inc_generation = false) const [inline]
```

Reset queue config for the given queue.

Parameters

<i>index</i>	The index of the queue to reset.
<i>num_max</i>	The maximum number of descriptors supported by this queue.
<i>inc_generation</i>	The config generation will be incremented when this is true.

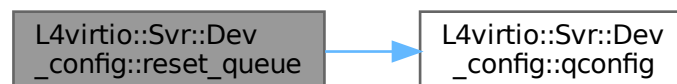
Returns

true on success, or false when *index* is out of range.

Definition at line 345 of file [l4virtio](#).

References [L4_UNLIKELY](#), [l4virtio_config_queue_t::num](#), [l4virtio_config_queue_t::num_max](#), [qconfig\(\)](#), and [l4virtio_config_queue_t::ready](#).

Here is the call graph for this function:



16.411.3.11 set_device_needs_reset()

```
void L4virtio::Svr::Dev_config::set_device_needs_reset () [inline]
```

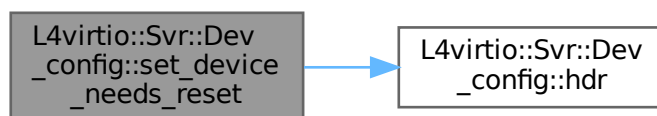
Set `DEVICE_NEEDS_RESET` bit in device status register.

This function sets the internal status register and also the status register in the shared memory to `DEVICE_NEEDS_RESET`.

Definition at line 276 of file [l4virtio](#).

References [hdr\(\)](#).

Here is the call graph for this function:



16.411.3.12 set_status()

```
void L4virtio::Svr::Dev_config::set_status (
    Status status) [inline]
```

Set device status register.

Parameters

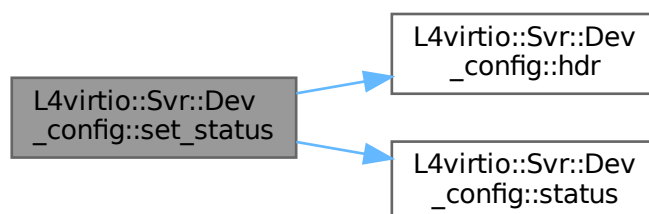
<i>status</i>	The new value for the device status register.
---------------	---

This function sets the internal status register and also the status register in the shared memory to *status*.

Definition at line 253 of file [l4virtio](#).

References [hdr\(\)](#), and [status\(\)](#).

Here is the call graph for this function:



16.411.3.13 status()

```
Status L4virtio::Svr::Dev_config::status () const [inline]
```

Get current device status (trusted).

Returns

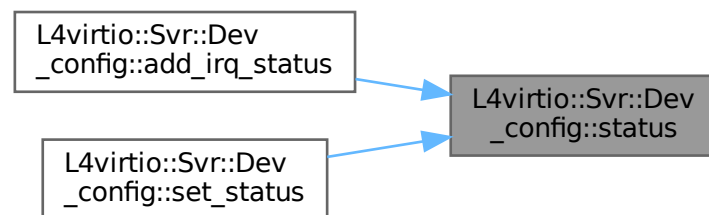
Current device status register (trusted).

The status returned by this function is value stored internally and cannot be written by the guest (i.e., the value can be taken as trusted.)

Definition at line 222 of file [l4virtio](#).

Referenced by [add_irq_status\(\)](#), and [set_status\(\)](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

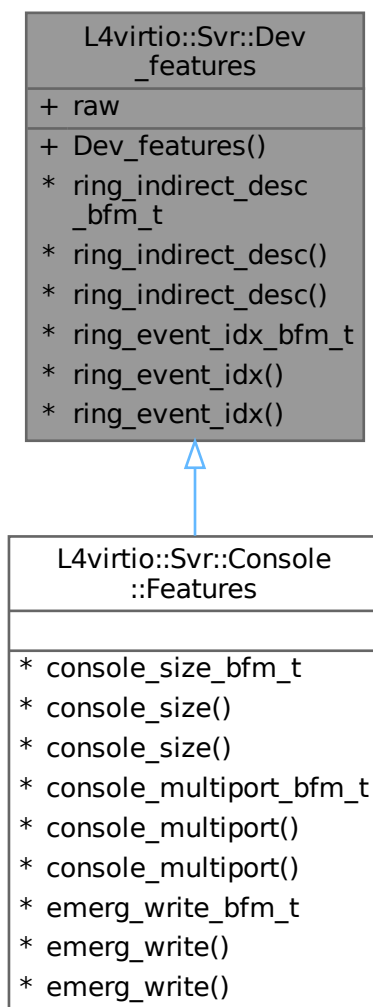
- `l4/l4virtio/server/l4virtio`

16.412 L4virtio::Svr::Dev_features Struct Reference

Type for device feature bitmap.

```
#include <virtio>
```


Inheritance diagram for L4virtio::Svr::Dev_features:



Collaboration diagram for L4virtio::Svr::Dev_features:

L4virtio::Svr::Dev_features
+ raw
+ Dev_features()
* ring_indirect_desc_bfm_t
* ring_indirect_desc()
* ring_indirect_desc()
* ring_event_idx_bfm_t
* ring_event_idx()
* ring_event_idx()

Public Member Functions

- **Dev_features** ([l4_uint32_t](#) v)
Make Features from a raw bitmap.

Data Fields

- [l4_uint32_t](#) raw
The raw value of the features bitmap.
- typedef [cxx::Bitfield](#)< decltype(raw), 28, 28 > **ring_indirect_desc_bfm_t**
Type to access the [ring_indirect_desc](#) bits (28 to 28) of raw.
- constexpr [ring_indirect_desc_bfm_t::Val](#) **ring_indirect_desc** () const
Get the [ring_indirect_desc](#) bits (28 to 28) of raw.
- constexpr [ring_indirect_desc_bfm_t::Ref](#) **ring_indirect_desc** ()
Get a reference to the [ring_indirect_desc](#) bits (28 to 28) of raw.
- typedef [cxx::Bitfield](#)< decltype(raw), 29, 29 > **ring_event_idx_bfm_t**
Type to access the [ring_event_idx](#) bits (29 to 29) of raw.
- constexpr [ring_event_idx_bfm_t::Val](#) **ring_event_idx** () const
Get the [ring_event_idx](#) bits (29 to 29) of raw.
- constexpr [ring_event_idx_bfm_t::Ref](#) **ring_event_idx** ()
Get a reference to the [ring_event_idx](#) bits (29 to 29) of raw.

16.412.1 Detailed Description

Type for device feature bitmap.

Definition at line 66 of file [virtio](#).

The documentation for this struct was generated from the following file:

- l4/l4virtio/server/virtio

16.413 L4virtio::Svr::Dev_status Struct Reference

Type of the device status register.

```
#include <virtio>
```

Collaboration diagram for L4virtio::Svr::Dev_status:

L4virtio::Svr::Dev_status
+ raw
+ Dev_status()
+ running()
* acknowledged_bfm_t
* acknowledged()
* acknowledged()
* driver_bfm_t
* driver()
* driver()
* driver_ok_bfm_t
* driver_ok()
* driver_ok()
* features_ok_bfm_t
* features_ok()
* features_ok()
* fail_state_bfm_t
* fail_state()
* fail_state()
* device_needs_reset_bfm_t
* device_needs_reset()
* device_needs_reset()
* failed_bfm_t
* failed()
* failed()

Public Member Functions

- **Dev_status** ([l4_uint32_t](#) v)
Make Status from raw value.
- bool **running** () const
Check if the device is in running state.

Data Fields

- unsigned char **raw**
Raw value of the VIRTIO device status register.
- typedef [cxx::Bitfield](#)< decltype([raw](#)), 0, 0 > **acked_bfm_t**
Type to access the [acked](#) bits (0 to 0) of [raw](#).
- constexpr [acked_bfm_t::Val](#) **acked** () const
Get the [acked](#) bits (0 to 0) of [raw](#).
- constexpr [acked_bfm_t::Ref](#) **acked** ()
Get a reference to the [acked](#) bits (0 to 0) of [raw](#).
- typedef [cxx::Bitfield](#)< decltype([raw](#)), 1, 1 > **driver_bfm_t**
Type to access the [driver](#) bits (1 to 1) of [raw](#).
- constexpr [driver_bfm_t::Val](#) **driver** () const
Get the [driver](#) bits (1 to 1) of [raw](#).
- constexpr [driver_bfm_t::Ref](#) **driver** ()
Get a reference to the [driver](#) bits (1 to 1) of [raw](#).
- typedef [cxx::Bitfield](#)< decltype([raw](#)), 2, 2 > **driver_ok_bfm_t**
Type to access the [driver_ok](#) bits (2 to 2) of [raw](#).
- constexpr [driver_ok_bfm_t::Val](#) **driver_ok** () const
Get the [driver_ok](#) bits (2 to 2) of [raw](#).
- constexpr [driver_ok_bfm_t::Ref](#) **driver_ok** ()
Get a reference to the [driver_ok](#) bits (2 to 2) of [raw](#).
- typedef [cxx::Bitfield](#)< decltype([raw](#)), 3, 3 > **features_ok_bfm_t**
Type to access the [features_ok](#) bits (3 to 3) of [raw](#).
- constexpr [features_ok_bfm_t::Val](#) **features_ok** () const
Get the [features_ok](#) bits (3 to 3) of [raw](#).
- constexpr [features_ok_bfm_t::Ref](#) **features_ok** ()
Get a reference to the [features_ok](#) bits (3 to 3) of [raw](#).
- typedef [cxx::Bitfield](#)< decltype([raw](#)), 6, 7 > **fail_state_bfm_t**
Type to access the [fail_state](#) bits (6 to 7) of [raw](#).
- constexpr [fail_state_bfm_t::Val](#) **fail_state** () const
Get the [fail_state](#) bits (6 to 7) of [raw](#).
- constexpr [fail_state_bfm_t::Ref](#) **fail_state** ()
Get a reference to the [fail_state](#) bits (6 to 7) of [raw](#).
- typedef [cxx::Bitfield](#)< decltype([raw](#)), 6, 6 > **device_needs_reset_bfm_t**

Type to access the *device_needs_reset* bits (6 to 6) of *raw*.

- constexpr [device_needs_reset_bfm_t::Val](#) **device_needs_reset** () const

Get the *device_needs_reset* bits (6 to 6) of *raw*.

- constexpr [device_needs_reset_bfm_t::Ref](#) **device_needs_reset** ()

Get a reference to the *device_needs_reset* bits (6 to 6) of *raw*.

- typedef [cxx::Bitfield](#)< decltype(*raw*), 7, 7 > **failed_bfm_t**

Type to access the *failed* bits (7 to 7) of *raw*.

- constexpr [failed_bfm_t::Val](#) **failed** () const

Get the *failed* bits (7 to 7) of *raw*.

- constexpr [failed_bfm_t::Ref](#) **failed** ()

Get a reference to the *failed* bits (7 to 7) of *raw*.

16.413.1 Detailed Description

Type of the device status register.

Definition at line 32 of file [virtio](#).

16.413.2 Member Function Documentation

16.413.2.1 running()

```
bool L4virtio::Svr::Dev_status::running () const [inline]
```

Check if the device is in running state.

Returns

true if the device is in running state.

The device is in running state when [acked\(\)](#), [driver\(\)](#), [features_ok\(\)](#), and [driver_ok\(\)](#) return true, and [device_needs_reset\(\)](#) and [failed\(\)](#) return false.

Definition at line 57 of file [virtio](#).

References [raw](#).

The documentation for this struct was generated from the following file:

- I4/I4virtio/server/virtio

Collaboration diagram for L4virtio::Svr::Device_t< DATA >:

L4virtio::Svr::Device _t< DATA >
_mem_info
+ reset() + check_features() + check_queues() + reconfig_queue() + cfg_changed() + register_single_driver_irq() + trigger_driver_config_irq() + device_notify_irq() + register_driver_irq() + device_notify_irq() and 10 more...

Public Member Functions

- virtual void **reset** ()=0
reset callback, called for doing a device reset
- virtual bool **check_features** ()
callback for checking the subset of accepted features
- virtual bool **check_queues** ()=0
callback for checking if the queues at DRIVER_OK transition
- virtual int **reconfig_queue** (unsigned idx)=0
callback for client queue-config request
- virtual void **cfg_changed** (unsigned)
callback for client device configuration changes
- virtual void **register_single_driver_irq** ()
callback for registering a single guest IRQ for all queues (old-style)
- virtual void **trigger_driver_config_irq** ()=0
callback for triggering configuration change notification IRQ
- virtual [L4::Cap](#)< [L4::Irq](#) > **device_notify_irq** () const
callback to gather the device notification IRQ (old-style)
- virtual void [register_driver_irq](#) (unsigned idx)
Callback for registering an notification IRQ (multi IRQ).
- virtual [L4::Cap](#)< [L4::Irq](#) > **device_notify_irq** (unsigned idx)
Callback to gather the device notification IRQ (multi IRQ).
- virtual unsigned **num_events_supported** () const
Return the highest notification index supported.

- **Device_t** ([Dev_config](#) *dev_config)
Make a device for the given config.
- **Mem_list** const * **mem_info** () const
Get the memory region list used for this device.
- void **reset_queue_config** (unsigned idx, unsigned num_max, bool inc_generation=false)
Trigger reset for the configuration space for queue idx.
- void **init_mem_info** (unsigned num)
Initialize the memory region list to the given maximum.
- void **device_error** ()
Transition device into DEVICE_NEEDS_RESET state.
- bool **setup_queue** ([Virtqueue](#) *q, unsigned qn, unsigned num_max)
Enable/disable the specified queue.
- bool **handle_mem_cmd_write** ()
Check for a value in the cmd register and handle a write.
- void **enable_trusted_ds_validation** ()
Enable trusted dataspace validation.
- void **add_trusted_dataspaces** (std::shared_ptr< Ds_vector const > ds)
Provide a list of trusted dataspaces that can be used for validation.

Protected Attributes

- **Mem_list** **mem_info**
Memory region list.

16.414.1 Detailed Description

```
template<typename DATA>
class L4virtio::Svr::Device_t< DATA >
```

Server-side L4-VIRTIO device stub.

This stub supports new-style multi-event registration (using `get_device_config()`, `bind()` and `get_device_notification_irq()`).

Definition at line 801 of file [l4virtio](#).

16.414.2 Member Function Documentation

16.414.2.1 add_trusted_dataspaces()

```
template<typename DATA>
void L4virtio::Svr::Device_t< DATA >::add_trusted_dataspaces (
    std::shared_ptr< Ds_vector const > ds) [inline]
```

Provide a list of trusted dataspaces that can be used for validation.

Parameters

<i>ds</i>	list of trusted dataspace.
-----------	----------------------------

Definition at line 1198 of file [l4virtio](#).

16.414.2.2 device_error()

```
template<typename DATA>
void L4virtio::Svr::Device_t< DATA >::device_error () [inline]
```

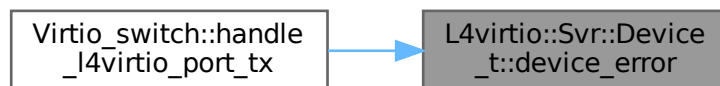
Transition device into DEVICE_NEEDS_RESET state.

This function does a full reset, sets the DEVICE_NEEDS_RESET bit in the device status register, triggering a guest config IRQ if necessary. The driver still needs to perform its own reset and initialization sequence.

Definition at line 1024 of file [l4virtio](#).

Referenced by [Virtio_switch::handle_l4virtio_port_tx\(\)](#).

Here is the caller graph for this function:



16.414.2.3 device_notify_irq()

```
template<typename DATA>
virtual L4::Cap< L4::Irq > L4virtio::Svr::Device_t< DATA >::device_notify_irq (
    unsigned idx) [inline], [virtual]
```

Callback to gather the device notification IRQ (multi IRQ).

The default implementation maps to the implementation for single IRQ notification points.

Definition at line 874 of file [l4virtio](#).

16.414.2.4 handle_mem_cmd_write()

```
template<typename DATA>
bool L4virtio::Svr::Device_t< DATA >::handle_mem_cmd_write () [inline]
```

Check for a value in the `cmd` register and handle a write.

This function checks for a value in the `cmd` register and executes the command if there is any, or returns false if there was no command.

Execution of the command is signaled by a zero in the `cmd` register.

Definition at line 1154 of file [l4virtio](#).

16.414.2.5 `init_mem_info()`

```
template<typename DATA>
void L4virtio::Svr::Device_t< DATA >::init_mem_info (
    unsigned num) [inline]
```

Initialize the memory region list to the given maximum.

Parameters

<i>num</i>	Maximum number of memory regions that can be managed.
------------	---

Definition at line 1012 of file [l4virtio](#).

16.414.2.6 `register_driver_irq()`

```
template<typename DATA>
virtual void L4virtio::Svr::Device_t< DATA >::register_driver_irq (
    unsigned idx) [inline], [virtual]
```

Callback for registering an notification IRQ (multi IRQ).

The default implementation maps to the implementation for single IRQ notification points.

Definition at line 860 of file [l4virtio](#).

16.414.2.7 `reset_queue_config()`

```
template<typename DATA>
void L4virtio::Svr::Device_t< DATA >::reset_queue_config (
    unsigned idx,
    unsigned num_max,
    bool inc_generation = false) [inline]
```

Trigger reset for the configuration space for queue *idx*.

Parameters

<i>idx</i>	The queue index to reset.
<i>num_max</i>	Maximum number of entries in this queue.
<i>inc_generation</i>	The config generation will be incremented when this is true.

This function resets the driver-readable configuration space for the queue with the given index. The queue configuration is reset to all 0, and the maximum number of entries in the queue is set to *num_max*.

Definition at line 1002 of file [l4virtio](#).

16.414.2.8 setup_queue()

```
template<typename DATA>
bool L4virtio::Svr::Device_t< DATA >::setup_queue (
    Virtqueue * q,
    unsigned qn,
    unsigned num_max) [inline]
```

Enable/disable the specified queue.

Parameters

<i>q</i>	Pointer to the ring that represents the virtqueue internally.
<i>qn</i>	Index of the queue.
<i>num_max</i>	Maximum number of supported entries in this queue.

Returns

true for success.

- This function calculates the parameters of the virtqueue from the clients configuration space values, checks the accessibility of the queue data structures and initializes *q* to ready state when all checks succeeded.

Definition at line 1047 of file [l4virtio](#).

The documentation for this class was generated from the following file:

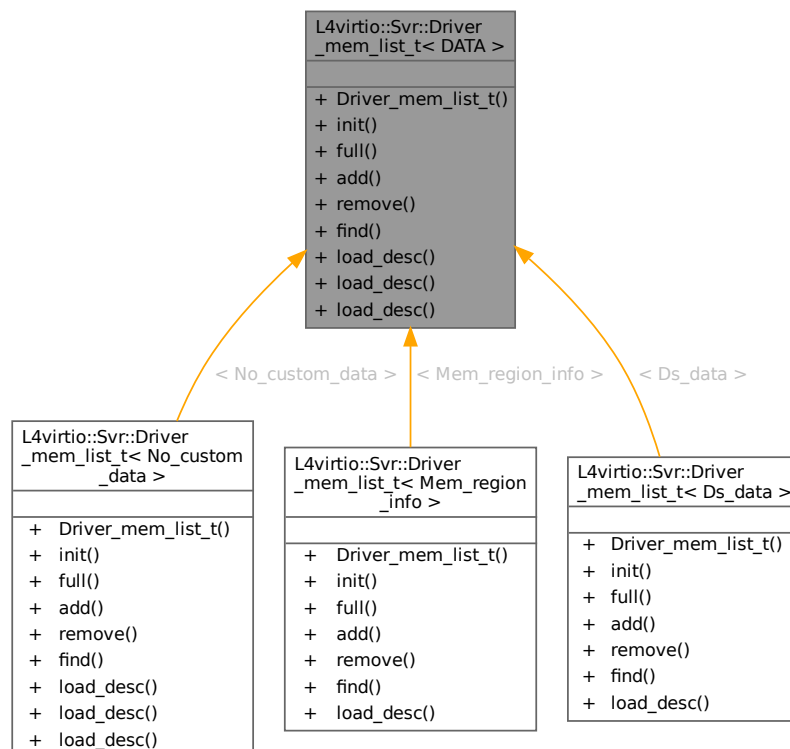
- [l4/l4virtio/server/l4virtio](#)

16.415 L4virtio::Svr::Driver_mem_list_t< DATA > Class Template Reference

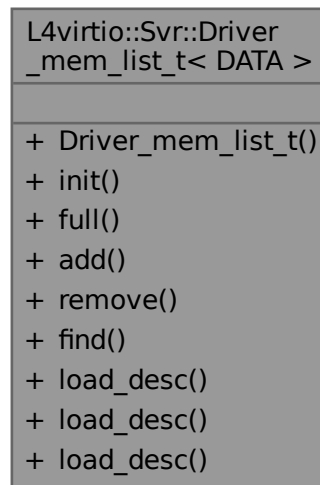
List of driver memory regions assigned to a single L4-VIRTIO transport instance.

```
#include <l4virtio>
```

Inheritance diagram for L4virtio::Svr::Driver_mem_list_t< DATA >:



Collaboration diagram for L4virtio::Svr::Driver_mem_list_t< DATA >:



Public Types

- typedef [L4Re::Util::Unique_cap](#)< [L4Re::Dataspace](#) > **Ds_cap**
type for storing a data-space capability internally

Public Member Functions

- **Driver_mem_list_t ()**
Make an empty, zero capacity list.
- void **init** (unsigned max)
Make a fresh list with capacity max.
- bool **full** () const
- [Mem_region](#) const * **add** ([l4_uint64_t](#) drv_base, [l4_umword_t](#) size, [l4_addr_t](#) offset, [Ds_cap](#) &&ds)
Add a new region to the list.
- void **remove** ([Mem_region](#) const *r)
Remove the given region from the list.
- [Mem_region](#) * **find** ([l4_uint64_t](#) base, [l4_umword_t](#) size) const
Find memory region containing the given driver address region.
- void **load_desc** ([Virtqueue::Desc](#) const &desc, [Request_processor](#) const *p, [Virtqueue::Desc](#) const **table) const
Default implementation for loading an indirect descriptor.
- void **load_desc** ([Virtqueue::Desc](#) const &desc, [Request_processor](#) const *p, [Mem_region](#) const **data) const
Default implementation returning the Driver_mem_region.
- template<typename ARG>
void **load_desc** ([Virtqueue::Desc](#) const &desc, [Request_processor](#) const *p, ARG *data) const
Default implementation returning generic information.

16.415.1 Detailed Description

```
template<typename DATA>
class L4virtio::Svr::Driver_mem_list_t< DATA >
```

List of driver memory regions assigned to a single L4-VIRTIO transport instance.

Note

The regions added to this list *must* never overlap.

Definition at line 629 of file [l4virtio](#).

16.415.2 Member Function Documentation

16.415.2.1 add()

```
template<typename DATA>
Mem_region const * L4virtio::Svr::Driver_mem_list_t< DATA >::add (
    l4_uint64_t drv_base,
    l4_umword_t size,
    l4_addr_t offset,
    Ds_cap && ds) [inline]
```

Add a new region to the list.

Parameters

<i>drv_base</i>	Driver base address of the region.
<i>size</i>	Size of the region in bytes.
<i>offset</i>	Offset within the data space attached to <i>drv_base</i> .
<i>ds</i>	Data space backing the driver memory.

Returns

A pointer to the new region.

Definition at line 669 of file [l4virtio](#).

16.415.2.2 find()

```
template<typename DATA>
Mem_region * L4virtio::Svr::Driver_mem_list_t< DATA >::find (
    l4_uint64_t base,
    l4_umword_t size) const [inline]
```

Find memory region containing the given driver address region.

Parameters

<i>base</i>	Driver base address.
<i>size</i>	Size of the region.

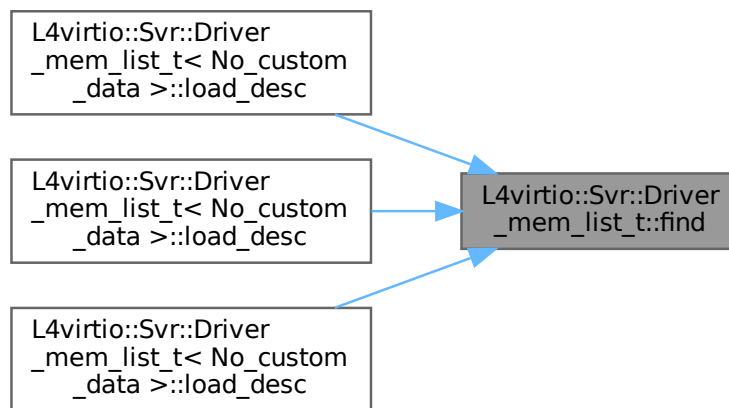
Returns

Pointer to the region containing the given region, NULL if none is found.

Definition at line 703 of file [l4virtio](#).

Referenced by [L4virtio::Svr::Driver_mem_list_t< No_custom_data >::load_desc\(\)](#), [L4virtio::Svr::Driver_mem_list_t< No_custom_data >::load_desc\(\)](#) and [L4virtio::Svr::Driver_mem_list_t< No_custom_data >::load_desc\(\)](#).

Here is the caller graph for this function:

**16.415.2.3 full()**

```

template<typename DATA>
bool L4virtio::Svr::Driver_mem_list_t< DATA >::full () const [inline]

```

Returns

True if the remaining capacity is 0.

Definition at line 658 of file [l4virtio](#).

Referenced by [L4virtio::Svr::Driver_mem_list_t< No_custom_data >::add\(\)](#).

Here is the caller graph for this function:



16.415.2.4 init()

```
template<typename DATA>
void L4virtio::Svr::Driver_mem_list_t< DATA >::init (
    unsigned max) [inline]
```

Make a fresh list with capacity *max*.

Parameters

<i>max</i>	The capacity of this vector.
------------	------------------------------

Definition at line 650 of file [l4virtio](#).

16.415.2.5 load_desc() [1/3]

```
template<typename DATA>
template<typename ARG>
void L4virtio::Svr::Driver_mem_list_t< DATA >::load_desc (
    Virtqueue::Desc const & desc,
    Request_processor const * p,
    ARG * data) const [inline]
```

Default implementation returning generic information.

Template Parameters

<i>ARG</i>	Abstract argument type used with Request_processor::start() and Request_processor::next() to deliver the result of loading a descriptor. This type must provide a constructor taking three arguments: (1) pointer to a <code>Driver_mem_region</code> , (2) the Virtqueue::Desc descriptor, and (3) a pointer to the calling Request_processor .
------------	--

Parameters

	<i>desc</i>	The descriptor to load
--	-------------	------------------------

	<i>p</i>	The request processor calling us
out	<i>data</i>	Shall be assigned to ARG(mem, desc, p)

Exceptions

<i>Bad_descriptor</i>	The descriptor address could not be translated.
---------------------------------------	---

Definition at line 764 of file [l4virtio](#).

16.415.2.6 load_desc() [2/3]

```
template<typename DATA>
void L4virtio::Svr::Driver_mem_list_t< DATA >::load_desc (
    Virtqueue::Desc const & desc,
    Request_processor const * p,
    Mem_region const ** data) const [inline]
```

Default implementation returning the Driver_mem_region.

Parameters

	<i>desc</i>	The descriptor to load
	<i>p</i>	The request processor calling us
out	<i>data</i>	Shall be set to a pointer to the Driver_mem_region that covers the descriptor.

Exceptions

<i>Bad_descriptor</i>	The descriptor address could not be translated.
---------------------------------------	---

Definition at line 737 of file [l4virtio](#).

16.415.2.7 load_desc() [3/3]

```
template<typename DATA>
void L4virtio::Svr::Driver_mem_list_t< DATA >::load_desc (
    Virtqueue::Desc const & desc,
    Request_processor const * p,
    Virtqueue::Desc const ** table) const [inline]
```

Default implementation for loading an indirect descriptor.

Parameters

	<i>desc</i>	The descriptor to load
--	-------------	------------------------

	<i>p</i>	The request processor calling us
out	<i>table</i>	Shall be set to the loaded descriptor table

Exceptions

Bad_descriptor	The descriptor address could not be translated.
--------------------------------	---

Definition at line 717 of file [l4virtio](#).

16.415.2.8 remove()

```
template<typename DATA>
void L4virtio::Svr::Driver_mem_list_t< DATA >::remove (
    Mem_region const * r) [inline]
```

Remove the given region from the list.

Parameters

<i>r</i>	The region to remove (result from add() , or find()).
----------	--

Definition at line 683 of file [l4virtio](#).

The documentation for this class was generated from the following file:

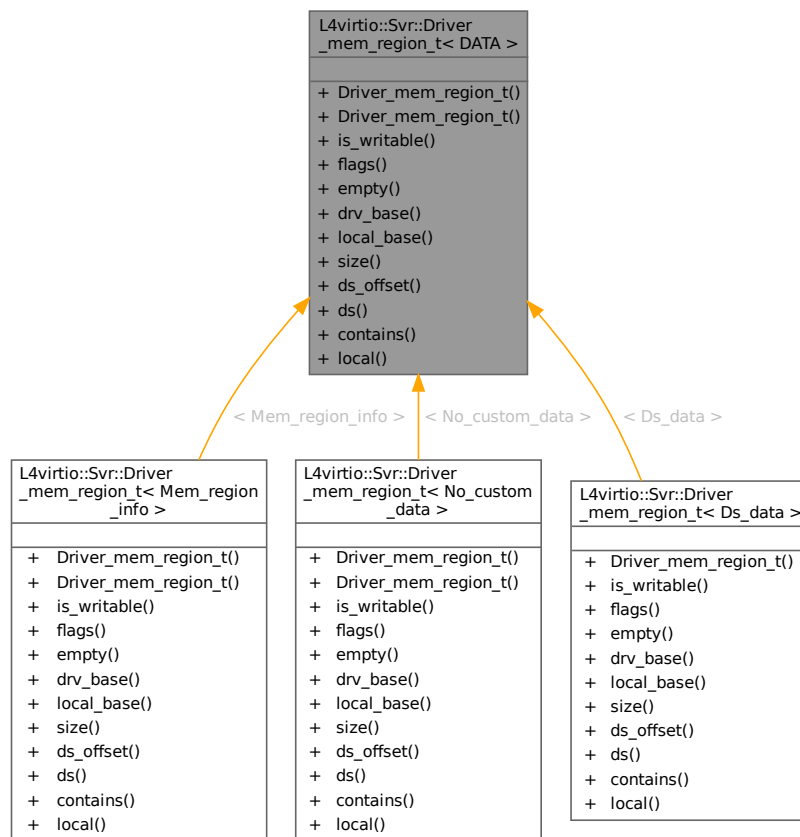
- [l4/l4virtio/server/l4virtio](#)

16.416 L4virtio::Svr::Driver_mem_region_t< DATA > Class Template Reference

Region of driver memory, that shall be managed locally.

```
#include <l4virtio>
```

Inheritance diagram for L4virtio::Svr::Driver_mem_region_t< DATA >:



Collaboration diagram for L4virtio::Svr::Driver_mem_region_t< DATA >:

L4virtio::Svr::Driver_mem_region_t< DATA >
<ul style="list-style-type: none"> + Driver_mem_region_t() + Driver_mem_region_t() + is_writable() + flags() + empty() + drv_base() + local_base() + size() + ds_offset() + ds() + contains() + local()

Public Member Functions

- **Driver_mem_region_t ()**
Make default empty memory region.
- **Driver_mem_region_t (l4_uint64_t drv_base, l4_umword_t size, l4_addr_t offset, Ds_cap &&ds)**
Make a local memory region for the given driver values.
- bool **is_writable ()** const
- Flags **flags ()** const
- bool **empty ()** const
- **l4_uint64_t drv_base ()** const
- **l4_addr_t local_base ()** const
- **l4_umword_t size ()** const
- **l4_addr_t ds_offset ()** const
- **L4::Cap< L4Re::Dataspace > ds ()** const
- bool **contains (l4_uint64_t base, l4_umword_t size)** const
Test if the given driver address range is within this region.
- template<typename T>
T * local (Ptr< T > p) const
Get the local address for driver address p.

16.416.1 Detailed Description

```
template<typename DATA>
class L4virtio::Svr::Driver_mem_region_t< DATA >
```

Region of driver memory, that shall be managed locally.

Template Parameters

<i>DATA</i>	Class defining additional information
-------------	---------------------------------------

Definition at line 450 of file [l4virtio](#).

16.416.2 Constructor & Destructor Documentation

16.416.2.1 Driver_mem_region_t()

```
template<typename DATA>
L4virtio::Svr::Driver_mem_region_t< DATA >::Driver_mem_region_t (
    l4_uint64_t drv_base,
    l4_umword_t size,
    l4_addr_t offset,
    Ds_cap && ds) [inline]
```

Make a local memory region for the given driver values.

Parameters

<i>drv_base</i>	Base address of the memory region used by the driver.
<i>size</i>	Size of the memory region.
<i>offset</i>	Offset within the data space that is mapped to <i>drv_base</i> within the driver.
<i>ds</i>	Data space capability backing the memory.

This constructor attaches the region of given data space to the local address space and stores the corresponding data for later reference.

Definition at line 498 of file [l4virtio](#).

16.416.3 Member Function Documentation

16.416.3.1 contains()

```
template<typename DATA>
bool L4virtio::Svr::Driver_mem_region_t< DATA >::contains (
    l4_uint64_t base,
    l4_umword_t size) const [inline]
```

Test if the given driver address range is within this region.

Parameters

<i>base</i>	The driver base address.
<i>size</i>	The size of the region to lookup.

Returns

true if the given driver address region is contained in this region, false else.

Definition at line 592 of file [l4virtio](#).

16.416.3.2 drv_base()

```
template<typename DATA>
l4_uint64_t L4virtio::Svr::Driver_mem_region_t< DATA >::drv_base () const [inline]
```

Returns

The base address used by the driver.

Definition at line 571 of file [l4virtio](#).

16.416.3.3 ds()

```
template<typename DATA>
L4::Cap< L4Re::Dataspace > L4virtio::Svr::Driver_mem_region_t< DATA >::ds () const [inline]
```

Returns

The data space capability for this region.

Definition at line 583 of file [l4virtio](#).

16.416.3.4 ds_offset()

```
template<typename DATA>
l4_addr_t L4virtio::Svr::Driver_mem_region_t< DATA >::ds_offset () const [inline]
```

Returns

The offset within the data space.

Definition at line 580 of file [l4virtio](#).

16.416.3.5 empty()

```
template<typename DATA>
bool L4virtio::Svr::Driver_mem_region_t< DATA >::empty () const [inline]
```

Returns

True if the region is empty (size == 0), false otherwise.

Definition at line 567 of file [l4virtio](#).

16.416.3.6 flags()

```
template<typename DATA>
Flags L4virtio::Svr::Driver_mem_region_t< DATA >::flags () const [inline]
```

Returns

The flags for this region.

Definition at line 564 of file [l4virtio](#).

16.416.3.7 is_writable()

```
template<typename DATA>
bool L4virtio::Svr::Driver_mem_region_t< DATA >::is_writable () const [inline]
```

Returns

True if the region is writable, false otherwise.

Definition at line 561 of file [l4virtio](#).

16.416.3.8 local()

```
template<typename DATA>
template<typename T>
T * L4virtio::Svr::Driver_mem_region_t< DATA >::local (
    Ptr< T > p) const [inline]
```

Get the local address for driver address *p*.

Parameters

<i>p</i>	Driver address to translate.
----------	------------------------------

Precondition

p must be contained in this region.

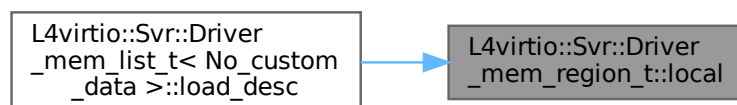
Returns

Local address for the given driver address *p*.

Definition at line 616 of file [l4virtio](#).

Referenced by [L4virtio::Svr::Driver_mem_list_t< No_custom_data >::load_desc\(\)](#).

Here is the caller graph for this function:



16.416.3.9 local_base()

```
template<typename DATA>
l4_addr_t L4virtio::Svr::Driver_mem_region_t< DATA >::local_base () const [inline]
```

Returns

The local base address.

Definition at line 574 of file [l4virtio](#).

16.416.3.10 size()

```
template<typename DATA>
l4_umword_t L4virtio::Svr::Driver_mem_region_t< DATA >::size () const [inline]
```

Returns

The size of the region in bytes.

Definition at line 577 of file [l4virtio](#).

The documentation for this class was generated from the following file:

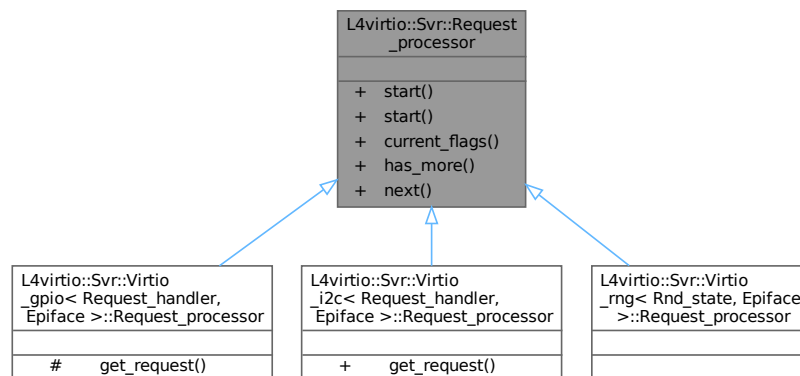
- [l4/l4virtio/server/l4virtio](#)

16.417 L4virtio::Svr::Request_processor Class Reference

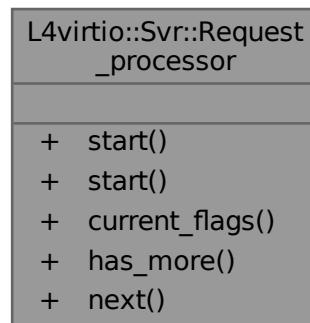
Encapsulate the state for processing a VIRTIO request.

```
#include <virtio>
```

Inheritance diagram for L4virtio::Svr::Request_processor:



Collaboration diagram for L4virtio::Svr::Request_processor:



Public Member Functions

- `template<typename DESC_MAN, typename ... ARGS>`
`void start (DESC_MAN *dm, Virtqueue *ring, Virtqueue::Head_desc const &request, ARGS... args)`
Start processing a new request.
- `template<typename DESC_MAN, typename ... ARGS>`
`Virtqueue::Request const & start (DESC_MAN *dm, Virtqueue::Request const &request, ARGS... args)`
Start processing a new request.
- `Virtqueue::Desc::Flags current_flags () const`
Get the flags of the currently processed descriptor.
- `bool has_more () const`
Are there more chained descriptors?
- `template<typename DESC_MAN, typename ... ARGS>`
`bool next (DESC_MAN *dm, ARGS... args)`
Switch to the next descriptor in a descriptor chain.

16.417.1 Detailed Description

Encapsulate the state for processing a VIRTIO request.

A VIRTIO request is a possibly chained list of descriptors retrieved from the available ring of a virtqueue, using `Virtqueue::next_avail()`.

The descriptor processing depends on helper (DESC_MAN) for interpreting the descriptors in the context of the device implementation.

DESC_MAN has to provide the functionality to safely dereference a descriptor from a descriptor list.

The following methods must be provided by DESC_MAN:

- `DESC_MAN::load_desc (Virtqueue::Desc const &desc,`
`Request_processor const *proc,`
`Virtqueue::Desc const **table)`

This function is used to dereference `desc` as an indirect descriptor table, and must return a pointer to an indirect descriptor table.

- `DESC_MAN::load_desc(Virtqueue::Desc const &desc, Request_processor const *proc, ...)`

This function is used to dereference a descriptor as a normal data buffer, and '...' are the arguments that are passed to `start()` and `next()`.

Definition at line 472 of file [virtio](#).

16.417.2 Member Function Documentation

16.417.2.1 `current_flags()`

```
Virtqueue::Desc::Flags L4virtio::Svr::Request_processor::current_flags () const [inline]
```

Get the flags of the currently processed descriptor.

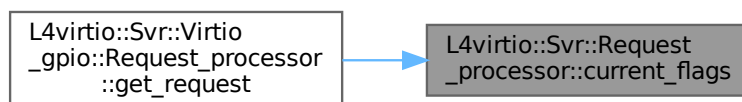
Returns

The flags of the currently processed descriptor.

Definition at line 545 of file [virtio](#).

Referenced by [L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::Request_processor::get_request\(\)](#).

Here is the caller graph for this function:



16.417.2.2 `has_more()`

```
bool L4virtio::Svr::Request_processor::has_more () const [inline]
```

Are there more chained descriptors?

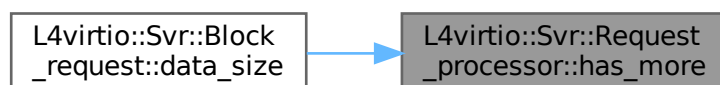
Returns

true if there are more chained descriptors in the current request.

Definition at line 553 of file [virtio](#).

Referenced by [L4virtio::Svr::Block_request< Ds_data >::data_size\(\)](#).

Here is the caller graph for this function:



16.417.2.3 next()

```
template<typename DESC_MAN, typename ... ARGS>
bool L4virtio::Svr::Request_processor::next (
    DESC_MAN * dm,
    ARGS... args) [inline]
```

Switch to the next descriptor in a descriptor chain.

Template Parameters

<i>DESC_MAN</i>	Type of descriptor manager (implicit).
-----------------	--

Parameters

<i>dm</i>	Descriptor manager that is used to translate VIRTIO descriptor addresses.
<i>args</i>	Extra arguments passed to dm->load_desc()

Return values

<i>true</i>	A next descriptor is available.
<i>false</i>	No descriptor available.

Exceptions

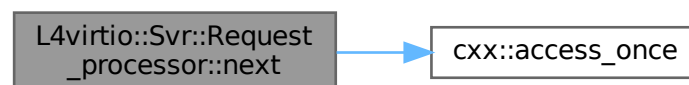
<i>Bad_descriptor</i>	The next index of this descriptor is invalid.
---------------------------------------	---

Definition at line 570 of file [virtio](#).

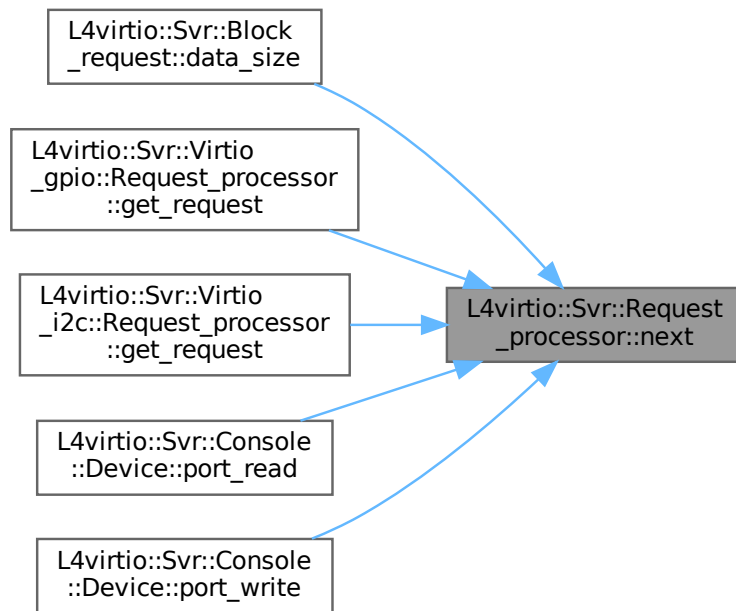
References [cxx::access_once\(\)](#), [L4virtio::Svr::Bad_descriptor::Bad_flags](#), [L4virtio::Svr::Bad_descriptor::Bad_next](#), and [L4_UNLIKELY](#).

Referenced by [L4virtio::Svr::Block_request<Ds_data>::data_size\(\)](#), [L4virtio::Svr::Virtio_gpio<Request_handler, Epiface>::Request_handler::get_request\(\)](#), [L4virtio::Svr::Virtio_i2c<Request_handler, Epiface>::Request_processor::get_request\(\)](#), [L4virtio::Svr::Console::Device::port_read\(\)](#) and [L4virtio::Svr::Console::Device::port_write\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.417.2.4 start() [1/2]

```

template<typename DESC_MAN, typename ... ARGS>
void L4virtio::Svr::Request_processor::start (
    DESC_MAN * dm,
    Virtqueue * ring,
    Virtqueue::Head_desc const & request,
    ARGS... args) [inline]

```

Start processing a new request.

Template Parameters

<i>DESC_MAN</i>	Type of descriptor manager (implicit).
-----------------	--

Parameters

<i>dm</i>	Descriptor manager that is used to translate VIRTIO descriptor addresses.
<i>ring</i>	VIRTIO ring of the request.
<i>request</i>	VIRTIO request from Virtqueue::next_avail()
<i>args</i>	Extra arguments passed to <code>dm->load_desc()</code>

Precondition

The given request must be valid.

Exceptions

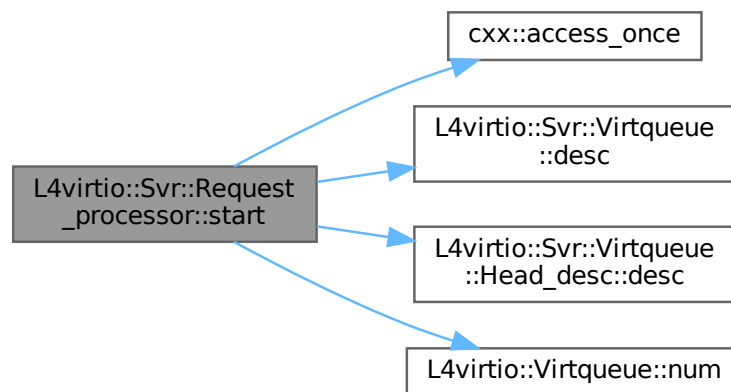
Bad_descriptor	The descriptor has an invalid size or <code>load_desc()</code> has thrown an exception by itself.
--------------------------------	---

Definition at line 501 of file [virtio](#).

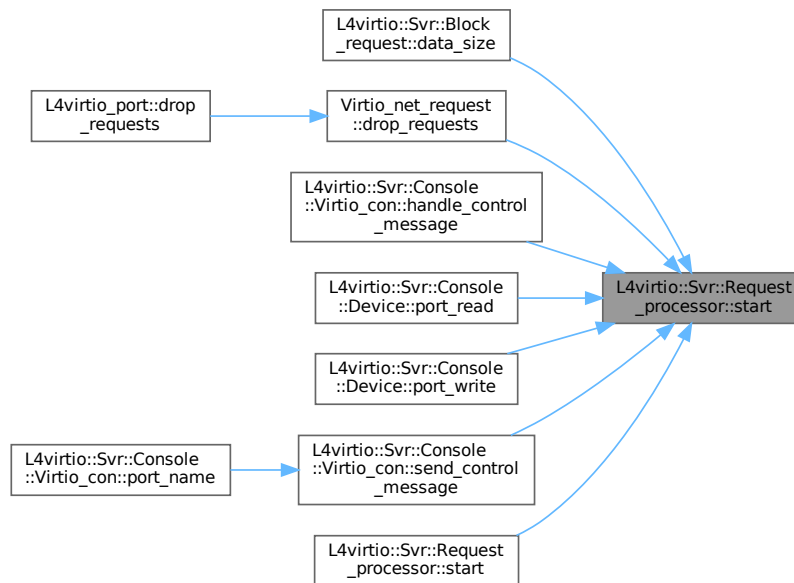
References [cxx::access_once\(\)](#), [L4virtio::Svr::Bad_descriptor::Bad_size](#), [L4virtio::Svr::Virtqueue::desc\(\)](#), [L4virtio::Svr::Virtqueue::Head_desc::desc\(\)](#), [L4_UNLIKELY](#), and [L4virtio::Virtqueue::num\(\)](#).

Referenced by [L4virtio::Svr::Block_request<Ds_data>::data_size\(\)](#), [Virtio_net_request::drop_requests\(\)](#), [L4virtio::Svr::Console::Virtio_con::handle_control_message\(\)](#), [L4virtio::Svr::Console::Device::port_read\(\)](#), [L4virtio::Svr::Console::Device::port_write\(\)](#), [L4virtio::Svr::Console::Virtio_con::send_control_message\(\)](#), and [start\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



16.417.2.5 start() [2/2]

```

template<typename DESC_MAN, typename ... ARGS>
Virtqueue::Request const & L4virtio::Svr::Request_processor::start (
    DESC_MAN * dm,
    Virtqueue::Request const & request,
    ARGS... args) [inline]

```

Start processing a new request.

Template Parameters

<i>DESC_MAN</i>	Type of descriptor manager (implicit).
-----------------	--

Parameters

<i>dm</i>	Descriptor manager that is used to translate VIRTIO descriptor addresses.
<i>request</i>	VIRTIO request from Virtqueue::next_avail()
<i>args</i>	Extra arguments passed to <code>dm->load_desc()</code>

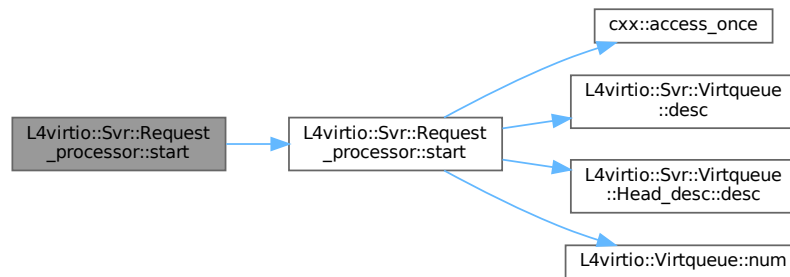
Precondition

The given request must be valid.

Definition at line 534 of file [virtio](#).

References [start\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

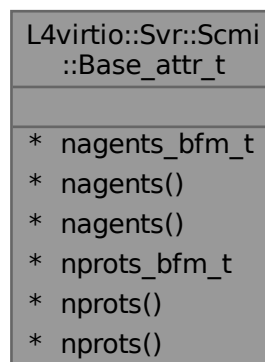
- `l4/l4virtio/server/virtio`

16.418 L4virtio::Svr::Scmi::Base_attr_t Struct Reference

SCMI base protocol attributes.

```
#include <virtio-scmi-device>
```

Collaboration diagram for L4virtio::Svr::Scmi::Base_attr_t:



- typedef [cxx::Bitfield](#)< decltype(attr_raw), 8, 15 > **nagents_bfm_t**
Type to access the *nagents* bits (8 to 15) of *attr_raw*.
- constexpr [nagents_bfm_t::Val](#) **nagents** () const
Get the *nagents* bits (8 to 15) of *attr_raw*.
- constexpr [nagents_bfm_t::Ref](#) **nagents** ()
Get a reference to the *nagents* bits (8 to 15) of *attr_raw*.

- typedef [cxx::Bitfield](#)< decltype(attr_raw), 0, 7 > **nprots_bfm_t**
Type to access the *nprots* bits (0 to 7) of *attr_raw*.
- constexpr [nprots_bfm_t::Val](#) **nprots** () const
Get the *nprots* bits (0 to 7) of *attr_raw*.
- constexpr [nprots_bfm_t::Ref](#) **nprots** ()
Get a reference to the *nprots* bits (0 to 7) of *attr_raw*.

16.418.1 Detailed Description

SCMI base protocol attributes.

Definition at line 90 of file [virtio-scmi-device](#).

The documentation for this struct was generated from the following file:

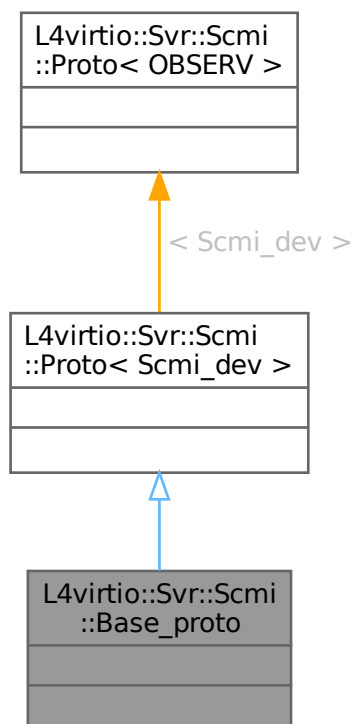
- I4/I4virtio/server/virtio-scmi-device

16.419 L4virtio::Svr::Scmi::Base_proto Class Reference

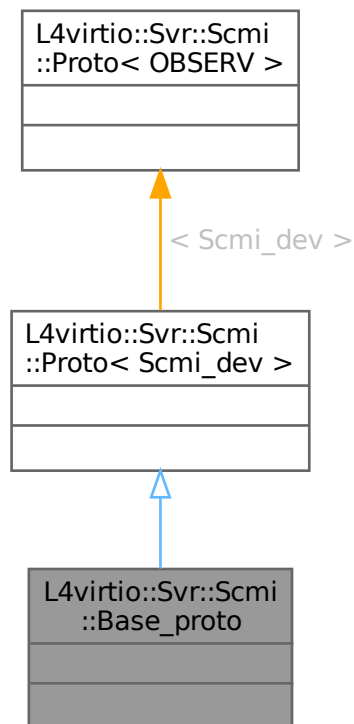
Base class for the SCMI base protocol.

```
#include <virtio-scmi-device>
```

Inheritance diagram for L4virtio::Svr::Scmi::Base_proto:



Collaboration diagram for L4virtio::Svr::Scmi::Base_proto:



16.419.1 Detailed Description

Base class for the SCMI base protocol.

Use this class as a base to implement the base protocol.

Definition at line 458 of file [virtio-scmi-device](#).

The documentation for this class was generated from the following file:

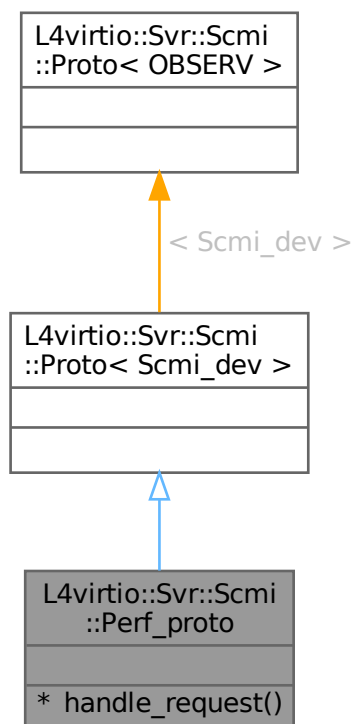
- I4/I4virtio/server/virtio-scmi-device

16.420 L4virtio::Svr::Scmi::Perf_proto Class Reference

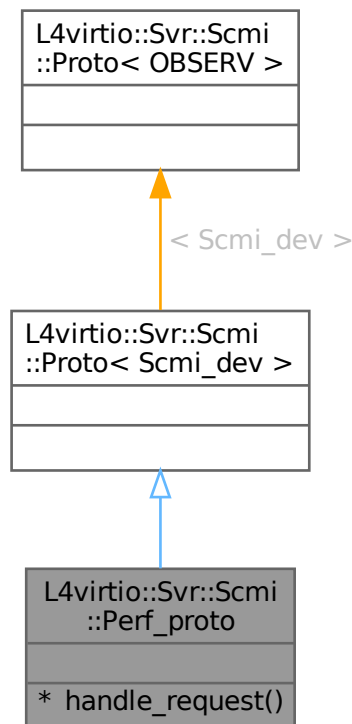
Base class for the SCMI performance protocol.

```
#include <virtio-scmi-device>
```

Inheritance diagram for L4virtio::Svr::Scmi::Perf_proto:



Collaboration diagram for L4virtio::Svr::Scmi::Perf_proto:



16.420.1 Detailed Description

Base class for the SCMI performance protocol.

Use this class as a base to implement the performance protocol.

If you want to use this from a Uvmm Linux guest, the device tree needs to look something like this:

```

firmware {
    scmi {
        compatible = "arm,scmi-virtio";

        #address-cells = <1>;
        #size-cells = <0>;

        cpufreq: protocol@13 {
            reg = <0x13>;
            #clock-cells = <1>;
        };
    };
};
....

cpu@0 {
    device_type = "cpu";
    reg = <0x0>;
    clocks = <&cpufreq 0>; // domain_id
};

```

Definition at line 662 of file [virtio-scmi-device](#).

The documentation for this class was generated from the following file:

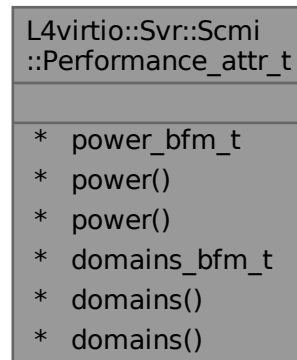
- `I4/I4virtio/server/virtio-scmi-device`

16.421 L4virtio::Svr::Scmi::Performance_attr_t Struct Reference

SCMI performance protocol attributes.

```
#include <virtio-scmi-device>
```

Collaboration diagram for L4virtio::Svr::Scmi::Performance_attr_t:



- typedef [cxx::Bitfield](#)< decltype(attr_raw), 16, 16 > **power_bfm_t**
Type to access the [power](#) bits (16 to 16) of *attr_raw*.
- constexpr [power_bfm_t::Val](#) **power** () const
Get the [power](#) bits (16 to 16) of *attr_raw*.
- constexpr [power_bfm_t::Ref](#) **power** ()
Get a reference to the [power](#) bits (16 to 16) of *attr_raw*.
- typedef [cxx::Bitfield](#)< decltype(attr_raw), 0, 15 > **domains_bfm_t**
Type to access the [domains](#) bits (0 to 15) of *attr_raw*.
- constexpr [domains_bfm_t::Val](#) **domains** () const
Get the [domains](#) bits (0 to 15) of *attr_raw*.
- constexpr [domains_bfm_t::Ref](#) **domains** ()
Get a reference to the [domains](#) bits (0 to 15) of *attr_raw*.

16.421.1 Detailed Description

SCMI performance protocol attributes.

Definition at line 112 of file [virtio-scmi-device](#).

The documentation for this struct was generated from the following file:

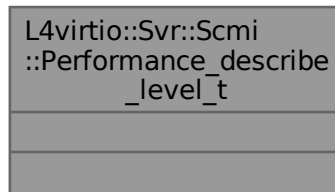
- I4/I4virtio/server/virtio-scmi-device

16.422 L4virtio::Svr::Scmi::Performance_describe_level_t Struct Reference

SCMI performance describe level.

```
#include <virtio-scmi-device>
```

Collaboration diagram for L4virtio::Svr::Scmi::Performance_describe_level_t:



16.422.1 Detailed Description

SCMI performance describe level.

Definition at line 150 of file [virtio-scmi-device](#).

The documentation for this struct was generated from the following file:

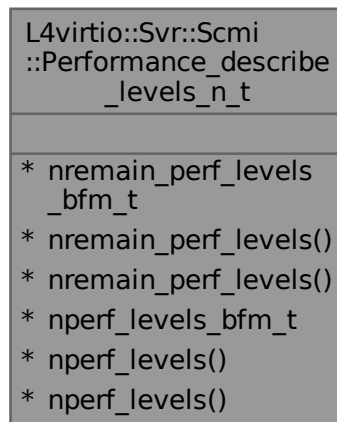
- I4/I4virtio/server/virtio-scmi-device

16.423 L4virtio::Svr::Scmi::Performance_describe_levels_n_t Struct Reference

SCMI performance describe levels numbers.

```
#include <virtio-scmi-device>
```

Collaboration diagram for L4virtio::Svr::Scmi::Performance_describe_levels_n_t:



- typedef [cxx::Bitfield](#)< decltype(num_levels_raw), 16, 31 > **nremain_perf_levels_bfm_t**
Type to access the [nremain_perf_levels](#) bits (16 to 31) of num_levels_raw.
- constexpr [nremain_perf_levels_bfm_t::Val](#) **nremain_perf_levels** () const
Get the [nremain_perf_levels](#) bits (16 to 31) of num_levels_raw.
- constexpr [nremain_perf_levels_bfm_t::Ref](#) **nremain_perf_levels** ()
Get a reference to the [nremain_perf_levels](#) bits (16 to 31) of num_levels_raw.

- typedef [cxx::Bitfield](#)< decltype(num_levels_raw), 0, 11 > **nperf_levels_bfm_t**
Type to access the [nperf_levels](#) bits (0 to 11) of num_levels_raw.
- constexpr [nperf_levels_bfm_t::Val](#) **nperf_levels** () const
Get the [nperf_levels](#) bits (0 to 11) of num_levels_raw.
- constexpr [nperf_levels_bfm_t::Ref](#) **nperf_levels** ()
Get a reference to the [nperf_levels](#) bits (0 to 11) of num_levels_raw.

16.423.1 Detailed Description

SCMI performance describe levels numbers.

Definition at line 142 of file [virtio-scmi-device](#).

The documentation for this struct was generated from the following file:

- l4/l4virtio/server/virtio-scmi-device

16.424 L4virtio::Svr::Scmi::Performance_domain_attr_t Struct Reference

SCMI performance domain protocol attributes.

```
#include <virtio-scmi-device>
```

Collaboration diagram for L4virtio::Svr::Scmi::Performance_domain_attr_t:

L4virtio::Svr::Scmi ::Performance_domain _attr_t
<ul style="list-style-type: none"> * set_limits_bfm_t * set_limits() * set_limits() * set_perf_level_bfm_t * set_perf_level() * set_perf_level() * perf_limits_change_notify_bfm_t * perf_limits_change_notify() * perf_limits_change_notify() * perf_level_change_notify_bfm_t * perf_level_change_notify() * perf_level_change_notify() * fast_channel_bfm_t * fast_channel() * fast_channel() * rate_limit_bfm_t * rate_limit() * rate_limit()

- typedef [cxx::Bitfield](#)< decltype(attr_raw), 31, 31 > **set_limits_bfm_t**
Type to access the [set_limits](#) bits (31 to 31) of *attr_raw*.
- constexpr [set_limits_bfm_t::Val](#) **set_limits** () const
Get the [set_limits](#) bits (31 to 31) of *attr_raw*.
- constexpr [set_limits_bfm_t::Ref](#) **set_limits** ()
Get a reference to the [set_limits](#) bits (31 to 31) of *attr_raw*.

- typedef [cxx::Bitfield](#)< decltype(attr_raw), 30, 30 > **set_perf_level_bfm_t**
Type to access the [set_perf_level](#) bits (30 to 30) of attr_raw.
- constexpr [set_perf_level_bfm_t::Val](#) **set_perf_level** () const
Get the [set_perf_level](#) bits (30 to 30) of attr_raw.
- constexpr [set_perf_level_bfm_t::Ref](#) **set_perf_level** ()
Get a reference to the [set_perf_level](#) bits (30 to 30) of attr_raw.

- typedef [cxx::Bitfield](#)< decltype(attr_raw), 29, 29 > **perf_limits_change_notify_bfm_t**
Type to access the [perf_limits_change_notify](#) bits (29 to 29) of attr_raw.
- constexpr [perf_limits_change_notify_bfm_t::Val](#) **perf_limits_change_notify** () const
Get the [perf_limits_change_notify](#) bits (29 to 29) of attr_raw.
- constexpr [perf_limits_change_notify_bfm_t::Ref](#) **perf_limits_change_notify** ()
Get a reference to the [perf_limits_change_notify](#) bits (29 to 29) of attr_raw.

- typedef [cxx::Bitfield](#)< decltype(attr_raw), 28, 28 > **perf_level_change_notify_bfm_t**
Type to access the [perf_level_change_notify](#) bits (28 to 28) of attr_raw.
- constexpr [perf_level_change_notify_bfm_t::Val](#) **perf_level_change_notify** () const
Get the [perf_level_change_notify](#) bits (28 to 28) of attr_raw.
- constexpr [perf_level_change_notify_bfm_t::Ref](#) **perf_level_change_notify** ()
Get a reference to the [perf_level_change_notify](#) bits (28 to 28) of attr_raw.

- typedef [cxx::Bitfield](#)< decltype(attr_raw), 27, 27 > **fast_channel_bfm_t**
Type to access the [fast_channel](#) bits (27 to 27) of attr_raw.
- constexpr [fast_channel_bfm_t::Val](#) **fast_channel** () const
Get the [fast_channel](#) bits (27 to 27) of attr_raw.
- constexpr [fast_channel_bfm_t::Ref](#) **fast_channel** ()
Get a reference to the [fast_channel](#) bits (27 to 27) of attr_raw.

- typedef [cxx::Bitfield](#)< decltype(rate_limit_raw), 0, 19 > **rate_limit_bfm_t**
Type to access the [rate_limit](#) bits (0 to 19) of rate_limit_raw.
- constexpr [rate_limit_bfm_t::Val](#) **rate_limit** () const
Get the [rate_limit](#) bits (0 to 19) of rate_limit_raw.
- constexpr [rate_limit_bfm_t::Ref](#) **rate_limit** ()
Get a reference to the [rate_limit](#) bits (0 to 19) of rate_limit_raw.

16.424.1 Detailed Description

SCMI performance domain protocol attributes.

Definition at line 124 of file [virtio-scmi-device](#).

The documentation for this struct was generated from the following file:

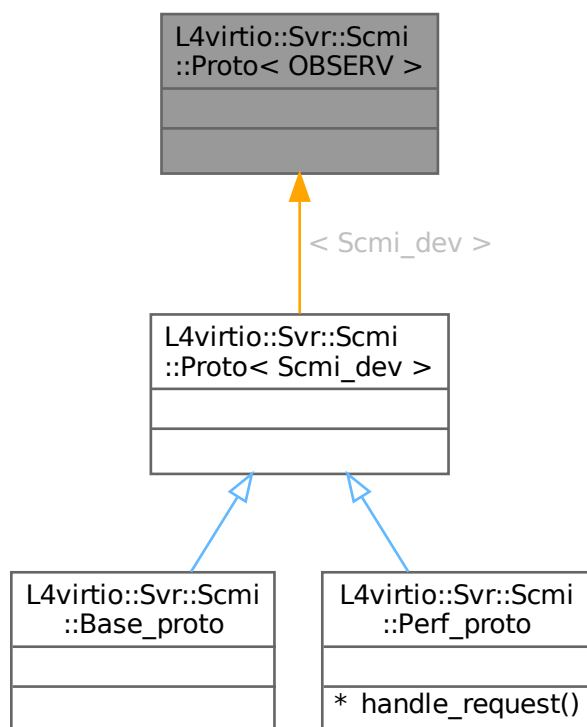
- I4/I4virtio/server/virtio-scmi-device

16.425 L4virtio::Svr::Scmi::Proto< OBSERV > Struct Template Reference

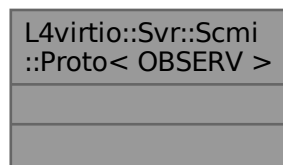
Base class for all protocols.

```
#include <virtio-scmi-device>
```

Inheritance diagram for L4virtio::Svr::Scmi::Proto< OBSERV >:



Collaboration diagram for L4virtio::Svr::Scmi::Proto< OBSERV >:



16.425.1 Detailed Description

```
template<typename OBSERV>
struct L4virtio::Svr::Scmi::Proto< OBSERV >
```

Base class for all protocols.

Defines an interface for processing the virtio buffers for the implemented protocol.

Definition at line 299 of file [virtio-scmi-device](#).

The documentation for this struct was generated from the following file:

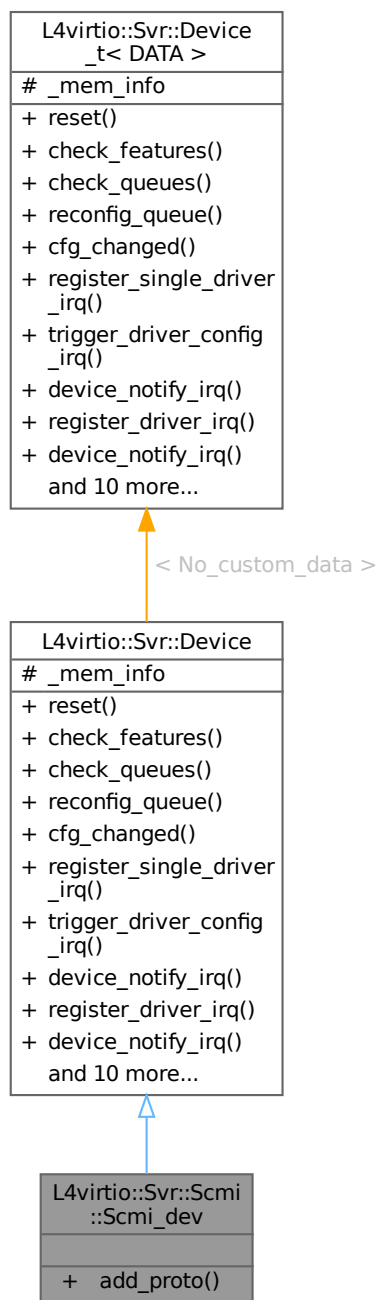
- l4/l4virtio/server/virtio-scmi-device

16.426 L4virtio::Svr::Scmi::Scmi_dev Class Reference

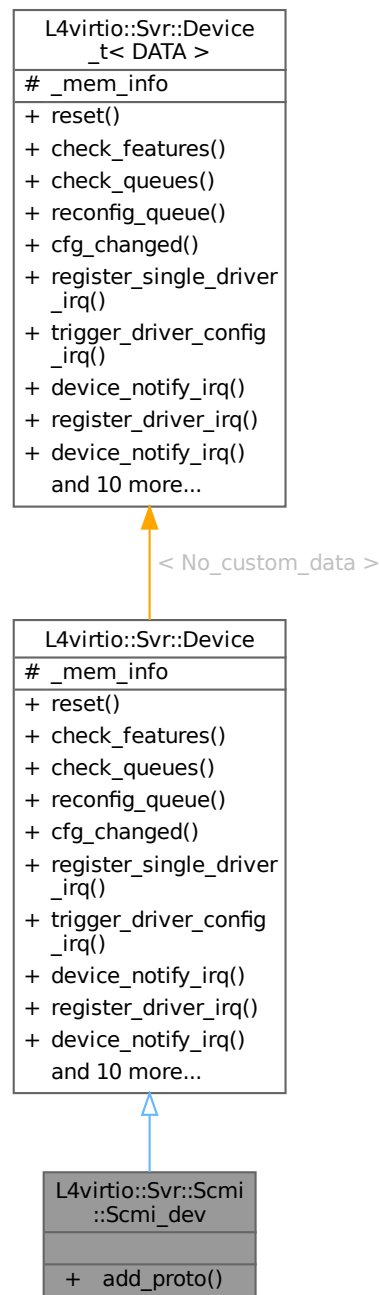
A server implementation of the virtio-scmi protocol.

```
#include <virtio-scmi-device>
```

Inheritance diagram for L4virtio::Svr::Scmi::Scmi_dev:



Collaboration diagram for L4virtio::Svr::Scmi::Scmi_dev:



Public Member Functions

- void **add_proto** ([l4_uint32_t](#) id, [Proto](#)< [Scmi_dev](#) > *proto)

Add an actual protocol implementation with the given id to the server.

Public Member Functions inherited from [L4virtio::Svr::Device_t< No_custom_data >](#)

- virtual bool **check_features** ()

- callback for checking the subset of accepted features*
- virtual void **cfg_changed** (unsigned)
 - callback for client device configuration changes*
- virtual void **register_driver_irq** (unsigned idx)
 - Callback for registering an notification IRQ (multi IRQ).*
- virtual L4::Cap< L4::Irq > **device_notify_irq** (unsigned idx)
 - Callback to gather the device notification IRQ (multi IRQ).*
- virtual unsigned **num_events_supported** () const
 - Return the highest notification index supported.*
- **Device_t** (Dev_config *dev_config)
 - Make a device for the given config.*
- **Mem_list** const * **mem_info** () const
 - Get the memory region list used for this device.*
- void **reset_queue_config** (unsigned idx, unsigned num_max, bool inc_generation=false)
 - Trigger reset for the configuration space for queue idx.*
- void **init_mem_info** (unsigned num)
 - Initialize the memory region list to the given maximum.*
- void **device_error** ()
 - Transition device into DEVICE_NEEDS_RESET state.*
- bool **setup_queue** (Virtqueue *q, unsigned qn, unsigned num_max)
 - Enable/disable the specified queue.*
- bool **handle_mem_cmd_write** ()
 - Check for a value in the cmd register and handle a write.*
- void **enable_trusted_ds_validation** ()
 - Enable trusted dataspace validation.*
- void **add_trusted_dataspaces** (std::shared_ptr< Ds_vector const > ds)
 - Provide a list of trusted dataspaces that can be used for validation.*

Additional Inherited Members

Protected Attributes inherited from L4virtio::Svr::Device_t< No_custom_data >

- **Mem_list _mem_info**
 - Memory region list.*

16.426.1 Detailed Description

A server implementation of the virtio-scmi protocol.

Use this class as a base to implement your own specific SCMI device.

SCMI defines multiple protocols which can be optionally handled. This server implementation is flexible enough to handle any combination of them. The user of this server has to deviate from the provided [Proto](#) classes (for the protocols he want to handle) and needs to implement the required callbacks.

Right now, support for the base and the performance protocol is provided.

The base protocol is mandatory.

If you want to use this from a Uvmm Linux guest, the device tree needs to look something like this:

```

firmware {
    scmi {
        compatible = "arm,scmi-virtio";

        #address-cells = <1>;
        #size-cells = <0>;

        // ... supported protocols ...
    };
};

```

Definition at line 336 of file [virtio-scmi-device](#).

The documentation for this class was generated from the following file:

- I4/I4virtio/server/virtio-scmi-device

16.427 L4virtio::Svr::Scmi::Scmi_hdr_t Struct Reference

SCMI header.

```
#include <virtio-scmi-device>
```

Collaboration diagram for L4virtio::Svr::Scmi::Scmi_hdr_t:

L4virtio::Svr::Scmi ::Scmi_hdr_t
<ul style="list-style-type: none"> * token_bfm_t * token() * token() * protocol_id_bfm_t * protocol_id() * protocol_id() * message_type_bfm_t * message_type() * message_type() * message_id_bfm_t * message_id() * message_id()

- typedef [cxx::Bitfield](#)< decltype(hdr_raw), 18, 27 > **token_bfm_t**
Type to access the *token* bits (18 to 27) of *hdr_raw*.
- constexpr [token_bfm_t::Val](#) **token** () const
Get the *token* bits (18 to 27) of *hdr_raw*.

- constexpr [token_bfm_t::Ref](#) token ()
Get a reference to the [token](#) bits (18 to 27) of [hdr_raw](#).
- typedef [cxx::Bitfield](#)< decltype(hdr_raw), 10, 17 > **protocol_id_bfm_t**
Type to access the [protocol_id](#) bits (10 to 17) of [hdr_raw](#).
- constexpr [protocol_id_bfm_t::Val](#) protocol_id () const
Get the [protocol_id](#) bits (10 to 17) of [hdr_raw](#).
- constexpr [protocol_id_bfm_t::Ref](#) protocol_id ()
Get a reference to the [protocol_id](#) bits (10 to 17) of [hdr_raw](#).
- typedef [cxx::Bitfield](#)< decltype(hdr_raw), 8, 9 > **message_type_bfm_t**
Type to access the [message_type](#) bits (8 to 9) of [hdr_raw](#).
- constexpr [message_type_bfm_t::Val](#) message_type () const
Get the [message_type](#) bits (8 to 9) of [hdr_raw](#).
- constexpr [message_type_bfm_t::Ref](#) message_type ()
Get a reference to the [message_type](#) bits (8 to 9) of [hdr_raw](#).
- typedef [cxx::Bitfield](#)< decltype(hdr_raw), 0, 7 > **message_id_bfm_t**
Type to access the [message_id](#) bits (0 to 7) of [hdr_raw](#).
- constexpr [message_id_bfm_t::Val](#) message_id () const
Get the [message_id](#) bits (0 to 7) of [hdr_raw](#).
- constexpr [message_id_bfm_t::Ref](#) message_id ()
Get a reference to the [message_id](#) bits (0 to 7) of [hdr_raw](#).

16.427.1 Detailed Description

SCMI header.

Definition at line 45 of file [virtio-scmi-device](#).

The documentation for this struct was generated from the following file:

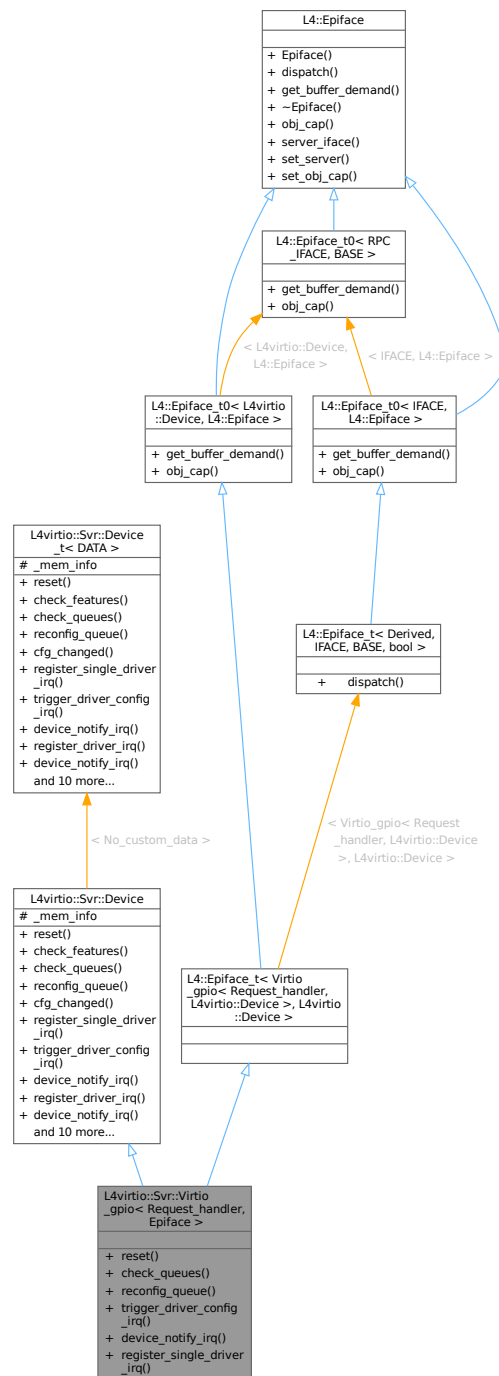
- `l4/l4virtio/server/virtio-scmi-device`

16.428 L4virtio::Svr::Virtio_gpio< Request_handler, Epiface > Class Template Reference

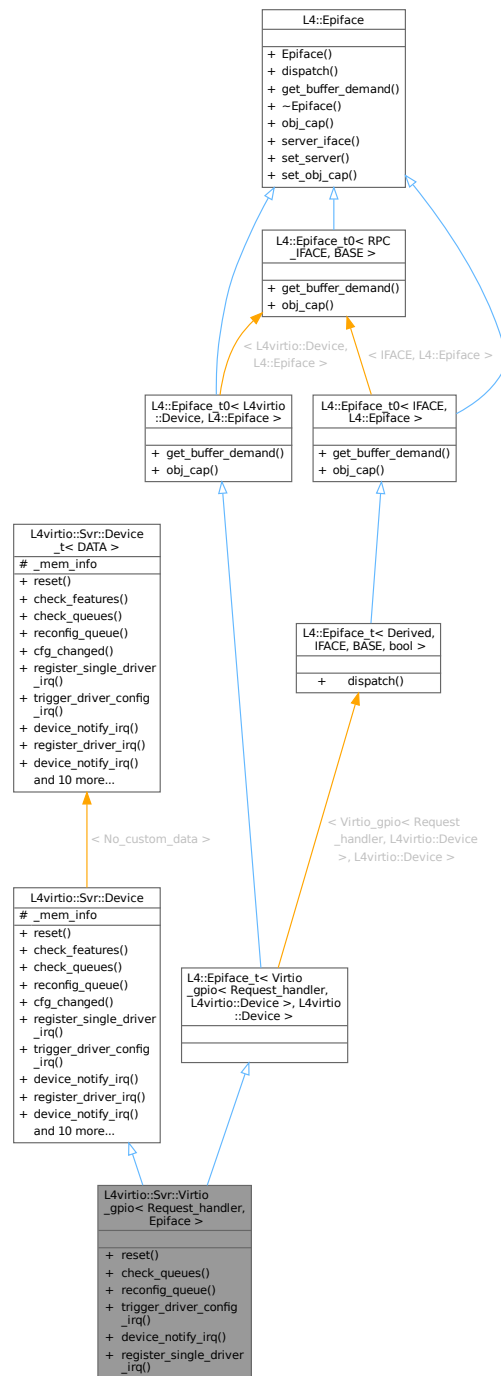
A server implementation of the virtio-gpio protocol.

```
#include <virtio-gpio-device>
```

Inheritance diagram for L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >:



Collaboration diagram for L4virtio::Svr::Virtio_gpio< Request_handler, Epiface > :



Data Structures

- struct [Irq_handler](#)
Handler for an gpio pin irq.
- struct [Host_irq](#)
Handler for the host irq.
- struct [Request_processor](#)
Generic handler for the Virtio requests.

Public Member Functions

- void **reset** () override
reset callback, called for doing a device reset
- bool **check_queues** () override
callback for checking if the queues at DRIVER_OK transition
- int **reconfig_queue** (unsigned idx) override
callback for client queue-config request
- void **trigger_driver_config_irq** () override
callback for triggering configuration change notification IRQ
- [L4::Cap](#) < [L4::lrq](#) > **device_notify_irq** () const override
callback to gather the device notification IRQ (old-style)
- void **register_single_driver_irq** () override
callback for registering a single guest IRQ for all queues (old-style)

Public Member Functions inherited from [L4virtio::Svr::Device_t](#) < [No_custom_data](#) >

- virtual bool **check_features** ()
callback for checking the subset of accepted features
- virtual void **cfg_changed** (unsigned)
callback for client device configuration changes
- virtual void **register_driver_irq** (unsigned idx)
Callback for registering an notification IRQ (multi IRQ).
- virtual [L4::Cap](#) < [L4::lrq](#) > **device_notify_irq** (unsigned idx)
Callback to gather the device notification IRQ (multi IRQ).
- virtual unsigned **num_events_supported** () const
Return the highest notification index supported.
- [Device_t](#) ([Dev_config](#) *dev_config)
Make a device for the given config.
- [Mem_list](#) const * **mem_info** () const
Get the memory region list used for this device.
- void **reset_queue_config** (unsigned idx, unsigned num_max, bool inc_generation=false)
Trigger reset for the configuration space for queue idx.
- void **init_mem_info** (unsigned num)
Initialize the memory region list to the given maximum.
- void **device_error** ()
Transition device into DEVICE_NEEDS_RESET state.
- bool **setup_queue** ([Virtqueue](#) *q, unsigned qn, unsigned num_max)
Enable/disable the specified queue.
- bool **handle_mem_cmd_write** ()
Check for a value in the cmd register and handle a write.
- void **enable_trusted_ds_validation** ()
Enable trusted dataspace validation.
- void **add_trusted_dataspaces** (std::shared_ptr < [Ds_vector](#) const > ds)
Provide a list of trusted dataspaces that can be used for validation.

Public Member Functions inherited from [L4::Epiface_t0](#) < [L4virtio::Device](#), [L4::Epiface](#) >

- [Type_info::Demand](#) **get_buffer_demand** () const
Get the server-side buffer demand based in IFACE.
- [Cap](#) < [L4virtio::Device](#) > **obj_cap** () const
Get the (typed) capability to this object.

Public Member Functions inherited from L4::Epiface

- **Epiface** ()
Make a server object.
- virtual ~**Epiface** ()=0
Destroy the object.
- Stored_cap **obj_cap** () const
Get the capability to the kernel object belonging to this object.
- **Server_iface** * **server_iface** () const
Get pointer to server interface at which the object is currently registered.
- int **set_server** (**Server_iface** *srv, **Cap**< void > cap, bool managed=false)
Set server registration info for the object.
- void **set_obj_cap** (**Cap**< void > const &cap)
Deprecated server registration function.

Additional Inherited Members

Public Types inherited from L4::Epiface_t0< L4virtio::Device, L4::Epiface >

- typedef L4virtio::Device **Interface**
Data type of the IPC interface definition.

Public Types inherited from L4::Epiface

- typedef **lpc_svr::Server_iface** **Server_iface**
Type for abstract server interface.
- typedef **lpc_svr::Server_iface::Demand** **Demand**
Type for server-side receive buffer demand.

Protected Attributes inherited from L4virtio::Svr::Device_t< No_custom_data >

- **Mem_list** **_mem_info**
Memory region list.

16.428.1 Detailed Description

```
template<typename Request_handler, typename Epiface = L4virtio::Device>
class L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >
```

A server implementation of the virtio-gpio protocol.

Template Parameters

<i>Request_handler</i>	<p>The type that is used to handle incoming requests. Needs to have:</p> <ul style="list-style-type: none"> • <code>bool get_direction(l4_uint16_t gpio, l4_uint8_t *dir)</code> • <code>bool set_direction(l4_uint16_t gpio, l4_uint8_t dir)</code> • <code>bool get_value(l4_uint16_t gpio, l4_uint8_t *val)</code> • <code>bool set_value(l4_uint16_t gpio, l4_uint8_t val)</code> • <code>bool set_irq_type(l4_uint16_t gpio, l4_uint8_t mode)</code> • <code>bool enable_irq(l4_uint16_t gpio, std::shared_ptr<Virtio_gpio::Irq_handler> const &hdl)</code> functions.
<i>Epiface</i>	The Epiface to derive from. Defaults to L4virtio::Device .

Definition at line 142 of file [virtio-gpio-device](#).

The documentation for this class was generated from the following file:

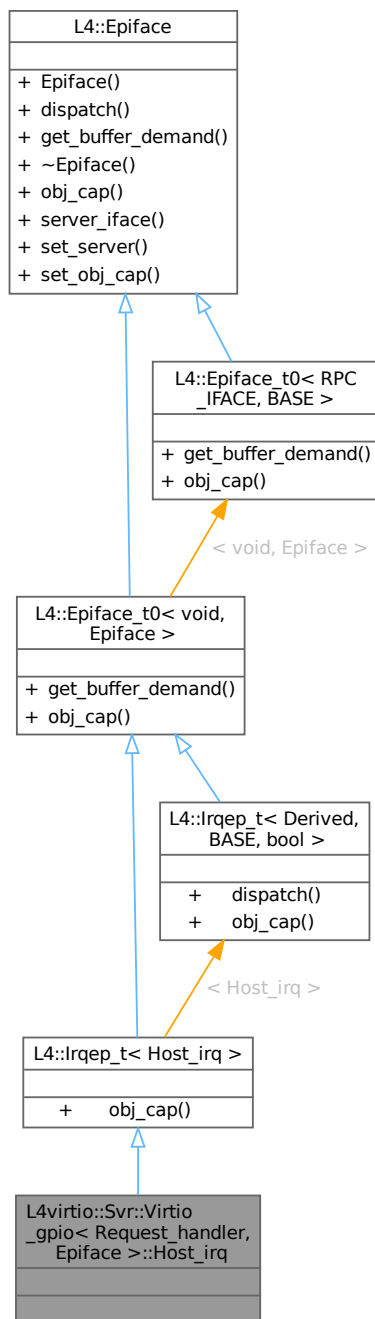
- `l4/l4virtio/server/virtio-gpio-device`

16.429 L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::Host_irq Struct Reference

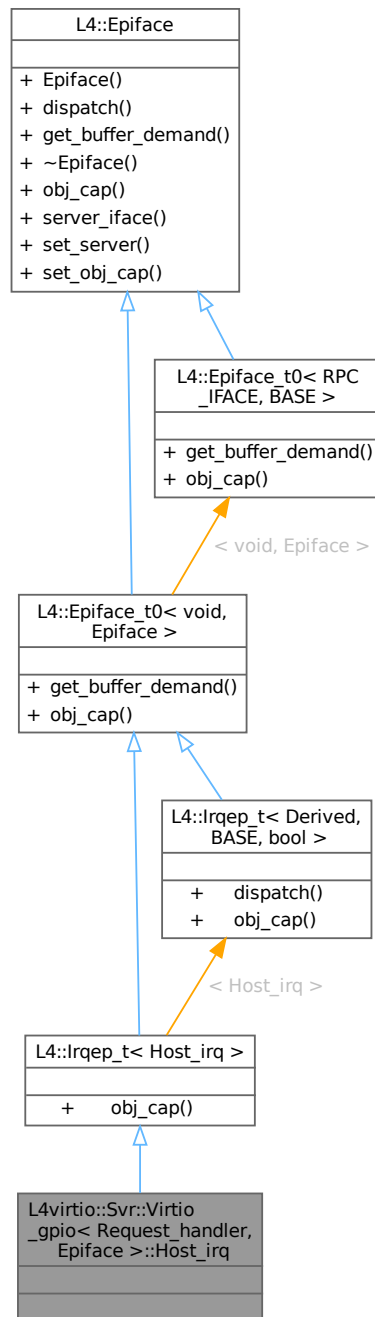
Handler for the host irq.

```
#include <virtio-gpio-device>
```

Inheritance diagram for L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::Host_irq:



Collaboration diagram for L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::Host_irq:



Additional Inherited Members

Public Types inherited from **L4::Epiface_t0< void, Epiface >**

- typedef void **Interface**

Data type of the IPC interface definition.

Public Types inherited from L4::Epiface

- typedef [lpc_svr::Server_iface](#) **Server_iface**
Type for abstract server interface.
- typedef [lpc_svr::Server_iface::Demand](#) **Demand**
Type for server-side receive buffer demand.

Public Member Functions inherited from L4::lrqep_t< Host_irq >

- [Cap< L4::lrq > obj_cap](#) () const
Get the (typed) capability to this object.

Public Member Functions inherited from L4::Epiface_t0< void, Epiface >

- [Type_info::Demand](#) **get_buffer_demand** () const
Get the server-side buffer demand based in IFACE.
- [Cap< void > obj_cap](#) () const
Get the (typed) capability to this object.

Public Member Functions inherited from L4::Epiface

- **Epiface** ()
Make a server object.
- virtual **~Epiface** ()=0
Destroy the object.
- Stored_cap [obj_cap](#) () const
Get the capability to the kernel object belonging to this object.
- [Server_iface](#) * [server_iface](#) () const
Get pointer to server interface at which the object is currently registered.
- int **set_server** ([Server_iface](#) *srv, [Cap< void > cap](#), bool managed=false)
Set server registration info for the object.
- void **set_obj_cap** ([Cap< void > const &cap](#))
Deprecated server registration function.

16.429.1 Detailed Description

```
template<typename Request_handler, typename Epiface = L4virtio::Device>
struct L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::Host_irq
```

Handler for the host irq.

An [L4::lrqep_t](#) to handle irq's send to the server.

Definition at line 198 of file [virtio-gpio-device](#).

The documentation for this struct was generated from the following file:

- l4/l4virtio/server/virtio-gpio-device

16.430 L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::Irq_handler Struct Reference

Handler for an gpio pin irq.

```
#include <virtio-gpio-device>
```

Collaboration diagram for L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::Irq_handler:



16.430.1 Detailed Description

```
template<typename Request_handler, typename Epiface = L4virtio::Device>
struct L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::Irq_handler
```

Handler for an gpio pin irq.

This notifies the virtio client that an gpio pin irq happened or the operation was canceled.

This needs to be called by any server implementation of the [Virtio_gpio](#) class, after an irq was enabled with the enable method of the Gpio_request_handler.

Definition at line 166 of file [virtio-gpio-device](#).

The documentation for this struct was generated from the following file:

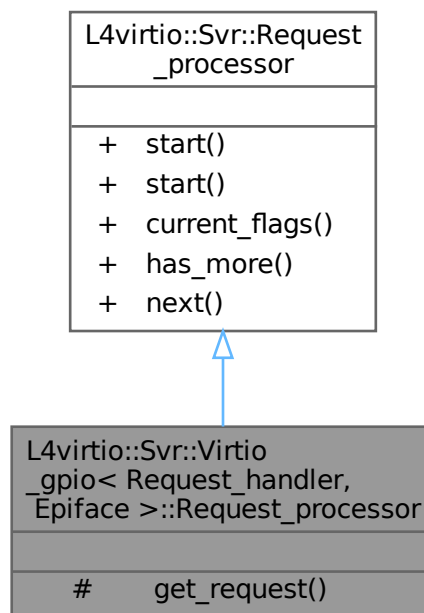
- I4/I4virtio/server/virtio-gpio-device

16.431 L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::Request_processor Struct Reference

Generic handler for the Virtio requests.

```
#include <virtio-gpio-device>
```

Inheritance diagram for L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::Request_processor:





- ```
T get_request ()
```

Generated for L4Re by Doxygen

## Additional Inherited Members

## Public Member Functions inherited from L4virtio::Svr::Request\_processor

- template<typename DESC\_MAN, typename ... ARGS>  
void [start](#) (DESC\_MAN \*dm, [Virtqueue](#) \*ring, [Virtqueue::Head\\_desc](#) const &request, ARGS... args)  
*Start processing a new request.*
- template<typename DESC\_MAN, typename ... ARGS>  
[Virtqueue::Request](#) const & [start](#) (DESC\_MAN \*dm, [Virtqueue::Request](#) const &request, ARGS... args)  
*Start processing a new request.*
- [Virtqueue::Desc::Flags](#) [current\\_flags](#) () const  
*Get the flags of the currently processed descriptor.*
- bool [has\\_more](#) () const  
*Are there more chained descriptors?*
- template<typename DESC\_MAN, typename ... ARGS>  
bool [next](#) (DESC\_MAN \*dm, ARGS... args)  
*Switch to the next descriptor in a descriptor chain.*

### 16.431.1 Detailed Description

template<typename Request\_handler, typename [Epiface](#) = L4virtio::Device>  
struct L4virtio::Svr::Virtio\_gpio< Request\_handler, [Epiface](#) >::Request\_processor

Generic handler for the Virtio requests.

Definition at line 213 of file [virtio-gpio-device](#).

### 16.431.2 Member Function Documentation

#### 16.431.2.1 get\_request()

```
template<typename Request_handler, typename Epiface = L4virtio::Device>
template<typename T>
T L4virtio::Svr::Virtio_gpio< Request_handler, Epiface >::Request_processor::get_request ()
[inline], [protected]
```

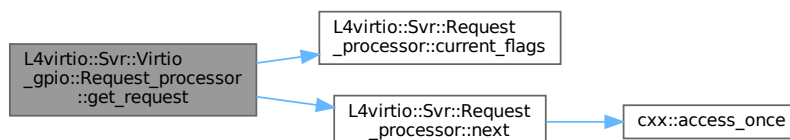
The driver prepares the GPIO request in two data parts: 1st: in\_hdr 2rd: out\_hdr.

This parses the two Data\_buffers and create the Gpio\_\* structure.

Definition at line 241 of file [virtio-gpio-device](#).

References [L4virtio::Svr::Request\\_processor::current\\_flags\(\)](#), and [L4virtio::Svr::Request\\_processor::next\(\)](#).

Here is the call graph for this function:



The documentation for this struct was generated from the following file:

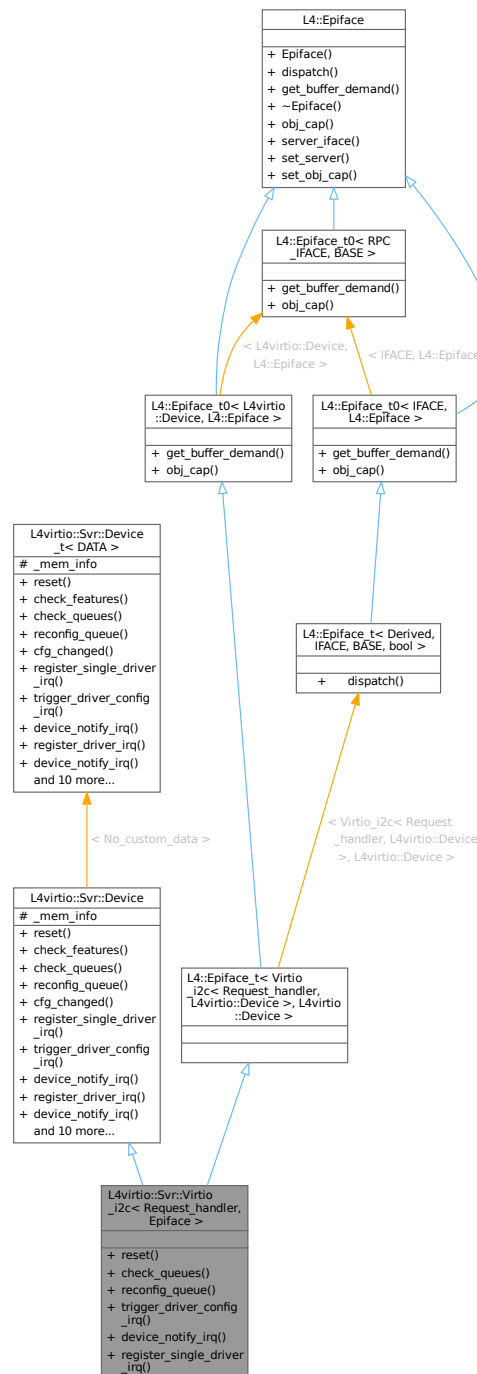
- I4/I4virtio/server/virtio-gpio-device

## 16.432 L4virtio::Svr::Virtio\_i2c< Request\_handler, Epiface > Class Template Reference

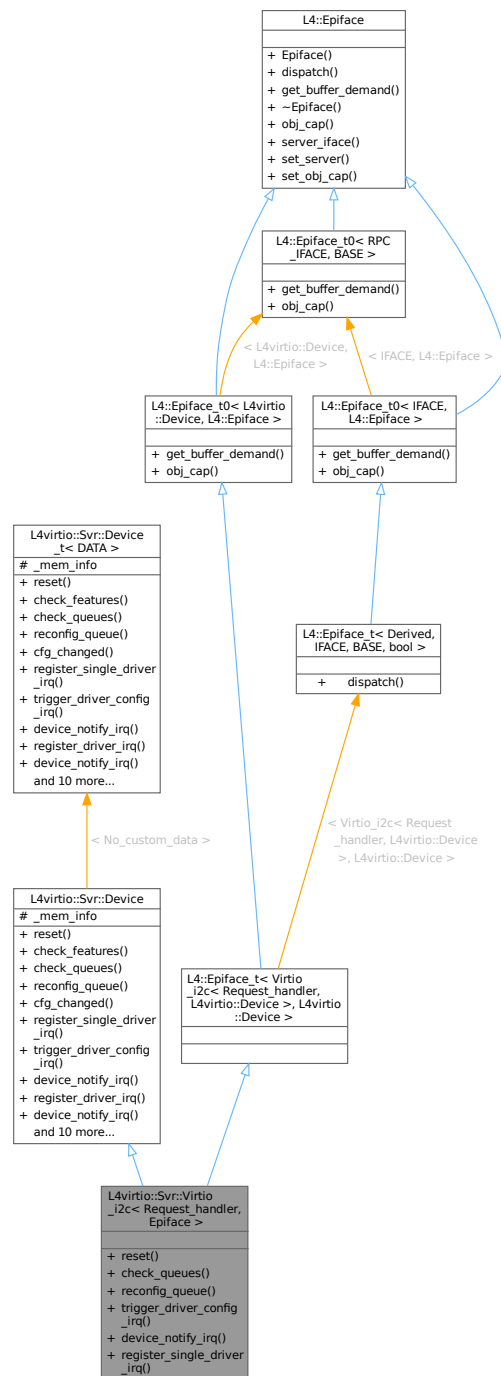
A server implementation of the virtio-i2c protocol.

```
#include <virtio-i2c-device>
```

Inheritance diagram for L4virtio::Svr::Virtio\_i2c< Request\_handler, Epiface >:



Collaboration diagram for L4virtio::Svr::Virtio\_i2c< Request\_handler, Epiface >:



## Data Structures

- class [Host\\_irq](#)  
Handler for the host irq.
- class [Request\\_processor](#)  
Handler for the Virtio requests.

## Public Member Functions

- void **reset** () override  
*reset callback, called for doing a device reset*
- bool **check\_queues** () override  
*callback for checking if the queues at DRIVER\_OK transition*
- int **reconfig\_queue** (unsigned idx) override  
*callback for client queue-config request*
- void **trigger\_driver\_config\_irq** () override  
*callback for triggering configuration change notification IRQ*
- [L4::Cap](#) < [L4::Irq](#) > **device\_notify\_irq** () const override  
*callback to gather the device notification IRQ (old-style)*
- void **register\_single\_driver\_irq** () override  
*callback for registering a single guest IRQ for all queues (old-style)*

## Public Member Functions inherited from [L4virtio::Svr::Device\\_t](#) < [No\\_custom\\_data](#) >

- virtual bool **check\_features** ()  
*callback for checking the subset of accepted features*
- virtual void **cfg\_changed** (unsigned)  
*callback for client device configuration changes*
- virtual void **register\_driver\_irq** (unsigned idx)  
*Callback for registering an notification IRQ (multi IRQ).*
- virtual [L4::Cap](#) < [L4::Irq](#) > **device\_notify\_irq** (unsigned idx)  
*Callback to gather the device notification IRQ (multi IRQ).*
- virtual unsigned **num\_events\_supported** () const  
*Return the highest notification index supported.*
- [Device\\_t](#) ([Dev\\_config](#) \*dev\_config)  
*Make a device for the given config.*
- [Mem\\_list](#) const \* **mem\_info** () const  
*Get the memory region list used for this device.*
- void **reset\_queue\_config** (unsigned idx, unsigned num\_max, bool inc\_generation=false)  
*Trigger reset for the configuration space for queue idx.*
- void **init\_mem\_info** (unsigned num)  
*Initialize the memory region list to the given maximum.*
- void **device\_error** ()  
*Transition device into DEVICE\_NEEDS\_RESET state.*
- bool **setup\_queue** ([Virtqueue](#) \*q, unsigned qn, unsigned num\_max)  
*Enable/disable the specified queue.*
- bool **handle\_mem\_cmd\_write** ()  
*Check for a value in the cmd register and handle a write.*
- void **enable\_trusted\_ds\_validation** ()  
*Enable trusted dataspace validation.*
- void **add\_trusted\_dataspaces** (std::shared\_ptr < [Ds\\_vector](#) const > ds)  
*Provide a list of trusted dataspaces that can be used for validation.*

## Public Member Functions inherited from [L4::Epiface\\_t0](#) < [L4virtio::Device](#), [L4::Epiface](#) >

- [Type\\_info::Demand](#) **get\_buffer\_demand** () const  
*Get the server-side buffer demand based in IFACE.*
- [Cap](#) < [L4virtio::Device](#) > **obj\_cap** () const  
*Get the (typed) capability to this object.*



## Public Member Functions inherited from L4::Epiface

- **Epiface** ()  
*Make a server object.*
- virtual **~Epiface** ()=0  
*Destroy the object.*
- Stored\_cap **obj\_cap** () const  
*Get the capability to the kernel object belonging to this object.*
- **Server\_iface** \* **server\_iface** () const  
*Get pointer to server interface at which the object is currently registered.*
- int **set\_server** (**Server\_iface** \*srv, **Cap**< void > cap, bool managed=false)  
*Set server registration info for the object.*
- void **set\_obj\_cap** (**Cap**< void > const &cap)  
*Deprecated server registration function.*

## Additional Inherited Members

## Public Types inherited from L4::Epiface\_t0< L4virtio::Device, L4::Epiface >

- typedef **L4virtio::Device** **Interface**  
*Data type of the IPC interface definition.*

## Public Types inherited from L4::Epiface

- typedef **lpc\_svr::Server\_iface** **Server\_iface**  
*Type for abstract server interface.*
- typedef **lpc\_svr::Server\_iface::Demand** **Demand**  
*Type for server-side receive buffer demand.*

## Protected Attributes inherited from L4virtio::Svr::Device\_t< No\_custom\_data >

- **Mem\_list** **\_mem\_info**  
*Memory region list.*

## 16.432.1 Detailed Description

```
template<typename Request_handler, typename Epiface = L4virtio::Device>
class L4virtio::Svr::Virtio_i2c< Request_handler, Epiface >
```

A server implementation of the virtio-i2c protocol.

### Template Parameters

|                        |                                                                                                                                                                                          |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Request_handler</i> | The type that is used to handle incoming requests. Needs to have <code>handle_read(l4_uint8_t *, unsigned)</code> and <code>handle_write(l4_uint8_t const *, unsigned)</code> functions. |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                |                                                                            |
|----------------|----------------------------------------------------------------------------|
| <i>Epiface</i> | The Epiface to derive from. Defaults to <a href="#">L4virtio::Device</a> . |
|----------------|----------------------------------------------------------------------------|

Definition at line 86 of file [virtio-i2c-device](#).

The documentation for this class was generated from the following file:

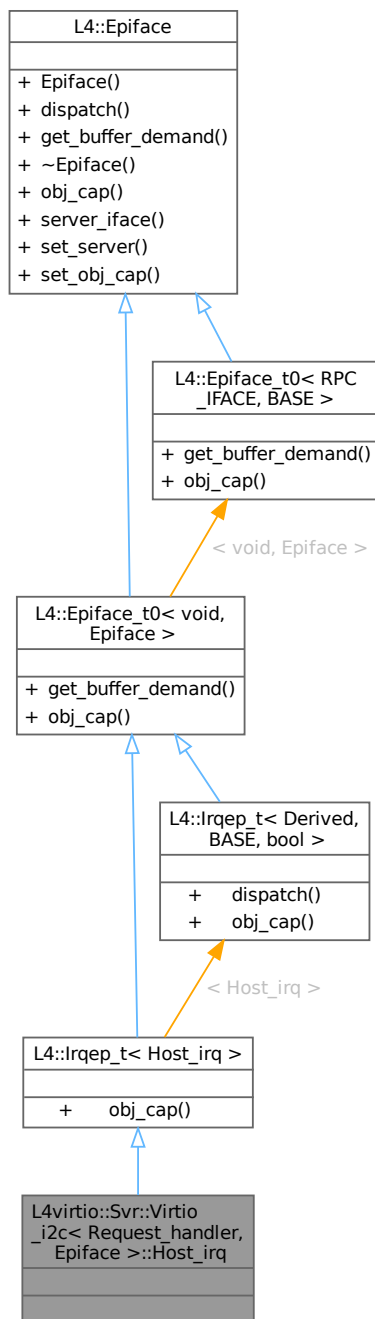
- l4/l4virtio/server/virtio-i2c-device

### 16.433 L4virtio::Svr::Virtio\_i2c< Request\_handler, Epiface >::Host\_irq Class Reference

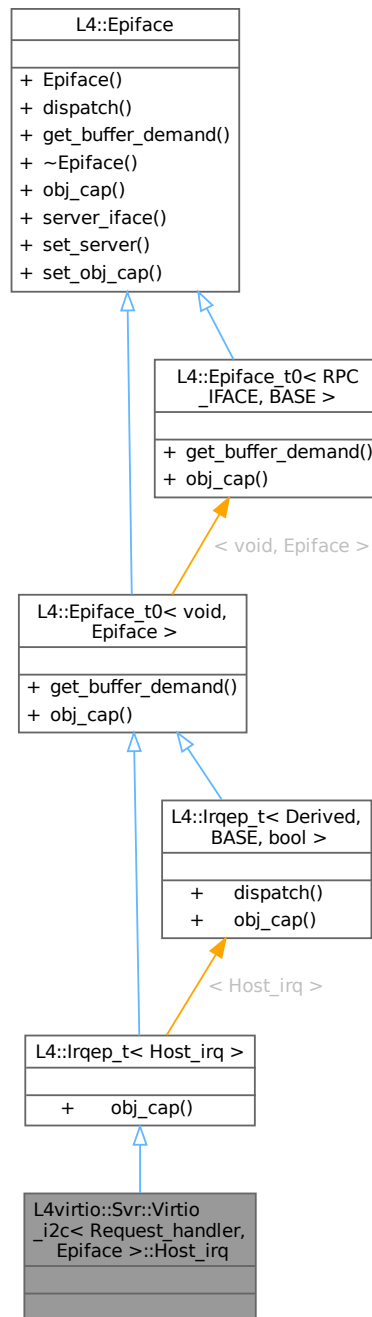
Handler for the host irq.

```
#include <virtio-i2c-device>
```

Inheritance diagram for L4virtio::Svr::Virtio\_i2c< Request\_handler, Epiface >::Host\_irq:



Collaboration diagram for L4virtio::Svr::Virtio\_i2c< Request\_handler, Epiface >::Host\_irq:



### Additional Inherited Members

### Public Types inherited from [L4::Epiface\\_t0< void, Epiface >](#)

- typedef void **Interface**

*Data type of the IPC interface definition.*

## Public Types inherited from L4::Epiface

- typedef [lpc\\_svr::Server\\_iface](#) **Server\_iface**  
*Type for abstract server interface.*
- typedef [lpc\\_svr::Server\\_iface::Demand](#) **Demand**  
*Type for server-side receive buffer demand.*

## Public Member Functions inherited from L4::lrqep\_t< Host\_irq >

- [Cap< L4::lrq >](#) **obj\_cap** () const  
*Get the (typed) capability to this object.*

## Public Member Functions inherited from L4::Epiface\_t0< void, Epiface >

- [Type\\_info::Demand](#) **get\_buffer\_demand** () const  
*Get the server-side buffer demand based in IFACE.*
- [Cap< void >](#) **obj\_cap** () const  
*Get the (typed) capability to this object.*

## Public Member Functions inherited from L4::Epiface

- **Epiface** ()  
*Make a server object.*
- virtual **~Epiface** ()=0  
*Destroy the object.*
- Stored\_cap [obj\\_cap](#) () const  
*Get the capability to the kernel object belonging to this object.*
- [Server\\_iface](#) \* [server\\_iface](#) () const  
*Get pointer to server interface at which the object is currently registered.*
- int **set\_server** ([Server\\_iface](#) \*srv, [Cap< void >](#) cap, bool managed=false)  
*Set server registration info for the object.*
- void **set\_obj\_cap** ([Cap< void >](#) const &cap)  
*Deprecated server registration function.*

### 16.433.1 Detailed Description

```
template<typename Request_handler, typename Epiface = L4virtio::Device>
class L4virtio::Svr::Virtio_i2c< Request_handler, Epiface >::Host_irq
```

Handler for the host irq.

An [L4::lrqep\\_t](#) to handle irq's send to the server.

Definition at line 106 of file [virtio-i2c-device](#).

The documentation for this class was generated from the following file:

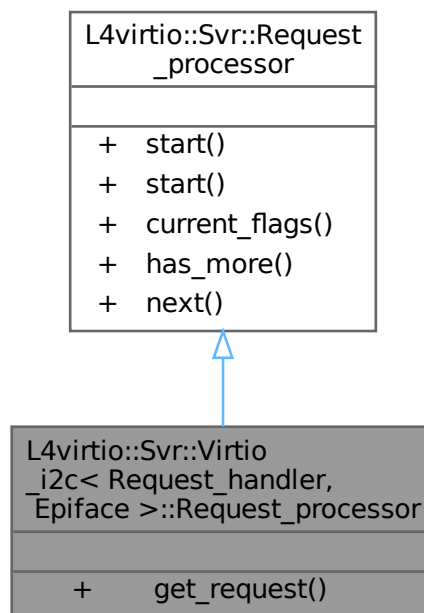
- [l4/l4virtio/server/virtio-i2c-device](#)

## 16.434 L4virtio::Svr::Virtio\_i2c< Request\_handler, Epiface >::Request\_processor Class Reference

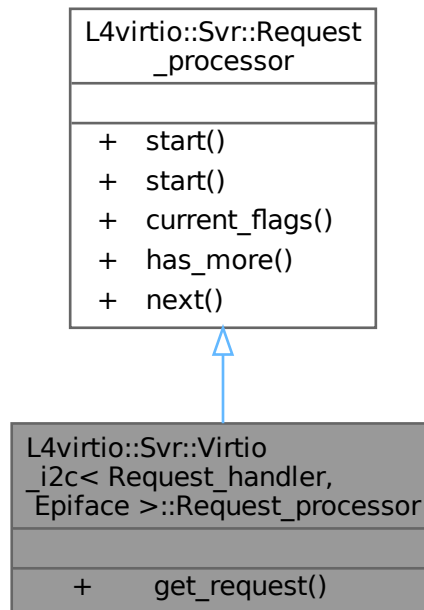
Handler for the Virtio requests.

```
#include <virtio-i2c-device>
```

Inheritance diagram for L4virtio::Svr::Virtio\_i2c< Request\_handler, Epiface >::Request\_processor:



Collaboration diagram for L4virtio::Svr::Virtio\_i2c< Request\_handler, Epiface >::Request\_processor:



## Public Member Functions

- I2c\_req [get\\_request](#) ()

*Linux prepares the I2C request in three data parts: 1st: out\_hdr 2nd: buffer (optional) 3rd: in\_hdr.*

## Public Member Functions inherited from [L4virtio::Svr::Request\\_processor](#)

- template<typename DESC\_MAN, typename ... ARGS>  
void [start](#) (DESC\_MAN \*dm, [Virtqueue](#) \*ring, [Virtqueue::Head\\_desc](#) const &request, ARGS... args)  
*Start processing a new request.*
- template<typename DESC\_MAN, typename ... ARGS>  
[Virtqueue::Request](#) const & [start](#) (DESC\_MAN \*dm, [Virtqueue::Request](#) const &request, ARGS... args)  
*Start processing a new request.*
- [Virtqueue::Desc::Flags](#) [current\\_flags](#) () const  
*Get the flags of the currently processed descriptor.*
- bool [has\\_more](#) () const  
*Are there more chained descriptors?*
- template<typename DESC\_MAN, typename ... ARGS>  
bool [next](#) (DESC\_MAN \*dm, ARGS... args)  
*Switch to the next descriptor in a descriptor chain.*

### 16.434.1 Detailed Description

```
template<typename Request_handler, typename Epiface = L4virtio::Device>
class L4virtio::Svr::Virtio_i2c< Request_handler, Epiface >::Request_processor
```

Handler for the Virtio requests.

Definition at line 123 of file [virtio-i2c-device](#).

### 16.434.2 Member Function Documentation

#### 16.434.2.1 get\_request()

```
template<typename Request_handler, typename Epiface = L4virtio::Device>
I2c_req L4virtio::Svr::Virtio_i2c< Request_handler, Epiface >::Request_processor::get_request
() [inline]
```

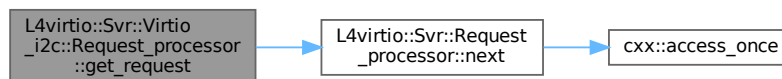
Linux prepares the I2C request in three data parts: 1st: out\_hdr 2nd: buffer (optional) 3rd: in\_hdr.

This parses the three Data\_buffers and recreate the virtio\_i2c\_req structure.

Definition at line 172 of file [virtio-i2c-device](#).

References [L4virtio::Svr::Data\\_buffer::left](#), [L4virtio::Svr::Request\\_processor::next\(\)](#), and [L4virtio::Svr::Data\\_buffer::pos](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

- [l4/l4virtio/server/virtio-i2c-device](#)

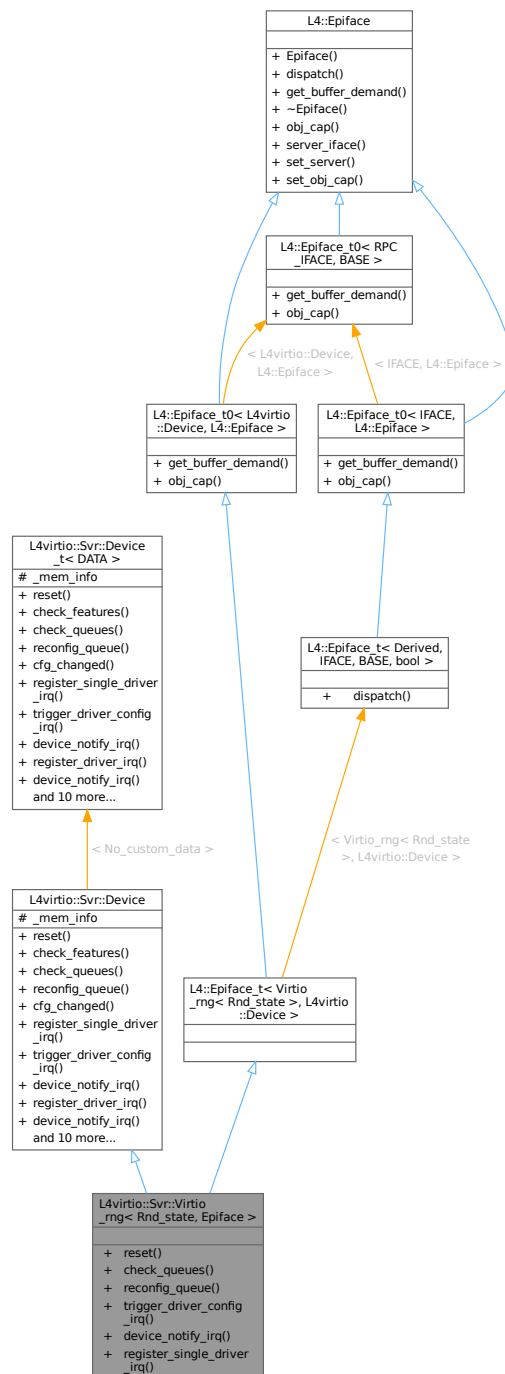


## 16.435 L4virtio::Svr::Virtio\_rng< Rnd\_state, Epiface > Class Template Reference

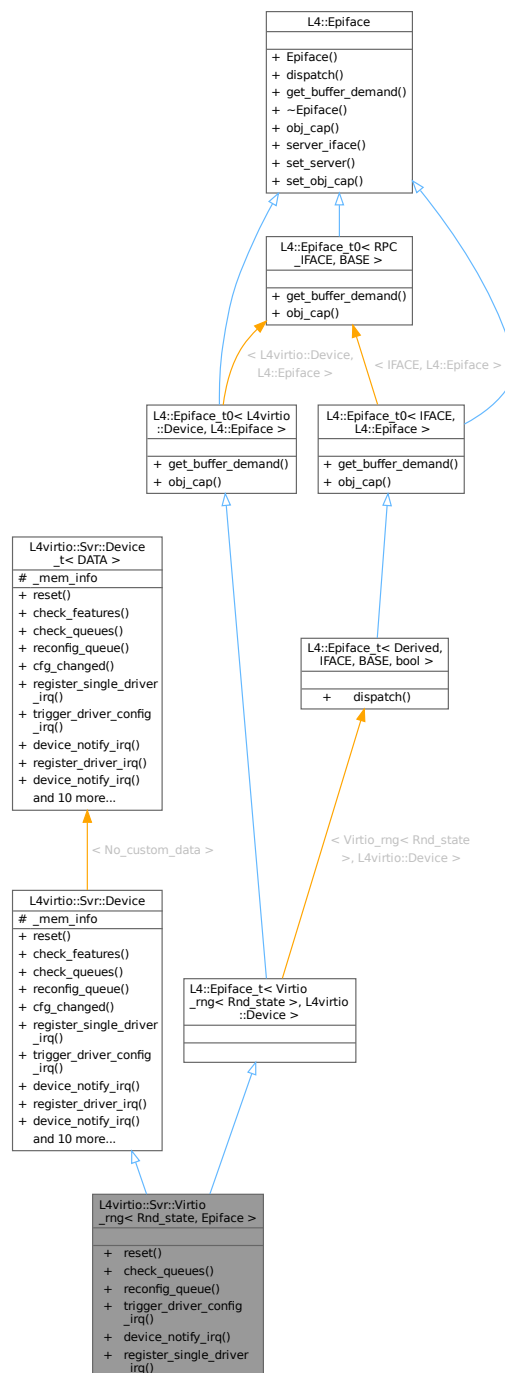
A server implementation of the virtio-rng protocol.

```
#include <virtio-rng-device>
```

Inheritance diagram for L4virtio::Svr::Virtio\_rng< Rnd\_state, Epiface >:



Collaboration diagram for L4virtio::Svr::Virtio\_rng< Rnd\_state, Epiface >:



## Data Structures

- class [Host\\_irq](#)  
Handler for the host irq.
- class [Request\\_processor](#)  
Handler for the Virtio requests.

**Public Member Functions**

- void **reset** () override  
*reset callback, called for doing a device reset*
- bool **check\_queues** () override  
*callback for checking if the queues at DRIVER\_OK transition*
- int **reconfig\_queue** (unsigned idx) override  
*callback for client queue-config request*
- void **trigger\_driver\_config\_irq** () override  
*callback for triggering configuration change notification IRQ*
- L4::Cap< L4::lrq > **device\_notify\_irq** () const override  
*callback to gather the device notification IRQ (old-style)*
- void **register\_single\_driver\_irq** () override  
*callback for registering a single guest IRQ for all queues (old-style)*

**Public Member Functions inherited from L4virtio::Svr::Device\_t< No\_custom\_data >**

- virtual bool **check\_features** ()  
*callback for checking the subset of accepted features*
- virtual void **cfg\_changed** (unsigned)  
*callback for client device configuration changes*
- virtual void **register\_driver\_irq** (unsigned idx)  
*Callback for registering an notification IRQ (multi IRQ).*
- virtual L4::Cap< L4::lrq > **device\_notify\_irq** (unsigned idx)  
*Callback to gather the device notification IRQ (multi IRQ).*
- virtual unsigned **num\_events\_supported** () const  
*Return the highest notification index supported.*
- **Device\_t** (**Dev\_config** \*dev\_config)  
*Make a device for the given config.*
- **Mem\_list** const \* **mem\_info** () const  
*Get the memory region list used for this device.*
- void **reset\_queue\_config** (unsigned idx, unsigned num\_max, bool inc\_generation=false)  
*Trigger reset for the configuration space for queue idx.*
- void **init\_mem\_info** (unsigned num)  
*Initialize the memory region list to the given maximum.*
- void **device\_error** ()  
*Transition device into DEVICE\_NEEDS\_RESET state.*
- bool **setup\_queue** (**Virtqueue** \*q, unsigned qn, unsigned num\_max)  
*Enable/disable the specified queue.*
- bool **handle\_mem\_cmd\_write** ()  
*Check for a value in the cmd register and handle a write.*
- void **enable\_trusted\_ds\_validation** ()  
*Enable trusted dataspace validation.*
- void **add\_trusted\_dataspaces** (std::shared\_ptr< Ds\_vector const > ds)  
*Provide a list of trusted dataspaces that can be used for validation.*

**Public Member Functions inherited from L4::Epiface\_t0< L4virtio::Device, L4::Epiface >**

- **Type\_info::Demand** **get\_buffer\_demand** () const  
*Get the server-side buffer demand based in IFACE.*
- **Cap**< L4virtio::Device > **obj\_cap** () const  
*Get the (typed) capability to this object.*

## Public Member Functions inherited from [L4::Epiface](#)

- **Epiface** ()  
*Make a server object.*
- virtual **~Epiface** ()=0  
*Destroy the object.*
- Stored\_cap **obj\_cap** () const  
*Get the capability to the kernel object belonging to this object.*
- [Server\\_iface](#) \* **server\_iface** () const  
*Get pointer to server interface at which the object is currently registered.*
- int **set\_server** ([Server\\_iface](#) \*srv, [Cap](#)< void > cap, bool managed=false)  
*Set server registration info for the object.*
- void **set\_obj\_cap** ([Cap](#)< void > const &cap)  
*Deprecated server registration function.*

## Additional Inherited Members

## Public Types inherited from [L4::Epiface\\_t0](#)< [L4virtio::Device](#), [L4::Epiface](#) >

- typedef [L4virtio::Device](#) **Interface**  
*Data type of the IPC interface definition.*

## Public Types inherited from [L4::Epiface](#)

- typedef [lpc\\_svr::Server\\_iface](#) **Server\_iface**  
*Type for abstract server interface.*
- typedef [lpc\\_svr::Server\\_iface::Demand](#) **Demand**  
*Type for server-side receive buffer demand.*

## Protected Attributes inherited from [L4virtio::Svr::Device\\_t](#)< [No\\_custom\\_data](#) >

- [Mem\\_list](#) **\_mem\_info**  
*Memory region list.*

## 16.435.1 Detailed Description

```
template<typename Rnd_state, typename Epiface = L4virtio::Device>
class L4virtio::Svr::Virtio_rng< Rnd_state, Epiface >
```

A server implementation of the virtio-rng protocol.

## Template Parameters

---

|                  |                                                                                                                                                                                                                                                                                                                       |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Rnd_state</i> | The type that implements the random data generation. <code>Rnd_state::get_random(int len, unsigned char *buf)</code> is called to get len random bytes written into buf TODO: virtio-rng supports providing less random bytes then requested. This API currently does not support that, as I do not have a test case. |
| <i>Epiface</i>   | The Epiface to derive from. Defaults to <code>L4virtio::Device</code> .                                                                                                                                                                                                                                               |

Definition at line 33 of file [virtio-rng-device](#).

The documentation for this class was generated from the following file:

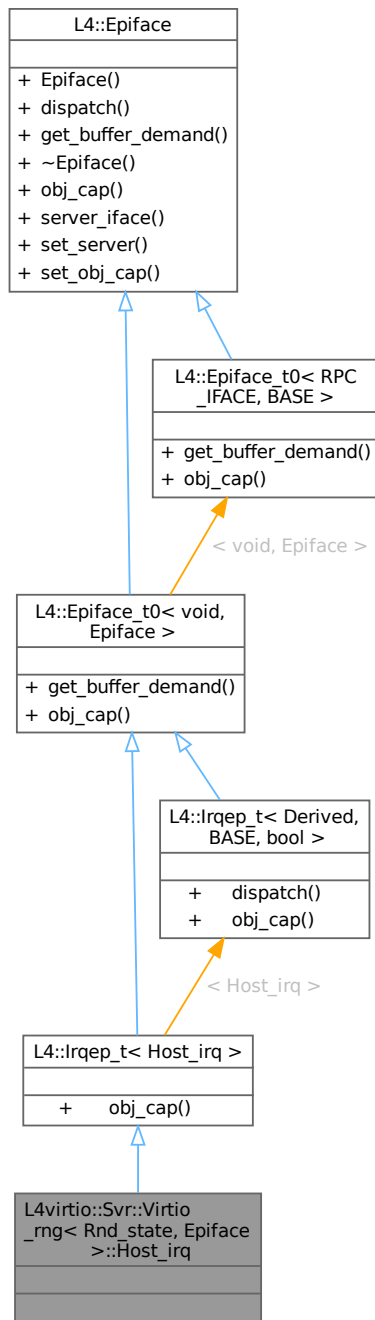
- I4/I4virtio/server/virtio-rng-device

## 16.436 L4virtio::Svr::Virtio\_rng< Rnd\_state, Epiface >::Host\_irq Class Reference

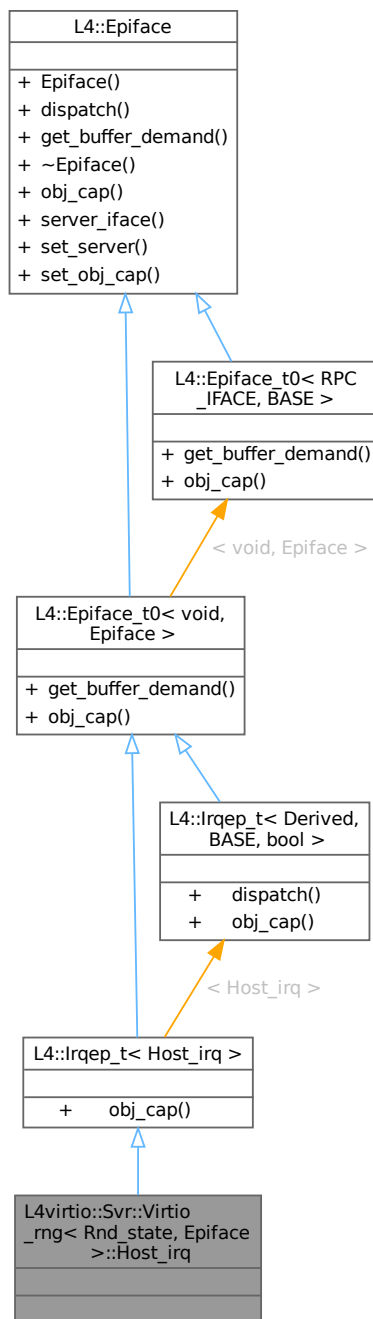
Handler for the host irq.

```
#include <virtio-rng-device>
```

Inheritance diagram for L4virtio::Svr::Virtio\_rng< Rnd\_state, Epiface >::Host\_irq:



Collaboration diagram for L4virtio::Svr::Virtio\_rng< Rnd\_state, Epiface >::Host\_irq:



#### Additional Inherited Members

#### Public Types inherited from `L4::Epiface_t0< void, Epiface >`

- typedef void **Interface**

*Data type of the IPC interface definition.*

## Public Types inherited from [L4::Epiface](#)

- typedef [lpc\\_svr::Server\\_iface](#) **Server\_iface**  
*Type for abstract server interface.*
- typedef [lpc\\_svr::Server\\_iface::Demand](#) **Demand**  
*Type for server-side receive buffer demand.*

## Public Member Functions inherited from [L4::lrqep\\_t< Host\\_irq >](#)

- [Cap< L4::lrq > obj\\_cap](#) () const  
*Get the (typed) capability to this object.*

## Public Member Functions inherited from [L4::Epiface\\_t0< void, Epiface >](#)

- [Type\\_info::Demand](#) **get\_buffer\_demand** () const  
*Get the server-side buffer demand based in IFACE.*
- [Cap< void > obj\\_cap](#) () const  
*Get the (typed) capability to this object.*

## Public Member Functions inherited from [L4::Epiface](#)

- **Epiface** ()  
*Make a server object.*
- virtual **~Epiface** ()=0  
*Destroy the object.*
- Stored\_cap [obj\\_cap](#) () const  
*Get the capability to the kernel object belonging to this object.*
- [Server\\_iface](#) \* [server\\_iface](#) () const  
*Get pointer to server interface at which the object is currently registered.*
- int **set\_server** ([Server\\_iface](#) \*srv, [Cap< void > cap](#), bool managed=false)  
*Set server registration info for the object.*
- void **set\_obj\_cap** ([Cap< void > const &cap](#))  
*Deprecated server registration function.*

### 16.436.1 Detailed Description

```
template<typename Rnd_state, typename Epiface = L4virtio::Device>
class L4virtio::Svr::Virtio_rng< Rnd_state, Epiface >::Host_irq
```

Handler for the host irq.

An [L4::lrqep\\_t](#) to handle irq's send to the server.

Definition at line 51 of file [virtio-rng-device](#).

The documentation for this class was generated from the following file:

- I4/I4virtio/server/virtio-rng-device

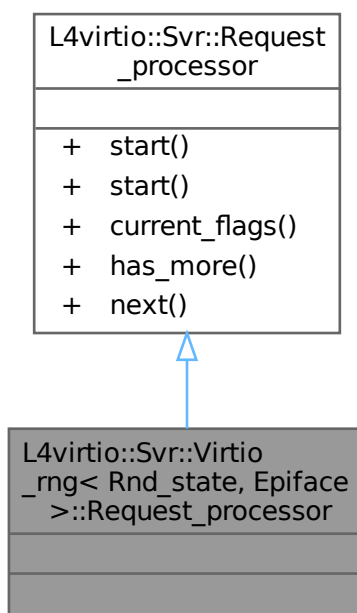


## 16.437 L4virtio::Svr::Virtio\_rng< Rnd\_state, Epiface >::Request\_processor Class Reference

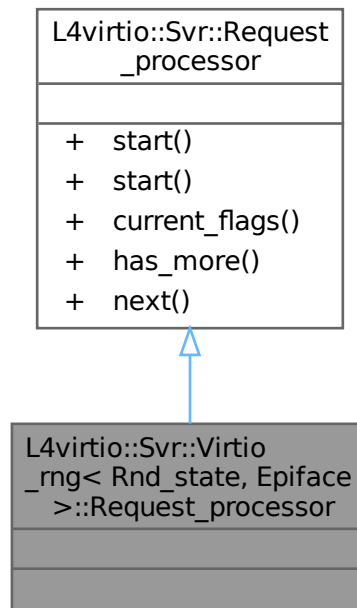
Handler for the Virtio requests.

```
#include <virtio-rng-device>
```

Inheritance diagram for L4virtio::Svr::Virtio\_rng< Rnd\_state, Epiface >::Request\_processor:



Collaboration diagram for L4virtio::Svr::Virtio\_rng< Rnd\_state, Epiface >::Request\_processor:



### Additional Inherited Members

### Public Member Functions inherited from L4virtio::Svr::Request\_processor

- template<typename DESC\_MAN, typename ... ARGS>  
void [start](#) (DESC\_MAN \*dm, [Virtqueue](#) \*ring, [Virtqueue::Head\\_desc](#) const &request, ARGS... args)  
*Start processing a new request.*
- template<typename DESC\_MAN, typename ... ARGS>  
[Virtqueue::Request](#) const & [start](#) (DESC\_MAN \*dm, [Virtqueue::Request](#) const &request, ARGS... args)  
*Start processing a new request.*
- [Virtqueue::Desc::Flags](#) [current\\_flags](#) () const  
*Get the flags of the currently processed descriptor.*
- bool [has\\_more](#) () const  
*Are there more chained descriptors?*
- template<typename DESC\_MAN, typename ... ARGS>  
bool [next](#) (DESC\_MAN \*dm, ARGS... args)  
*Switch to the next descriptor in a descriptor chain.*

### 16.437.1 Detailed Description

```
template<typename Rnd_state, typename Epiface = L4virtio::Device>
class L4virtio::Svr::Virtio_rng< Rnd_state, Epiface >::Request_processor
```

Handler for the Virtio requests.

Definition at line 68 of file [virtio-rng-device](#).

The documentation for this class was generated from the following file:

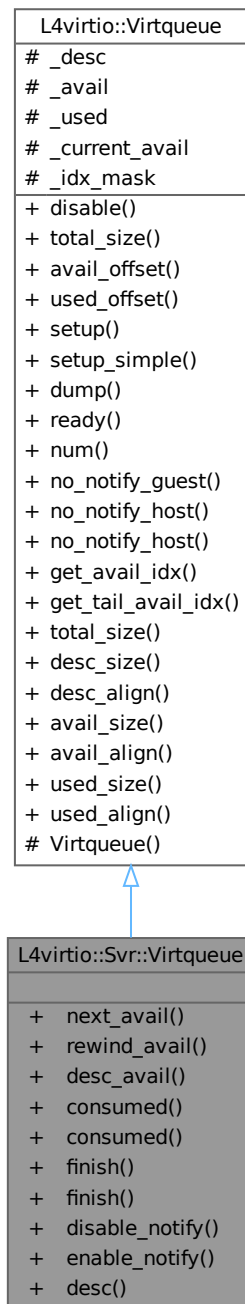
- l4/l4virtio/server/virtio-rng-device

## 16.438 L4virtio::Svr::Virtqueue Class Reference

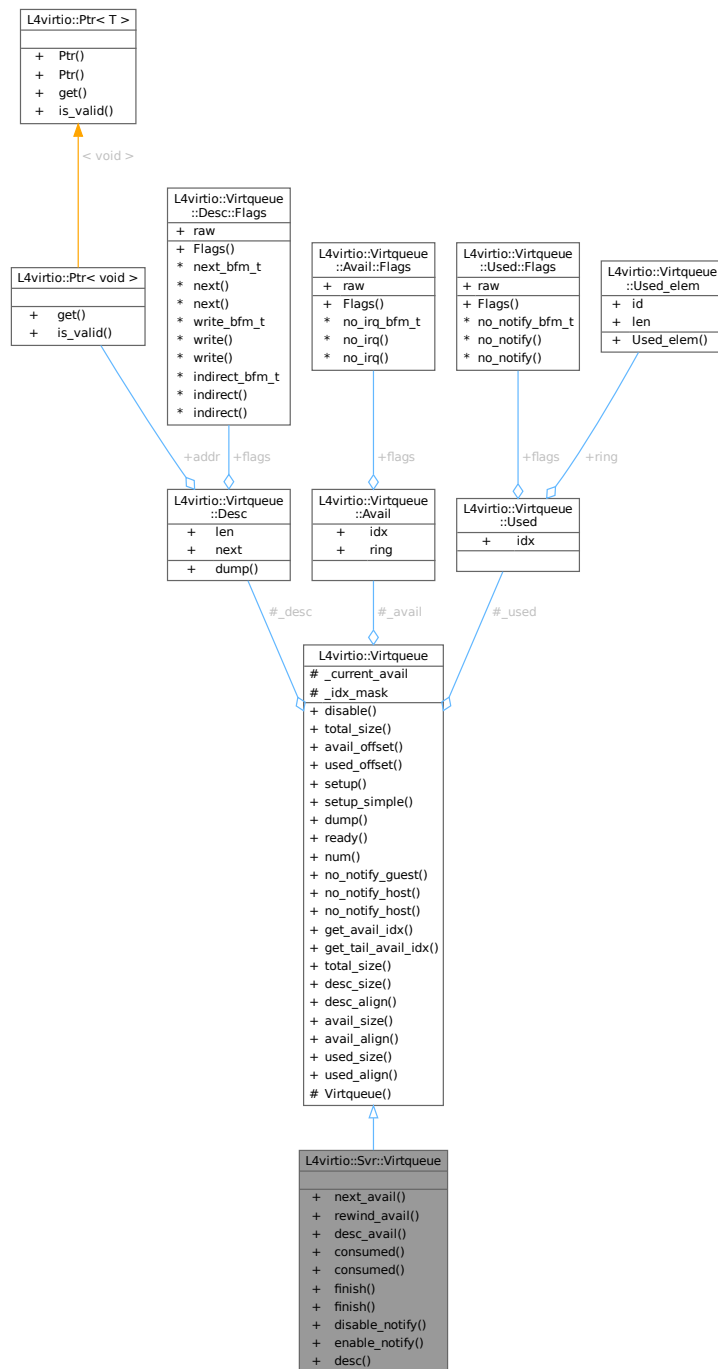
[Virtqueue](#) implementation for the device.

```
#include <virtio>
```

Inheritance diagram for L4virtio::Svr::Virtqueue:



Collaboration diagram for L4virtio::Svr::Virtqueue:



## Data Structures

- class [Head\\_desc](#)  
*VIRTIO request, essentially a descriptor from the available ring.*

## Public Member Functions

- Request [next\\_avail](#) ()

- Get the next available descriptor from the available ring.*

  - void `rewind_avail` (`Head_desc` const &d)

*Return unfinished descriptors to the available ring, i.e.*

- bool `desc_avail` () const

*Test for available descriptors.*

- void `consumed` (`Head_desc` const &r, `l4_uint32_t` len=0)

*Put the given descriptor into the used ring.*

- template<typename ITER>  
void `consumed` (ITER const &begin, ITER const &end)

*Put multiple descriptors into the used ring.*

- template<typename QUEUE\_OBSERVER>  
void `finish` (`Head_desc` &d, QUEUE\_OBSERVER \*o, `l4_uint32_t` len=0)

*Add a descriptor to the used ring, and notify an observer.*

- template<typename ITER, typename QUEUE\_OBSERVER>  
void `finish` (ITER const &begin, ITER const &end, QUEUE\_OBSERVER \*o)

*Add a range of descriptors to the used ring, and notify an observer once.*

- void `disable_notify` ()

*Set the 'no notify' flag for this queue.*

- void `enable_notify` ()

*Clear the 'no notify' flag for this queue.*

- `Desc` const \* `desc` (unsigned idx) const

*Get a descriptor from the descriptor list.*

## Public Member Functions inherited from `L4virtio::Virtqueue`

- void `disable` ()

*Completely disable the queue.*

- unsigned long `total_size` () const

*Calculate the total size of this virtqueue.*

- unsigned long `avail_offset` () const

*Get the offset of the available ring from the descriptor table.*

- unsigned long `used_offset` () const

*Get the offset of the used ring from the descriptor table.*

- void `setup` (unsigned `num`, void \*desc, void \*avail, void \*used)

*Enable this queue.*

- void `setup_simple` (unsigned `num`, void \*ring)

*Enable this queue.*

- void `dump` (`Desc` const \*d) const

*Dump descriptors for this queue.*

- bool `ready` () const

*Test if this queue is in working state.*

- unsigned `num` () const
- bool `no_notify_guest` () const

*Get the no IRQ flag of this queue.*

- bool `no_notify_host` () const

*Get the no notify flag of this queue.*

- void `no_notify_host` (bool value)

*Set the no-notify flag for this queue.*

- `l4_uint16_t` `get_avail_idx` () const

*Get available index from available ring (for debugging).*

- `l4_uint16_t` `get_tail_avail_idx` () const

*Get tail-available index stored in local state (for debugging).*

## Additional Inherited Members

### Public Types inherited from [L4virtio::Virtqueue](#)

- enum  
*Fixed alignment values for different parts of a virtqueue.*

### Static Public Member Functions inherited from [L4virtio::Virtqueue](#)

- static unsigned long [total\\_size](#) (unsigned [num](#))  
*Calculate the total size for a virtqueue of the given dimensions.*
- static unsigned long [desc\\_size](#) (unsigned [num](#))  
*Calculate the size of the descriptor table for [num](#) entries.*
- static unsigned long [desc\\_align](#) ()  
*Get the alignment in zero LSBs needed for the descriptor table.*
- static unsigned long [avail\\_size](#) (unsigned [num](#))  
*Calculate the size of the available ring for [num](#) entries.*
- static unsigned long [avail\\_align](#) ()  
*Get the alignment in zero LSBs needed for the available ring.*
- static unsigned long [used\\_size](#) (unsigned [num](#))  
*Calculate the size of the used ring for [num](#) entries.*
- static unsigned long [used\\_align](#) ()  
*Get the alignment in zero LSBs needed for the used ring.*

### Protected Member Functions inherited from [L4virtio::Virtqueue](#)

- [Virtqueue](#) ()=default  
*Create a disabled virtqueue.*

### Protected Attributes inherited from [L4virtio::Virtqueue](#)

- [Desc](#) \* [\\_desc](#) = nullptr  
*pointer to descriptor table, NULL if queue is off.*
- [Avail](#) \* [\\_avail](#) = nullptr  
*pointer to available ring.*
- [Used](#) \* [\\_used](#) = nullptr  
*pointer to used ring.*
- [l4\\_uint16\\_t](#) [\\_current\\_avail](#) = 0  
*The life counter for the queue.*
- [l4\\_uint16\\_t](#) [\\_idx\\_mask](#) = 0  
*mask used for indexing into the descriptor table and the rings.*

## 16.438.1 Detailed Description

[Virtqueue](#) implementation for the device.

This class represents a single virtqueue, with a local running available index.

#### Note

The [Virtqueue](#) implementation is not thread-safe.

Definition at line 87 of file [virtio](#).

## 16.438.2 Member Function Documentation

### 16.438.2.1 consumed() [1/2]

```
void L4virtio::Svr::Virtqueue::consumed (
 Head_desc const & r,
 l4_uint32_t len = 0) [inline]
```

Put the given descriptor into the used ring.

#### Parameters

|            |                                           |
|------------|-------------------------------------------|
| <i>r</i>   | Request that shall be marked as finished. |
| <i>len</i> | The total number of bytes written.        |

#### Precondition

queue must be in working state.

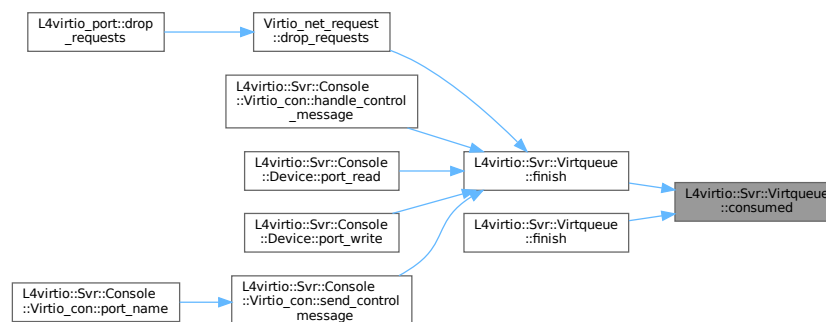
*r* must be a valid request from this queue.

Definition at line 190 of file [virtio](#).

References [L4virtio::Virtqueue::\\_desc](#), [L4virtio::Virtqueue::\\_idx\\_mask](#), and [L4virtio::Virtqueue::\\_used](#).

Referenced by [finish\(\)](#), and [finish\(\)](#).

Here is the caller graph for this function:



### 16.438.2.2 consumed() [2/2]

```
template<typename ITER>
void L4virtio::Svr::Virtqueue::consumed (
 ITER const & begin,
 ITER const & end) [inline]
```

Put multiple descriptors into the used ring.

A range of descriptors, specified by *begin* and *end* iterators is added. Each iterator points to a struct that has a first member that is a [Head\\_desc](#) and a second member that is the corresponding number of bytes written.

#### Template Parameters



|             |                                      |
|-------------|--------------------------------------|
| <i>ITER</i> | The type of the iterator (inferred). |
|-------------|--------------------------------------|

### Parameters

|              |                                            |
|--------------|--------------------------------------------|
| <i>begin</i> | Iterator pointing to first new descriptor. |
| <i>end</i>   | Iterator pointing to one past last entry.  |

### Precondition

queue must be in working state.

Definition at line 213 of file [virtio](#).

References [L4virtio::Virtqueue::\\_desc](#), [L4virtio::Virtqueue::\\_idx\\_mask](#), and [L4virtio::Virtqueue::\\_used](#).

### 16.438.2.3 desc()

```
Desc const * L4virtio::Svr::Virtqueue::desc (
 unsigned idx) const [inline]
```

Get a descriptor from the descriptor list.

### Parameters

|            |                              |
|------------|------------------------------|
| <i>idx</i> | The index of the descriptor. |
|------------|------------------------------|

### Precondition

`idx < num`

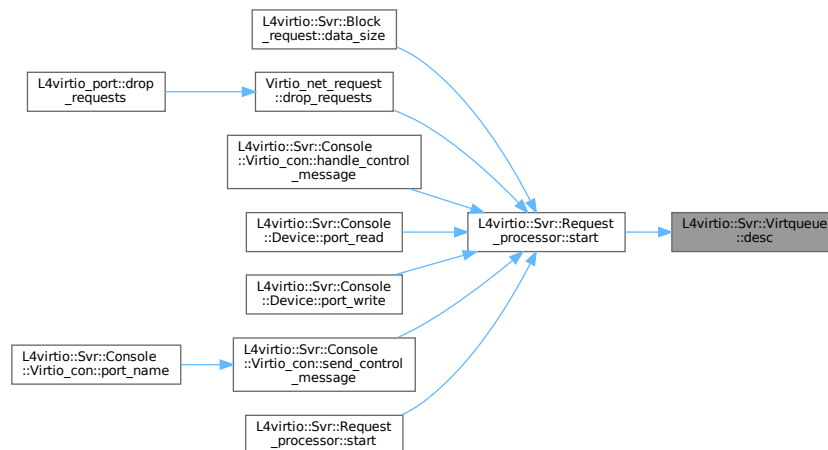
queue must be in working state

Definition at line 298 of file [virtio](#).

References [L4virtio::Virtqueue::\\_desc](#).

Referenced by [L4virtio::Svr::Request\\_processor::start\(\)](#).

Here is the caller graph for this function:



#### 16.438.2.4 desc\_avail()

```
bool L4virtio::Svr::Virtqueue::desc_avail () const [inline]
```

Test for available descriptors.

##### Returns

true if there are descriptors available, false if not.

##### Precondition

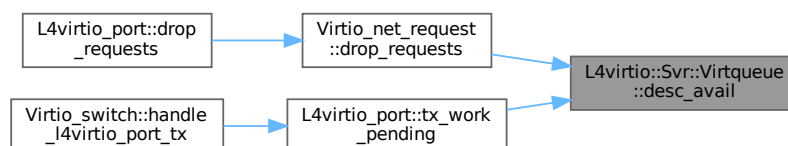
The queue must be in working state.

Definition at line 175 of file [virtio](#).

References [L4virtio::Virtqueue::\\_avail](#), and [L4virtio::Virtqueue::\\_current\\_avail](#).

Referenced by [Virtio\\_net\\_request::drop\\_requests\(\)](#), and [L4virtio\\_port::tx\\_work\\_pending\(\)](#).

Here is the caller graph for this function:



### 16.438.2.5 disable\_notify()

```
void L4virtio::Svr::Virtqueue::disable_notify () [inline]
```

Set the 'no notify' flag for this queue.

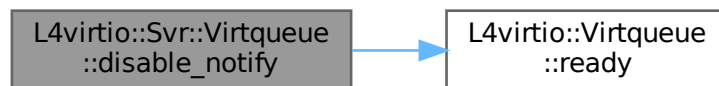
This function may be called on a disabled queue.

Definition at line 273 of file [virtio](#).

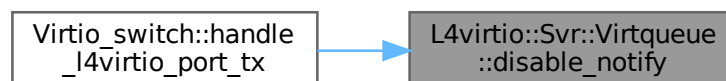
References [L4virtio::Virtqueue::\\_used](#), [L4\\_LIKELY](#), and [L4virtio::Virtqueue::ready\(\)](#).

Referenced by [Virtio\\_switch::handle\\_l4virtio\\_port\\_tx\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 16.438.2.6 enable\_notify()

```
void L4virtio::Svr::Virtqueue::enable_notify () [inline]
```

Clear the 'no notify' flag for this queue.

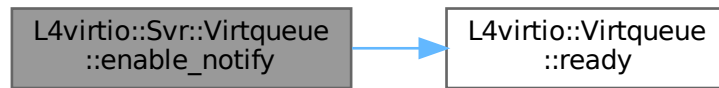
This function may be called on a disabled queue.

Definition at line 284 of file [virtio](#).

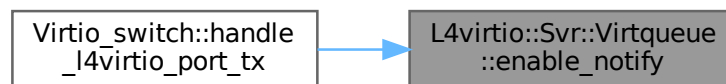
References [L4virtio::Virtqueue::\\_used](#), [L4\\_LIKELY](#), and [L4virtio::Virtqueue::ready\(\)](#).

Referenced by [Virtio\\_switch::handle\\_l4virtio\\_port\\_tx\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 16.438.2.7 finish() [1/2]

```

template<typename QUEUE_OBSERVER>
void L4virtio::Svr::Virtqueue::finish (
 Head_desc & d,
 QUEUE_OBSERVER * o,
 l4_uint32_t len = 0) [inline]

```

Add a descriptor to the used ring, and notify an observer.

#### Template Parameters

|                             |                                      |
|-----------------------------|--------------------------------------|
| <code>QUEUE_OBSERVER</code> | The type of the observer (inferred). |
|-----------------------------|--------------------------------------|

#### Parameters

|                  |                                                             |
|------------------|-------------------------------------------------------------|
| <code>d</code>   | descriptor of the request that is to be marked as finished. |
| <code>o</code>   | Pointer to the observer that is notified.                   |
| <code>len</code> | Number of bytes written for this request.                   |

**Precondition**

- queue must be in working state.
- d must be a valid request from this queue.

Definition at line 240 of file [virtio](#).

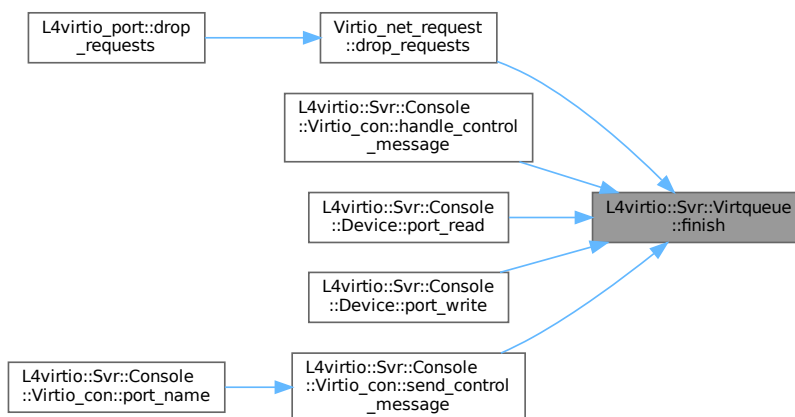
References [consumed\(\)](#).

Referenced by [Virtio\\_net\\_request::drop\\_requests\(\)](#), [L4virtio::Svr::Console::Virtio\\_con::handle\\_control\\_message\(\)](#), [L4virtio::Svr::Console::Device::port\\_read\(\)](#), [L4virtio::Svr::Console::Device::port\\_write\(\)](#), and [L4virtio::Svr::Console::Virtio\\_con::send\\_control\\_message\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**16.438.2.8 finish() [2/2]**

```

template<typename ITER, typename QUEUE_OBSERVER>
void L4virtio::Svr::Virtqueue::finish (
 ITER const & begin,
 ITER const & end,
 QUEUE_OBSERVER * o) [inline]

```

Add a range of descriptors to the used ring, and notify an observer once.

The iterators are passed to [consumed<ITER>\(ITER const &, ITER const &\)](#), and the requirements detailed there apply.

**Template Parameters**

|                       |                                      |
|-----------------------|--------------------------------------|
| <i>ITER</i>           | type of the iterator (inferred)      |
| <i>QUEUE_OBSERVER</i> | the type of the observer (inferred). |

### Parameters

|              |                                             |
|--------------|---------------------------------------------|
| <i>begin</i> | iterator pointing to first element.         |
| <i>end</i>   | iterator pointing to one past last element. |
| <i>o</i>     | pointer to the observer that is notified.   |

### Precondition

queue must be in working state.

Definition at line 262 of file [virtio](#).

References [consumed\(\)](#).

Here is the call graph for this function:



### 16.438.2.9 next\_avail()

```
Request L4virtio::Svr::Virtqueue::next_avail () [inline]
```

Get the next available descriptor from the available ring.

### Precondition

The queue must be in working state.

### Returns

A Request for the next available descriptor, the Request is invalid if there are no descriptors in the available ring.

**Note**

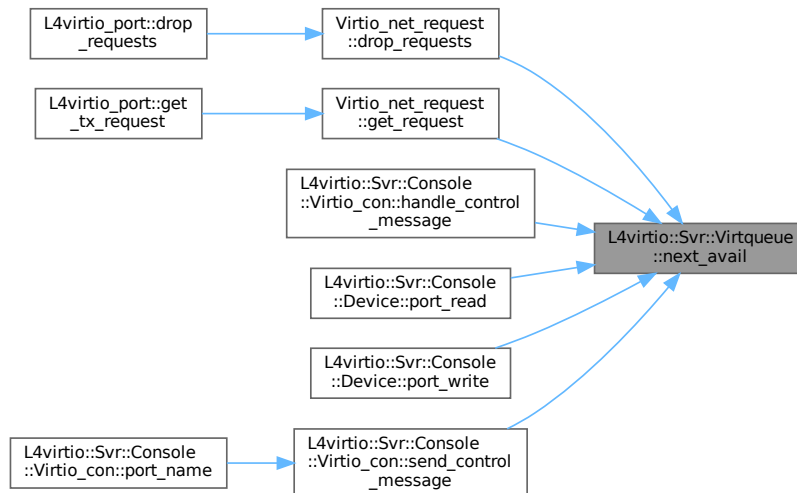
The return value must be checked even when a previous `desc_avail()` returned true.

Definition at line 136 of file `virtio`.

References `L4virtio::Virtqueue::_avail`, `L4virtio::Virtqueue::_current_avail`, `L4virtio::Virtqueue::_idx_mask`, and `L4_LIKELY`.

Referenced by `Virtio_net_request::drop_requests()`, `Virtio_net_request::get_request()`, `L4virtio::Svr::Console::Virtio_con::handle_control_message`, `L4virtio::Svr::Console::Device::port_read()`, `L4virtio::Svr::Console::Device::port_write()`, and `L4virtio::Svr::Console::Virtio_con::send_control_message`.

Here is the caller graph for this function:

**16.438.2.10 rewind\_avail()**

```
void L4virtio::Svr::Virtqueue::rewind_avail (
 Head_desc const & d) [inline]
```

Return unfinished descriptors to the available ring, i.e.

reset the local next index of the available ring to the given descriptor.

**Parameters**

|          |                                                             |
|----------|-------------------------------------------------------------|
| <i>d</i> | descriptor of the request that is to be marked as finished. |
|----------|-------------------------------------------------------------|

**Precondition**

queue must be in working state.

*d* must be a valid request from this queue, obtained via `next_avail()`, that has not yet been finished, and in addition, no descriptors following it have been finished.

Definition at line 160 of file `virtio`.

References `L4virtio::Virtqueue::_current_avail`, `L4virtio::Virtqueue::_desc`, and `L4virtio::Virtqueue::_idx_mask`.

The documentation for this class was generated from the following file:

- `I4/I4virtio/server/virtio`

## 16.439 L4virtio::Svr::Virtqueue::Head\_desc Class Reference

VIRTIO request, essentially a descriptor from the available ring.

```
#include <virtio>
```

Collaboration diagram for L4virtio::Svr::Virtqueue::Head\_desc:

| L4virtio::Svr::Virtqueue<br>::Head_desc |                 |
|-----------------------------------------|-----------------|
|                                         |                 |
| +                                       | Head_desc()     |
| +                                       | valid()         |
| +                                       | operator bool() |
| +                                       | desc()          |

### Public Member Functions

- **Head\_desc** ()  
*Make invalid (NULL) request.*
- bool **valid** () const
- **operator bool** () const
- **Desc** const \* **desc** () const

### 16.439.1 Detailed Description

VIRTIO request, essentially a descriptor from the available ring.

Definition at line 93 of file [virtio](#).

### 16.439.2 Member Function Documentation

#### 16.439.2.1 desc()

```
Desc const * L4virtio::Svr::Virtqueue::Head_desc::desc () const [inline]
```



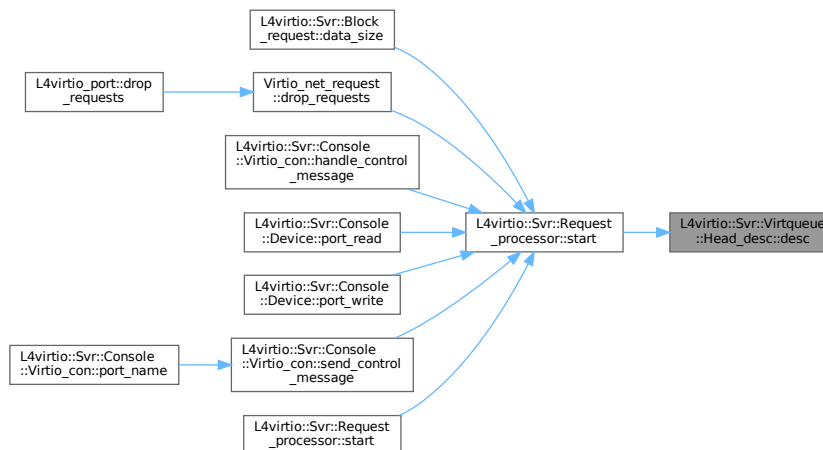
**Returns**

Pointer to the head descriptor of the request.

Definition at line 112 of file [virtio](#).

Referenced by [L4virtio::Svr::Request\\_processor::start\(\)](#).

Here is the caller graph for this function:

**16.439.2.2 operator bool()**

```
L4virtio::Svr::Virtqueue::Head_desc::operator bool () const [inline], [explicit]
```

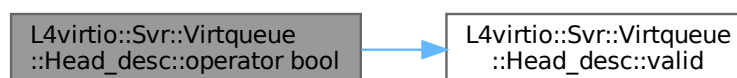
**Returns**

True if the request is valid (not NULL).

Definition at line 108 of file [virtio](#).

References [valid\(\)](#).

Here is the call graph for this function:



### 16.439.2.3 valid()

```
bool L4virtio::Svr::Virtqueue::Head_desc::valid () const [inline]
```

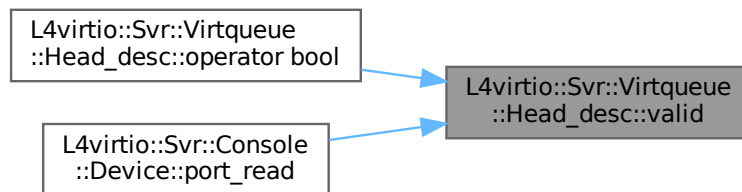
#### Returns

True if the request is valid (not NULL).

Definition at line 105 of file [virtio](#).

Referenced by [operator bool\(\)](#), and [L4virtio::Svr::Console::Device::port\\_read\(\)](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

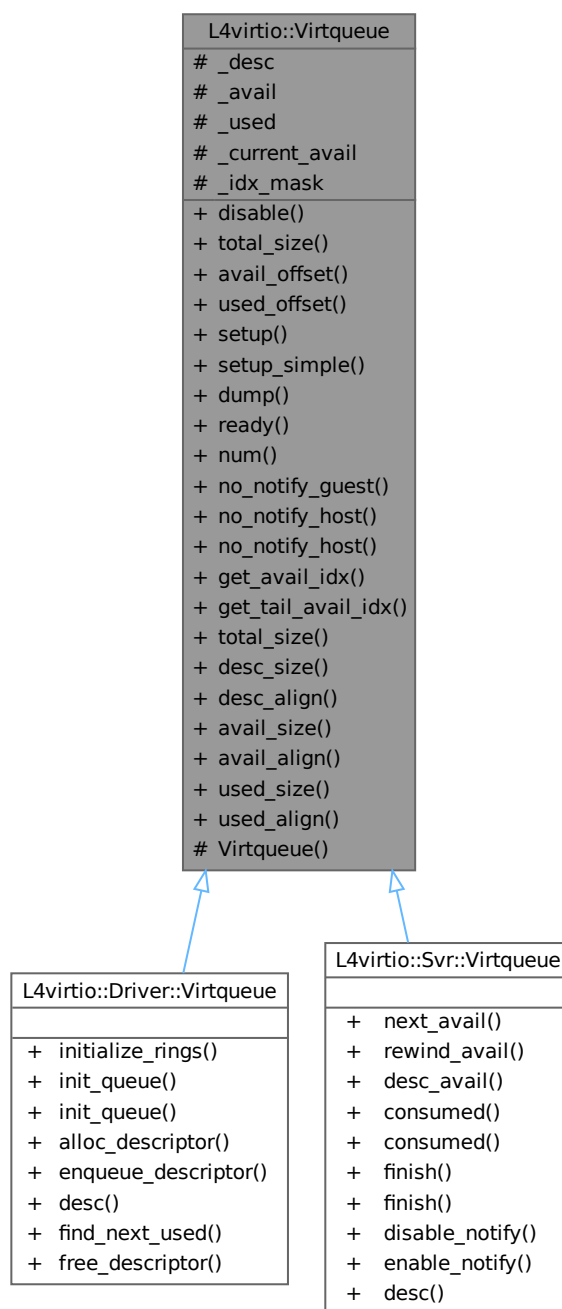
- `I4/I4virtio/server/virtio`

## 16.440 L4virtio::Virtqueue Class Reference

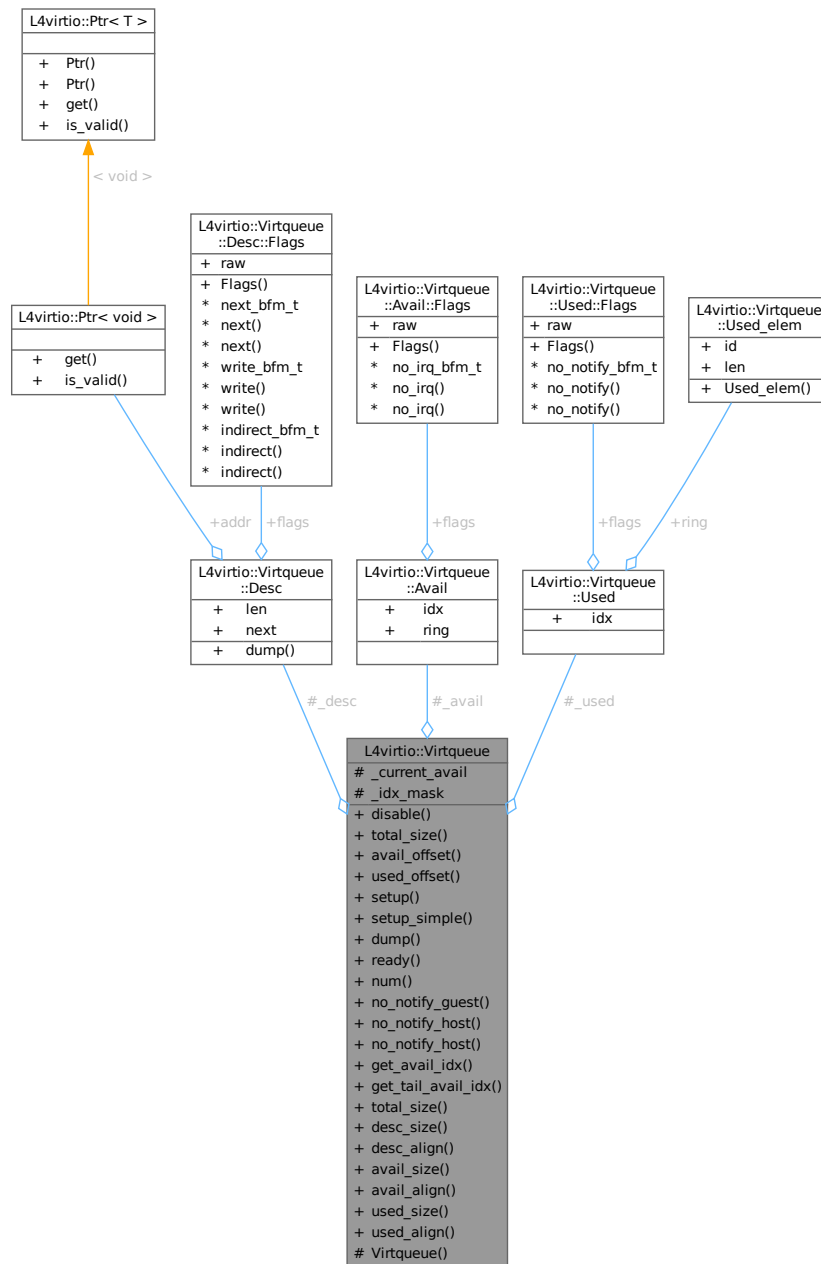
Low-level [Virtqueue](#).

```
#include <virtqueue>
```

Inheritance diagram for L4virtio::Virtqueue:



Collaboration diagram for L4virtio::Virtqueue:



## Data Structures

- class [Desc](#)  
*Descriptor in the descriptor table.*
- class [Avail](#)  
*Type of available ring, this is read-only for the host.*
- struct [Used\\_elem](#)  
*Type of an element of the used ring.*
- class [Used](#)  
*Used ring.*

## Public Types

- enum  
*Fixed alignment values for different parts of a virtqueue.*

## Public Member Functions

- void `disable` ()  
*Completely disable the queue.*
- unsigned long `total_size` () const  
*Calculate the total size of this virtqueue.*
- unsigned long `avail_offset` () const  
*Get the offset of the available ring from the descriptor table.*
- unsigned long `used_offset` () const  
*Get the offset of the used ring from the descriptor table.*
- void `setup` (unsigned `num`, void \*`desc`, void \*`avail`, void \*`used`)  
*Enable this queue.*
- void `setup_simple` (unsigned `num`, void \*`ring`)  
*Enable this queue.*
- void `dump` (Desc const \*`d`) const  
*Dump descriptors for this queue.*
- bool `ready` () const  
*Test if this queue is in working state.*
- unsigned `num` () const
- bool `no_notify_guest` () const  
*Get the no IRQ flag of this queue.*
- bool `no_notify_host` () const  
*Get the no notify flag of this queue.*
- void `no_notify_host` (bool `value`)  
*Set the no-notify flag for this queue.*
- `l4_uint16_t` `get_avail_idx` () const  
*Get available index from available ring (for debugging).*
- `l4_uint16_t` `get_tail_avail_idx` () const  
*Get tail-available index stored in local state (for debugging).*

## Static Public Member Functions

- static unsigned long `total_size` (unsigned `num`)  
*Calculate the total size for a virtqueue of the given dimensions.*
- static unsigned long `desc_size` (unsigned `num`)  
*Calculate the size of the descriptor table for `num` entries.*
- static unsigned long `desc_align` ()  
*Get the alignment in zero LSBs needed for the descriptor table.*
- static unsigned long `avail_size` (unsigned `num`)  
*Calculate the size of the available ring for `num` entries.*
- static unsigned long `avail_align` ()  
*Get the alignment in zero LSBs needed for the available ring.*
- static unsigned long `used_size` (unsigned `num`)  
*Calculate the size of the used ring for `num` entries.*
- static unsigned long `used_align` ()  
*Get the alignment in zero LSBs needed for the used ring.*

## Protected Member Functions

- **Virtqueue** ()=default  
*Create a disabled virtqueue.*

## Protected Attributes

- **Desc** \* **\_desc** = nullptr  
*pointer to descriptor table, NULL if queue is off.*
- **Avail** \* **\_avail** = nullptr  
*pointer to available ring.*
- **Used** \* **\_used** = nullptr  
*pointer to used ring.*
- **l4\_uint16\_t** **\_current\_avail** = 0  
*The life counter for the queue.*
- **l4\_uint16\_t** **\_idx\_mask** = 0  
*mask used for indexing into the descriptor table and the rings.*

### 16.440.1 Detailed Description

Low-level [Virtqueue](#).

This class represents a single virtqueue, with a local running available index.

#### Note

The [Virtqueue](#) implementation is not thread-safe.

Definition at line 80 of file [virtqueue](#).

### 16.440.2 Member Function Documentation

#### 16.440.2.1 [avail\\_align\(\)](#)

```
unsigned long L4virtio::Virtqueue::avail_align () [inline], [static]
```

Get the alignment in zero LSBs needed for the available ring.

#### Returns

The alignment in zero LSBs needed for an available ring.

Definition at line 287 of file [virtqueue](#).

#### 16.440.2.2 [avail\\_size\(\)](#)

```
unsigned long L4virtio::Virtqueue::avail_size (
 unsigned num) [inline], [static]
```

Calculate the size of the available ring for [num](#) entries.

#### Parameters

---

|            |                                              |
|------------|----------------------------------------------|
| <i>num</i> | The number of entries in the available ring. |
|------------|----------------------------------------------|

### Returns

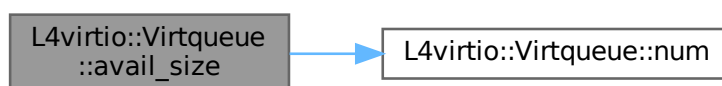
The size in bytes needed for an available ring with [num](#) entries.

Definition at line 279 of file [virtqueue](#).

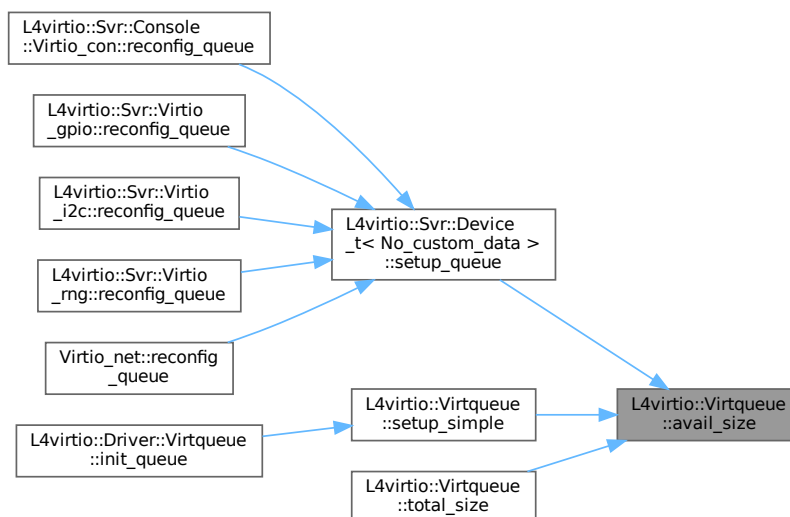
References [num\(\)](#).

Referenced by [L4virtio::Svr::Device\\_t< No\\_custom\\_data >::setup\\_queue\(\)](#), [setup\\_simple\(\)](#), and [total\\_size\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



**16.440.2.3 desc\_align()**

```
unsigned long L4virtio::Virtqueue::desc_align () [inline], [static]
```

Get the alignment in zero LSBs needed for the descriptor table.

**Returns**

The alignment in zero LSBs needed for a descriptor table.

Definition at line 269 of file [virtqueue](#).

**16.440.2.4 desc\_size()**

```
unsigned long L4virtio::Virtqueue::desc_size (
 unsigned num) [inline], [static]
```

Calculate the size of the descriptor table for [num](#) entries.

**Parameters**

|            |                                                |
|------------|------------------------------------------------|
| <i>num</i> | The number of entries in the descriptor table. |
|------------|------------------------------------------------|

**Returns**

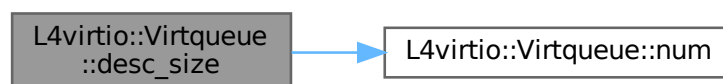
The size in bytes needed for a descriptor table with [num](#) entries.

Definition at line 261 of file [virtqueue](#).

References [num\(\)](#).

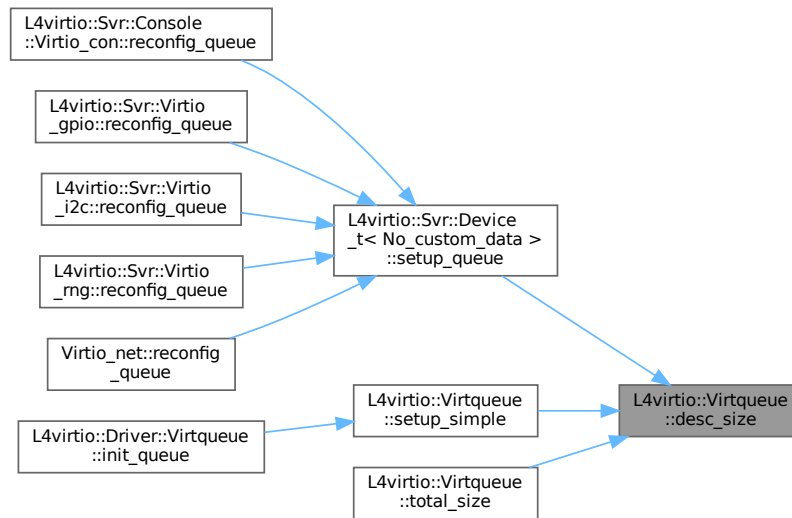
Referenced by [L4virtio::Svr::Device\\_t< No\\_custom\\_data >::setup\\_queue\(\)](#), [setup\\_simple\(\)](#), and [total\\_size\(\)](#).

Here is the call graph for this function:





Here is the caller graph for this function:



#### 16.440.2.5 disable()

```
void L4virtio::Virtqueue::disable () [inline]
```

Completely disable the queue.

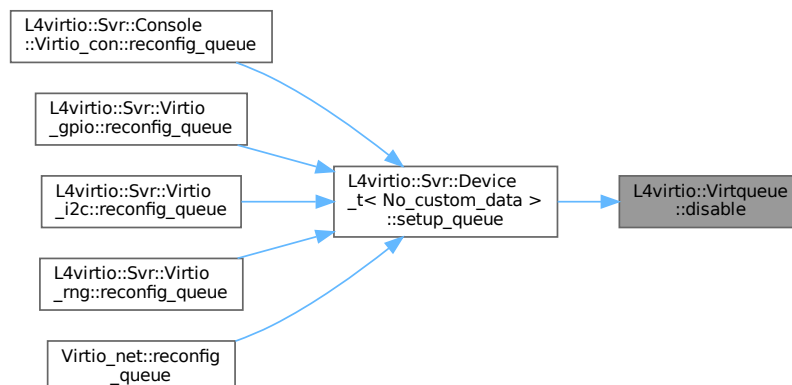
[setup\(\)](#) must be used to enable the queue again.

Definition at line 224 of file [virtqueue](#).

References [\\_desc](#).

Referenced by [L4virtio::Svr::Device\\_t< No\\_custom\\_data >::setup\\_queue\(\)](#).

Here is the caller graph for this function:



### 16.440.2.6 dump()

```
void L4virtio::Virtqueue::dump (
 Desc const * d) const [inline]
```

Dump descriptors for this queue.

#### Precondition

the queue must be in working state.

Definition at line 392 of file [virtqueue](#).

References [\\_desc](#), and [L4virtio::Virtqueue::Desc::dump\(\)](#).

Here is the call graph for this function:



### 16.440.2.7 get\_avail\_idx()

```
l4_uint16_t L4virtio::Virtqueue::get_avail_idx () const [inline]
```

Get available index from available ring (for debugging).

#### Precondition

Queue must be in a working state.

#### Returns

current index in the available ring (shared between device model and device driver).

Definition at line 449 of file [virtqueue](#).

References [\\_avail](#).

### 16.440.2.8 get\_tail\_avail\_idx()

```
l4_uint16_t L4virtio::Virtqueue::get_tail_avail_idx () const [inline]
```

Get tail-available index stored in local state (for debugging).

#### Returns

current tail index for the the available ring.

Definition at line 456 of file [virtqueue](#).

References [\\_current\\_avail](#).

### 16.440.2.9 no\_notify\_guest()

```
bool L4virtio::Virtqueue::no_notify_guest () const [inline]
```

Get the no IRQ flag of this queue.

#### Precondition

queue must be in working state.

#### Returns

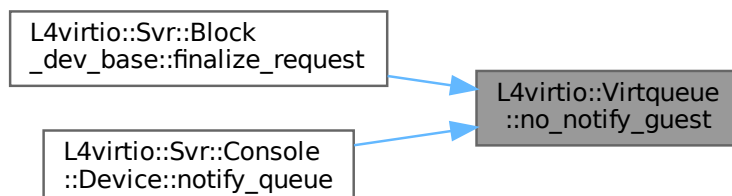
true if the guest does not want to get IRQs (currently).

Definition at line 414 of file [virtqueue](#).

References [\\_avail](#).

Referenced by [L4virtio::Svr::Block\\_dev\\_base<Ds\\_data>::finalize\\_request\(\)](#), and [L4virtio::Svr::Console::Device::notify\\_queue\(\)](#).

Here is the caller graph for this function:



**16.440.2.10 no\_notify\_host()** [1/2]

```
bool L4virtio::Virtqueue::no_notify_host () const [inline]
```

Get the no notify flag of this queue.

**Precondition**

queue must be in working state.

**Returns**

true if the host does not want to get IRQs (currently).

Definition at line [426](#) of file [virtqueue](#).

References [\\_used](#).

**16.440.2.11 no\_notify\_host()** [2/2]

```
void L4virtio::Virtqueue::no_notify_host (
 bool value) [inline]
```

Set the no-notify flag for this queue.

**Precondition**

Queue must be in a working state.

Definition at line [436](#) of file [virtqueue](#).

References [\\_used](#).

**16.440.2.12 num()**

```
unsigned L4virtio::Virtqueue::num () const [inline]
```

## Returns

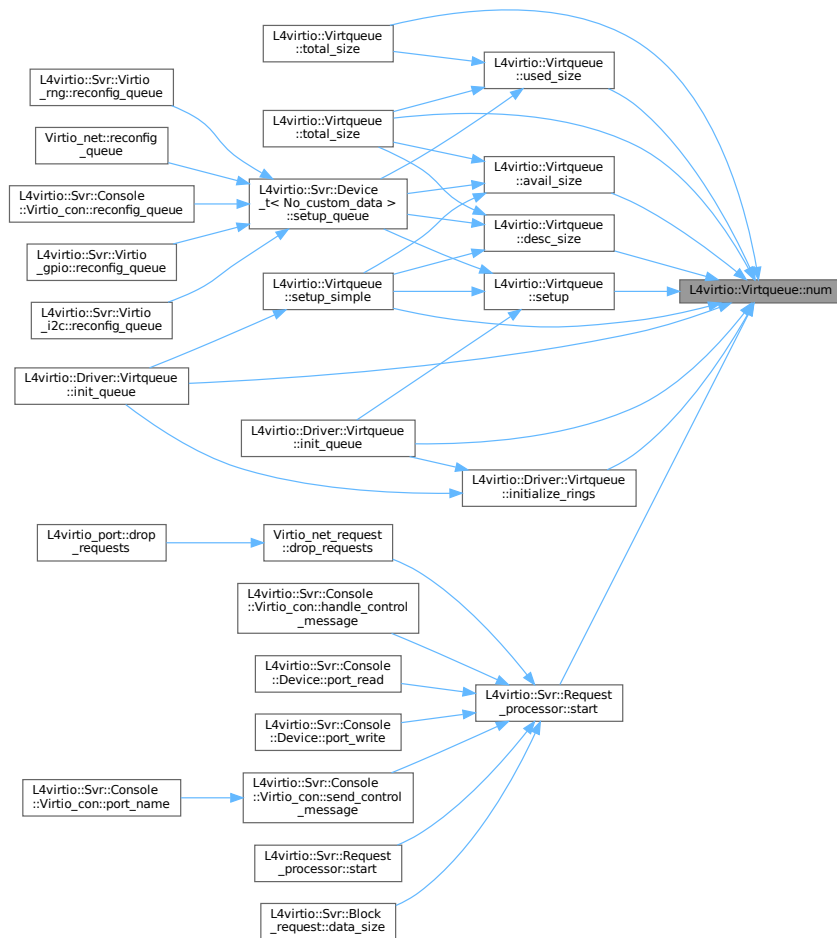
The number of entries in the ring.

Definition at line 404 of file [virtqueue](#).

References [\\_idx\\_mask](#).

Referenced by [avail\\_size\(\)](#), [desc\\_size\(\)](#), [L4virtio::Driver::Virtqueue::init\\_queue\(\)](#), [L4virtio::Driver::Virtqueue::init\\_queue\(\)](#), [L4virtio::Driver::Virtqueue::initialize\\_rings\(\)](#), [setup\(\)](#), [setup\\_simple\(\)](#), [L4virtio::Svr::Request\\_processor::start\(\)](#), [total\\_size\(\)](#), [total\\_size\(\)](#), and [used\\_size\(\)](#).

Here is the caller graph for this function:



## 16.440.2.13 ready()

```
bool L4virtio::Virtqueue::ready () const [inline]
```

Test if this queue is in working state.

**Returns**

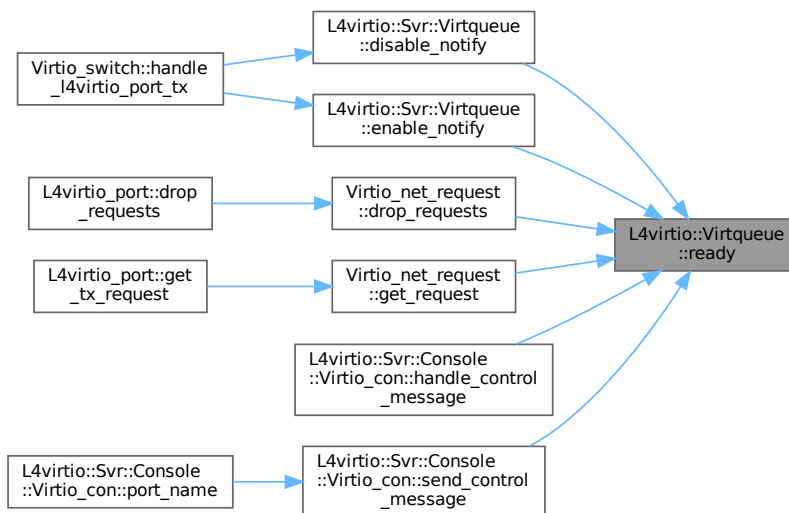
true when the queue is in working state, false else.

Definition at line 400 of file [virtqueue](#).

References [\\_desc](#), and [L4\\_LIKELY](#).

Referenced by [L4virtio::Svr::Virtqueue::disable\\_notify\(\)](#), [Virtio\\_net\\_request::drop\\_requests\(\)](#), [L4virtio::Svr::Virtqueue::enable\\_notify\(\)](#), [Virtio\\_net\\_request::get\\_request\(\)](#), [L4virtio::Svr::Console::Virtio\\_con::handle\\_control\\_message\(\)](#), and [L4virtio::Svr::Console::Virtio\\_con::port\\_name\(\)](#).

Here is the caller graph for this function:

**16.440.2.14 setup()**

```

void L4virtio::Virtqueue::setup (
 unsigned num,
 void * desc,
 void * avail,
 void * used) [inline]

```

Enable this queue.

**Parameters**

|              |                                                                                                                                        |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <i>num</i>   | The number of entries in the descriptor table, the available ring, and the used ring (must be a power of 2).                           |
| <i>desc</i>  | The address of the descriptor table. (Must be <code>Desc_align</code> aligned and at least <code>desc_size(num)</code> bytes in size.) |
| <i>avail</i> | The address of the available ring. (Must be <code>Avail_align</code> aligned and at least <code>avail_size(num)</code> bytes in size.) |

|             |                                                                                                                                 |
|-------------|---------------------------------------------------------------------------------------------------------------------------------|
| <i>used</i> | The address of the used ring. (Must be <code>Used_align</code> aligned and at least <code>used_size(num)</code> bytes in size.) |
|-------------|---------------------------------------------------------------------------------------------------------------------------------|

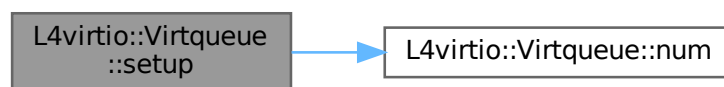
Due to the data type of the descriptors, the queue can have a maximum size of  $2^{16}$ .

Definition at line 349 of file [virtqueue](#).

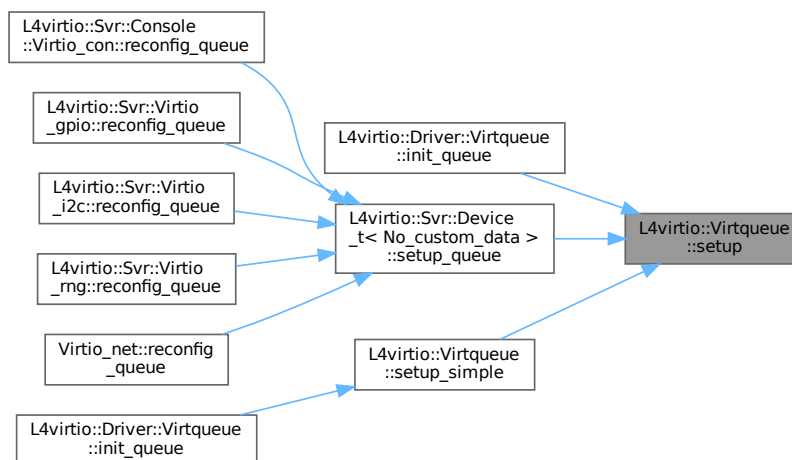
References [\\_avail](#), [\\_current\\_avail](#), [\\_desc](#), [\\_idx\\_mask](#), [\\_used](#), [L4\\_EINVAL](#), and [num\(\)](#).

Referenced by [L4virtio::Driver::Virtqueue::init\\_queue\(\)](#), [L4virtio::Svr::Device\\_t< No\\_custom\\_data >::setup\\_queue\(\)](#), and [setup\\_simple\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 16.440.2.15 setup\_simple()

```
void L4virtio::Virtqueue::setup_simple (
 unsigned num,
 void * ring) [inline]
```

Enable this queue.

#### Parameters

|             |                                                                                                                                                                                                                       |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>num</i>  | The number of entries in the descriptor table, the available ring, and the used ring (must be a power of 2).                                                                                                          |
| <i>ring</i> | The base address for the queue data structure. The memory block at <code>ring</code> must be at least <code>total_size(num)</code> bytes in size and have an alignment of <code>Desc_align(desc_align())</code> bits. |

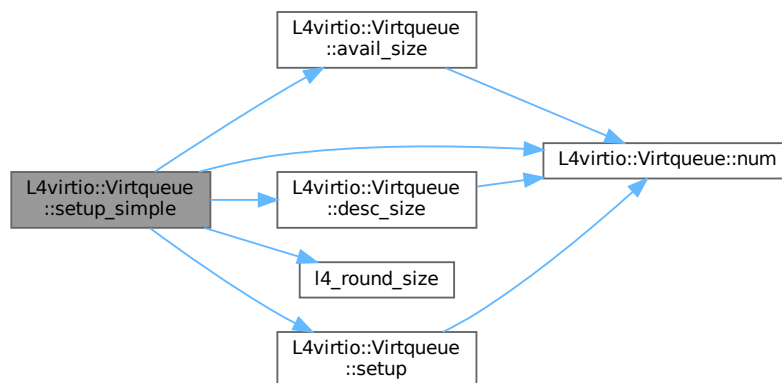
Due to the data type of the descriptors, the queue can have a maximum size of  $2^{16}$ .

Definition at line 378 of file [virtqueue](#).

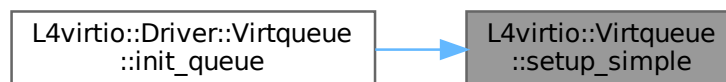
References [avail\\_size\(\)](#), [desc\\_size\(\)](#), [l4\\_round\\_size\(\)](#), [num\(\)](#), and [setup\(\)](#).

Referenced by [L4virtio::Driver::Virtqueue::init\\_queue\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 16.440.2.16 total\_size() [1/2]

```
unsigned long L4virtio::Virtqueue::total_size () const [inline]
```

Calculate the total size of this virtqueue.



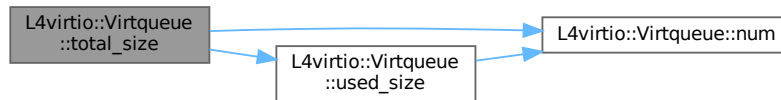
**Precondition**

The queue has been set up.

Definition at line 314 of file [virtqueue](#).

References [\\_desc](#), [\\_used](#), [num\(\)](#), and [used\\_size\(\)](#).

Here is the call graph for this function:

**16.440.2.17 total\_size() [2/2]**

```
unsigned long L4virtio::Virtqueue::total_size (
 unsigned num) [inline], [static]
```

Calculate the total size for a virtqueue of the given dimensions.

**Parameters**

|            |                                                                                                              |
|------------|--------------------------------------------------------------------------------------------------------------|
| <i>num</i> | The number of entries in the descriptor table, the available ring, and the used ring (must be a power of 2). |
|------------|--------------------------------------------------------------------------------------------------------------|

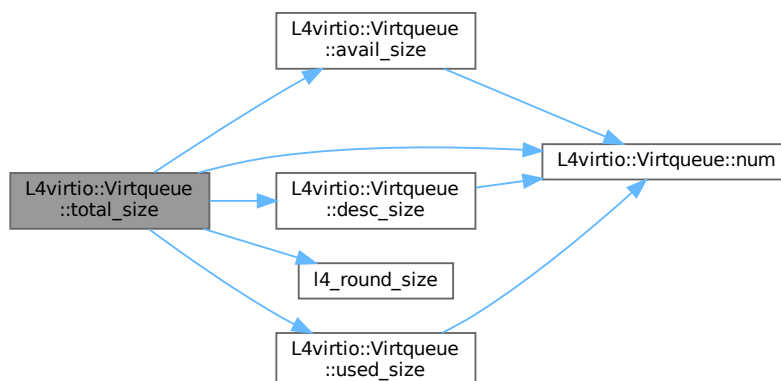
**Returns**

The total size in bytes of the queue data structures.

Definition at line 245 of file [virtqueue](#).

References [avail\\_size\(\)](#), [desc\\_size\(\)](#), [l4\\_round\\_size\(\)](#), [num\(\)](#), and [used\\_size\(\)](#).

Here is the call graph for this function:



#### 16.440.2.18 `used_align()`

```
unsigned long L4virtio::Virtqueue::used_align () [inline], [static]
```

Get the alignment in zero LSBs needed for the used ring.

##### Returns

The alignment in zero LSBs needed for an used ring.

Definition at line 306 of file [virtqueue](#).

#### 16.440.2.19 `used_size()`

```
unsigned long L4virtio::Virtqueue::used_size (
 unsigned num) [inline], [static]
```

Calculate the size of the used ring for [num](#) entries.

##### Parameters

|            |                                         |
|------------|-----------------------------------------|
| <i>num</i> | The number of entries in the used ring. |
|------------|-----------------------------------------|

##### Returns

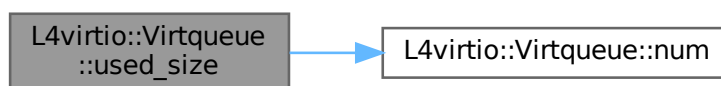
The size in bytes needed for an used ring with [num](#) entries.

Definition at line 298 of file [virtqueue](#).

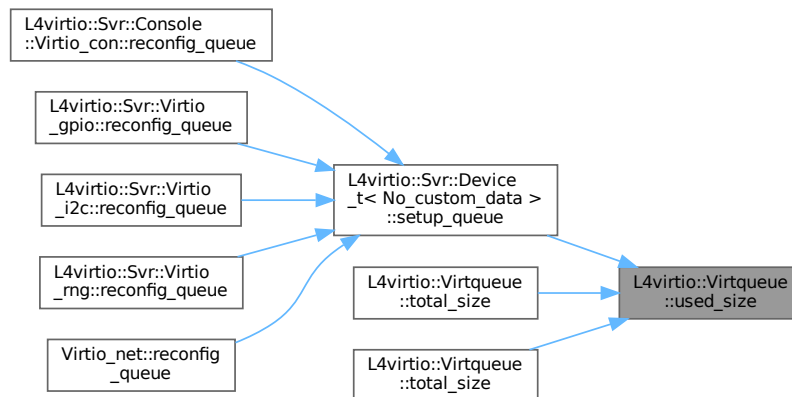
References [num\(\)](#).

Referenced by [L4virtio::Svr::Device\\_t< No\\_custom\\_data >::setup\\_queue\(\)](#), [total\\_size\(\)](#), and [total\\_size\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

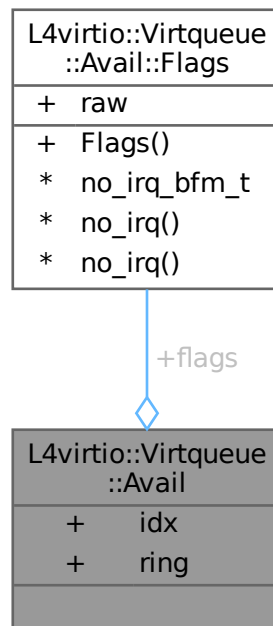
- `I4/I4virtio/virtqueue`

## 16.441 L4virtio::Virtqueue::Avail Class Reference

Type of available ring, this is read-only for the host.

```
#include <virtqueue>
```

Collaboration diagram for L4virtio::Virtqueue::Avail:



## Data Structures

- struct [Flags](#)  
*Flags of the available ring.*

## Data Fields

- [Flags](#) **flags**  
*flags of available ring*
- [l4\\_uint16\\_t](#) **idx**  
*available index written by guest*
- [l4\\_uint16\\_t](#) **ring** []  
*array of available descriptor indexes.*

## 16.441.1 Detailed Description

Type of available ring, this is read-only for the host.

Definition at line [128](#) of file [virtqueue](#).

The documentation for this class was generated from the following file:

- [l4/l4virtio/virtqueue](#)

## 16.442 L4virtio::Virtqueue::Avail::Flags Struct Reference

[Flags](#) of the available ring.

```
#include <virtqueue>
```

Collaboration diagram for L4virtio::Virtqueue::Avail::Flags:

| L4virtio::Virtqueue<br>::Avail::Flags |              |
|---------------------------------------|--------------|
| +                                     | raw          |
| +                                     | Flags()      |
| *                                     | no_irq_bfm_t |
| *                                     | no_irq()     |
| *                                     | no_irq()     |

### Public Member Functions

- **Flags** ([l4\\_uint16\\_t](#) v)  
*Make [Flags](#) from the raw value.*

### Data Fields

- [l4\\_uint16\\_t](#) raw  
*raw 16bit flags value of the available ring.*
- typedef [cxx::Bitfield](#)< decltype(raw), 0, 0 > [no\\_irq\\_bfm\\_t](#)  
*Guest does not want to receive interrupts when requests are finished.*
- constexpr [no\\_irq\\_bfm\\_t::Val](#) no\_irq () const  
*Get the [no\\_irq](#) bits (0 to 0) of [raw](#).*
- constexpr [no\\_irq\\_bfm\\_t::Ref](#) no\_irq ()  
*Get a reference to the [no\\_irq](#) bits (0 to 0) of [raw](#).*

### 16.442.1 Detailed Description

[Flags](#) of the available ring.

Definition at line 134 of file [virtqueue](#).

## 16.442.2 Member Typedef Documentation

### 16.442.2.1 no\_irq\_bfm\_t

```
typedef cxx::Bitfield<decltype(raw), 0, 0> L4virtio::Virtqueue::Avail::Flags::no_irq_bfm_t
```

Guest does not want to receive interrupts when requests are finished.

Type to access the [no\\_irq](#) bits (0 to 0) of [raw](#).

Definition at line [143](#) of file [virtqueue](#).

The documentation for this struct was generated from the following file:

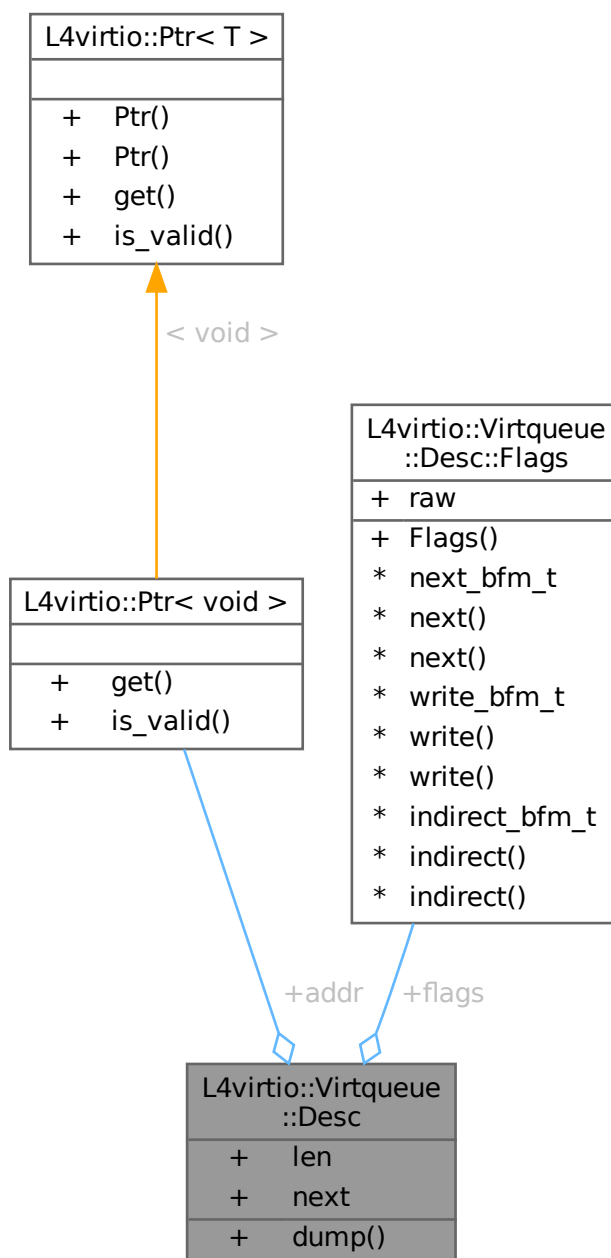
- [l4/l4virtio/virtqueue](#)

## 16.443 L4virtio::Virtqueue::Desc Class Reference

Descriptor in the descriptor table.

```
#include <virtqueue>
```

Collaboration diagram for L4virtio::Virtqueue::Desc:



## Data Structures

- struct [Flags](#)  
Type for descriptor flags.

## Public Member Functions

- void **dump** (unsigned idx) const

*Dump a single descriptor.*

## Data Fields

- **Ptr**< void > **addr**  
*Address stored in descriptor.*
- **l4\_uint32\_t** **len**  
*Length of described buffer.*
- **Flags** **flags**  
*Descriptor flags.*
- **l4\_uint16\_t** **next**  
*Index of the next chained descriptor.*

## 16.443.1 Detailed Description

Descriptor in the descriptor table.

Definition at line 86 of file [virtqueue](#).

The documentation for this class was generated from the following file:

- l4/l4virtio/virtqueue

## 16.444 L4virtio::Virtqueue::Desc::Flags Struct Reference

Type for descriptor flags.

```
#include <virtqueue>
```

Collaboration diagram for L4virtio::Virtqueue::Desc::Flags:

| L4virtio::Virtqueue<br>::Desc::Flags |
|--------------------------------------|
| + raw                                |
| + Flags()                            |
| * next_bfm_t                         |
| * next()                             |
| * next()                             |
| * write_bfm_t                        |
| * write()                            |
| * write()                            |
| * indirect_bfm_t                     |
| * indirect()                         |
| * indirect()                         |



## Public Member Functions

- **Flags** ([l4\\_uint16\\_t](#) v)  
*Make [Flags](#) from raw 16bit value.*

## Data Fields

- [l4\\_uint16\\_t](#) **raw**  
*raw flags value of a virtio descriptor.*
- typedef [cxx::Bitfield](#)< decltype([raw](#)), 0, 0 > [next\\_bfm\\_t](#)  
*Part of a descriptor chain which is continued with the next field.*
- constexpr [next\\_bfm\\_t::Val](#) **next** () const  
*Get the [next](#) bits (0 to 0) of [raw](#).*
- constexpr [next\\_bfm\\_t::Ref](#) **next** ()  
*Get a reference to the [next](#) bits (0 to 0) of [raw](#).*
- typedef [cxx::Bitfield](#)< decltype([raw](#)), 1, 1 > [write\\_bfm\\_t](#)  
*Block described by this descriptor is writeable.*
- constexpr [write\\_bfm\\_t::Val](#) **write** () const  
*Get the [write](#) bits (1 to 1) of [raw](#).*
- constexpr [write\\_bfm\\_t::Ref](#) **write** ()  
*Get a reference to the [write](#) bits (1 to 1) of [raw](#).*
- typedef [cxx::Bitfield](#)< decltype([raw](#)), 2, 2 > [indirect\\_bfm\\_t](#)  
*Indirect descriptor, block contains a list of descriptors.*
- constexpr [indirect\\_bfm\\_t::Val](#) **indirect** () const  
*Get the [indirect](#) bits (2 to 2) of [raw](#).*
- constexpr [indirect\\_bfm\\_t::Ref](#) **indirect** ()  
*Get a reference to the [indirect](#) bits (2 to 2) of [raw](#).*

### 16.444.1 Detailed Description

Type for descriptor flags.

Definition at line 92 of file [virtqueue](#).

### 16.444.2 Member Typedef Documentation

#### 16.444.2.1 indirect\_bfm\_t

```
typedef cxx::Bitfield<decltype(raw), 2, 2> L4virtio::Virtqueue::Desc::Flags::indirect_bfm_t
```

Indirect descriptor, block contains a list of descriptors.

Type to access the [indirect](#) bits (2 to 2) of [raw](#).

Definition at line 105 of file [virtqueue](#).

### 16.444.2.2 next\_bfm\_t

```
typedef cxx::Bitfield<decltype(raw), 0, 0> L4virtio::Virtqueue::Desc::Flags::next_bfm_t
```

Part of a descriptor chain which is continued with the next field.

Type to access the `next` bits (0 to 0) of `raw`.

Definition at line 101 of file `virtqueue`.

### 16.444.2.3 write\_bfm\_t

```
typedef cxx::Bitfield<decltype(raw), 1, 1> L4virtio::Virtqueue::Desc::Flags::write_bfm_t
```

Block described by this descriptor is writeable.

Type to access the `write` bits (1 to 1) of `raw`.

Definition at line 103 of file `virtqueue`.

The documentation for this struct was generated from the following file:

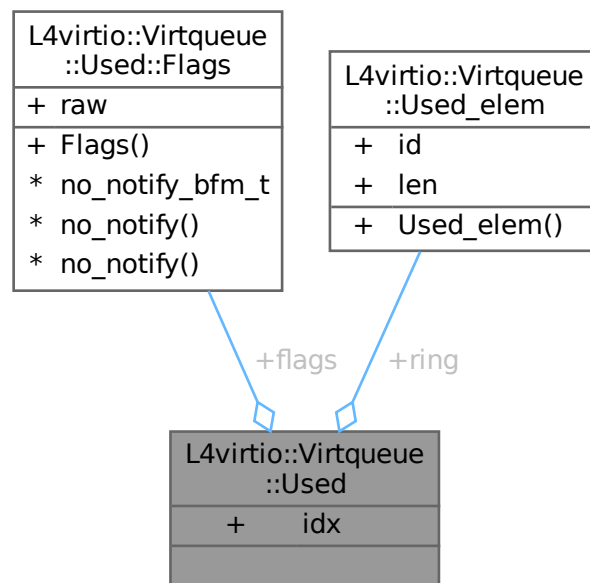
- `l4/l4virtio/virtqueue`

## 16.445 L4virtio::Virtqueue::Used Class Reference

`Used` ring.

```
#include <virtqueue>
```

Collaboration diagram for `L4virtio::Virtqueue::Used`:



## Data Structures

- struct [Flags](#)  
*flags for the used ring.*

## Data Fields

- [Flags](#) **flags**  
*flags of the used ring.*
- [l4\\_uint16\\_t](#) **idx**  
*index of the last entry in the ring.*
- [Used\\_elem](#) **ring** []  
*array of used descriptors.*

### 16.445.1 Detailed Description

[Used](#) ring.

Definition at line 173 of file [virtqueue](#).

The documentation for this class was generated from the following file:

- l4/l4virtio/virtqueue

## 16.446 L4virtio::Virtqueue::Used::Flags Struct Reference

flags for the used ring.

```
#include <virtqueue>
```

Collaboration diagram for L4virtio::Virtqueue::Used::Flags:

| L4virtio::Virtqueue<br>::Used::Flags |
|--------------------------------------|
| + raw                                |
| + Flags()                            |
| * no_notify_bfm_t                    |
| * no_notify()                        |
| * no_notify()                        |

## Public Member Functions

- **Flags** ([l4\\_uint16\\_t](#) v)  
*make [Flags](#) from raw value*

## Data Fields

- [l4\\_uint16\\_t](#) **raw**  
*raw flags value as specified by virtio.*
- typedef [cxx::Bitfield](#)< decltype([raw](#)), 0, 0 > **no\_notify\_bfm\_t**  
*host does not want to be notified when new requests have been queued.*
- constexpr [no\\_notify\\_bfm\\_t::Val](#) **no\_notify** () const  
*Get the [no\\_notify](#) bits (0 to 0) of [raw](#).*
- constexpr [no\\_notify\\_bfm\\_t::Ref](#) **no\_notify** ()  
*Get a reference to the [no\\_notify](#) bits (0 to 0) of [raw](#).*

### 16.446.1 Detailed Description

flags for the used ring.

Definition at line 179 of file [virtqueue](#).

### 16.446.2 Member Typedef Documentation

#### 16.446.2.1 no\_notify\_bfm\_t

```
typedef cxx::Bitfield<decltype(raw), 0, 0> L4virtio::Virtqueue::Used::Flags::no_notify_bfm_t
```

host does not want to be notified when new requests have been queued.

Type to access the [no\\_notify](#) bits (0 to 0) of [raw](#).

Definition at line 188 of file [virtqueue](#).

The documentation for this struct was generated from the following file:

- [l4/l4virtio/virtqueue](#)

## 16.447 L4virtio::Virtqueue::Used\_elem Struct Reference

Type of an element of the used ring.

```
#include <virtqueue>
```

Collaboration diagram for L4virtio::Virtqueue::Used\_elem:

| L4virtio::Virtqueue<br>::Used_elem |
|------------------------------------|
| + id                               |
| + len                              |
| + Used_elem()                      |

### Public Member Functions

- [Used\\_elem](#) ([l4\\_uint16\\_t id](#), [l4\\_uint32\\_t len](#))  
*Initialize a used ring element.*

### Data Fields

- [l4\\_uint32\\_t id](#)  
*descriptor index*
- [l4\\_uint32\\_t len](#)  
*length field*

### 16.447.1 Detailed Description

Type of an element of the used ring.

Definition at line [154](#) of file [virtqueue](#).

### 16.447.2 Constructor & Destructor Documentation

#### 16.447.2.1 Used\_elem()

```
L4virtio::Virtqueue::Used_elem::Used_elem (
 l4_uint16_t id,
 l4_uint32_t len) [inline]
```

Initialize a used ring element.

#### Parameters

---

|            |                                                                  |
|------------|------------------------------------------------------------------|
| <i>id</i>  | The index of the descriptor to be marked as used.                |
| <i>len</i> | The total bytes written into the buffer of the descriptor chain. |

Definition at line 165 of file [virtqueue](#).

References [id](#), and [len](#).

The documentation for this struct was generated from the following file:

- [l4/l4virtio/virtqueue](#)

## 16.448 l4virtio\_block\_config\_t Struct Reference

Device configuration for block devices.

```
#include <virtio_block.h>
```

Collaboration diagram for `l4virtio_block_config_t`:

| l4virtio_block_config_t |          |
|-------------------------|----------|
| +                       | capacity |
| +                       | size_max |
| +                       | seg_max  |
| +                       | blk_size |
|                         |          |

### Data Fields

- [l4\\_uint64\\_t](#) **capacity**  
*Capacity of device in 512-byte sectors.*
- [l4\\_uint32\\_t](#) **size\_max**  
*Maximum size of a single segment.*
- [l4\\_uint32\\_t](#) **seg\_max**  
*Maximum number of segments per request.*
- [l4\\_uint32\\_t](#) **blk\_size**  
*Block size of underlying disk.*

### 16.448.1 Detailed Description

Device configuration for block devices.

Definition at line 68 of file [virtio\\_block.h](#).

The documentation for this struct was generated from the following file:

- l4/l4virtio/virtio\_block.h

## 16.449 l4virtio\_block\_discard\_t Struct Reference

Structure used for the write zeroes and discard commands.

```
#include <virtio_block.h>
```

Collaboration diagram for l4virtio\_block\_discard\_t:



### 16.449.1 Detailed Description

Structure used for the write zeroes and discard commands.

Definition at line 58 of file [virtio\\_block.h](#).

The documentation for this struct was generated from the following file:

- l4/l4virtio/virtio\_block.h

## 16.450 l4virtio\_block\_header\_t Struct Reference

Header structure of a request for a block device.

```
#include <virtio_block.h>
```

Collaboration diagram for l4virtio\_block\_header\_t:

| l4virtio_block_header_t |        |
|-------------------------|--------|
| +                       | type   |
| +                       | ioprio |
| +                       | sector |
|                         |        |

### Data Fields

- [l4\\_uint32\\_t](#) **type**  
*Kind of request, see [L4virtio\\_block\\_operations](#).*
- [l4\\_uint32\\_t](#) **ioprio**  
*Priority (unused).*
- [l4\\_uint64\\_t](#) **sector**  
*First sector to read/write.*

### 16.450.1 Detailed Description

Header structure of a request for a block device.

Definition at line 42 of file [virtio\\_block.h](#).

The documentation for this struct was generated from the following file:

- l4/l4virtio/virtio\_block.h



## 16.451 l4virtio\_config\_hdr\_t Struct Reference

L4-VIRTIO config header, provided in shared data space.

```
#include <virtio.h>
```

Collaboration diagram for l4virtio\_config\_hdr\_t:

| l4virtio_config_hdr_t |
|-----------------------|
| + magic               |
| + version             |
| + device              |
| + vendor              |
| + dev_features        |
| + num_queues          |
| + queues_offset       |

### Data Fields

- [l4\\_uint32\\_t](#) **magic**  
*magic value (must be 'virt').*
- [l4\\_uint32\\_t](#) **version**  
*VIRTIO version.*
- [l4\\_uint32\\_t](#) **device**  
*device ID*
- [l4\\_uint32\\_t](#) **vendor**  
*vendor ID*
- [l4\\_uint32\\_t](#) **dev\_features**  
*device features windows selected by device\_feature\_sel*
- [l4\\_uint32\\_t](#) **num\_queues**  
*number of virtqueues*
- [l4\\_uint32\\_t](#) **queues\_offset**  
*offset of virtqueue config array*

### 16.451.1 Detailed Description

L4-VIRTIO config header, provided in shared data space.

Definition at line 132 of file [virtio.h](#).

The documentation for this struct was generated from the following file:

- l4/l4virtio/virtio.h

## 16.452 l4virtio\_config\_queue\_t Struct Reference

Queue configuration entry.

```
#include <virtio.h>
```

Collaboration diagram for l4virtio\_config\_queue\_t:

| l4virtio_config_queue_t |
|-------------------------|
| + num_max               |
| + num                   |
| + ready                 |
| + driver_notify_index   |
| + desc_addr             |
| + avail_addr            |
| + used_addr             |
| + device_notify_index   |

### Data Fields

- [l4\\_uint16\\_t](#) **num\_max**  
*R: maximum number of descriptors supported by this queue.*
- [l4\\_uint16\\_t](#) **num**  
*RW: number of descriptors configured for this queue.*
- [l4\\_uint16\\_t](#) **ready**  
*RW: queue ready flag (read-write).*
- [l4\\_uint16\\_t](#) **driver\_notify\_index**  
*W: Event index to be used for device notifications (device to driver).*
- [l4\\_uint64\\_t](#) **desc\_addr**  
*W: address of descriptor table.*
- [l4\\_uint64\\_t](#) **avail\_addr**  
*W: address of available ring.*
- [l4\\_uint64\\_t](#) **used\_addr**  
*W: address of used ring.*
- [l4\\_uint16\\_t](#) **device\_notify\_index**  
*R: Event index to be used by the driver (driver to device).*

### 16.452.1 Detailed Description

Queue configuration entry.

An array of such entries is available at the [l4virtio\\_config\\_hdr\\_t::queues\\_offset](#) in the config data space.

Consistency rules for the queue config are:

- A driver might read [num\\_max](#) at any time.
- A driver must write to [num](#), [desc\\_addr](#), [avail\\_addr](#), and [used\\_addr](#) only when [ready](#) is zero (0). Values in these fields are validated and used by the device only after successfully setting [ready](#) to one (1), either by the IPC or by L4VIRTIO\_CMD\_CFG\_QUEUE.
- The value of [device\\_notify\\_index](#) is valid only when [ready](#) is one.
- The driver might write to [device\\_notify\\_index](#) at any time, however the change is guaranteed to take effect after a successful L4VIRTIO\_CMD\_CFG\_QUEUE or after a config\_queue IPC. Note, the change might also have immediate effect, depending on the device implementation.

Definition at line [223](#) of file [virtio.h](#).

The documentation for this struct was generated from the following file:

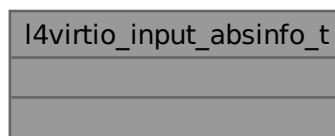
- l4/l4virtio/virtio.h

## 16.453 l4virtio\_input\_absinfo\_t Struct Reference

Information about the absolute axis in the underlying evdev implementation.

```
#include <virtio_input.h>
```

Collaboration diagram for l4virtio\_input\_absinfo\_t:



### 16.453.1 Detailed Description

Information about the absolute axis in the underlying evdev implementation.

Definition at line [33](#) of file [virtio\\_input.h](#).

The documentation for this struct was generated from the following file:

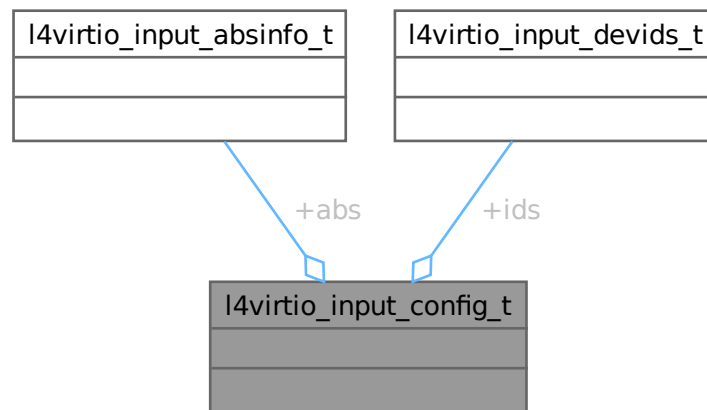
- l4/l4virtio/virtio\_input.h

## 16.454 l4virtio\_input\_config\_t Struct Reference

Device configuration for input devices.

```
#include <virtio_input.h>
```

Collaboration diagram for l4virtio\_input\_config\_t:



### 16.454.1 Detailed Description

Device configuration for input devices.

Definition at line 56 of file [virtio\\_input.h](#).

The documentation for this struct was generated from the following file:

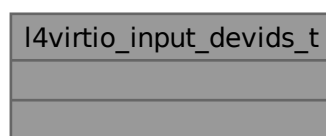
- l4/l4virtio/virtio\_input.h

## 16.455 l4virtio\_input\_devids\_t Struct Reference

Device ID information for the device.

```
#include <virtio_input.h>
```

Collaboration diagram for l4virtio\_input\_devids\_t:



### 16.455.1 Detailed Description

Device ID information for the device.

Definition at line 45 of file [virtio\\_input.h](#).

The documentation for this struct was generated from the following file:

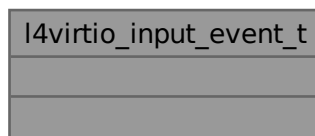
- l4/l4virtio/virtio\_input.h

## 16.456 l4virtio\_input\_event\_t Struct Reference

Single event in event or status queue.

```
#include <virtio_input.h>
```

Collaboration diagram for l4virtio\_input\_event\_t:



### 16.456.1 Detailed Description

Single event in event or status queue.

Definition at line 74 of file [virtio\\_input.h](#).

The documentation for this struct was generated from the following file:

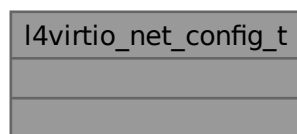
- l4/l4virtio/virtio\_input.h

## 16.457 l4virtio\_net\_config\_t Struct Reference

Device configuration for network devices.

```
#include <virtio_net.h>
```

Collaboration diagram for l4virtio\_net\_config\_t:



### 16.457.1 Detailed Description

Device configuration for network devices.

Definition at line 34 of file [virtio\\_net.h](#).

The documentation for this struct was generated from the following file:

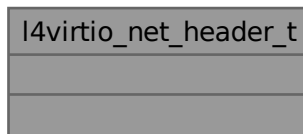
- l4/l4virtio/virtio\_net.h

## 16.458 l4virtio\_net\_header\_t Struct Reference

Header structure of a request for a network device.

```
#include <virtio_net.h>
```

Collaboration diagram for l4virtio\_net\_header\_t:



### 16.458.1 Detailed Description

Header structure of a request for a network device.

Definition at line 20 of file [virtio\\_net.h](#).

The documentation for this struct was generated from the following file:

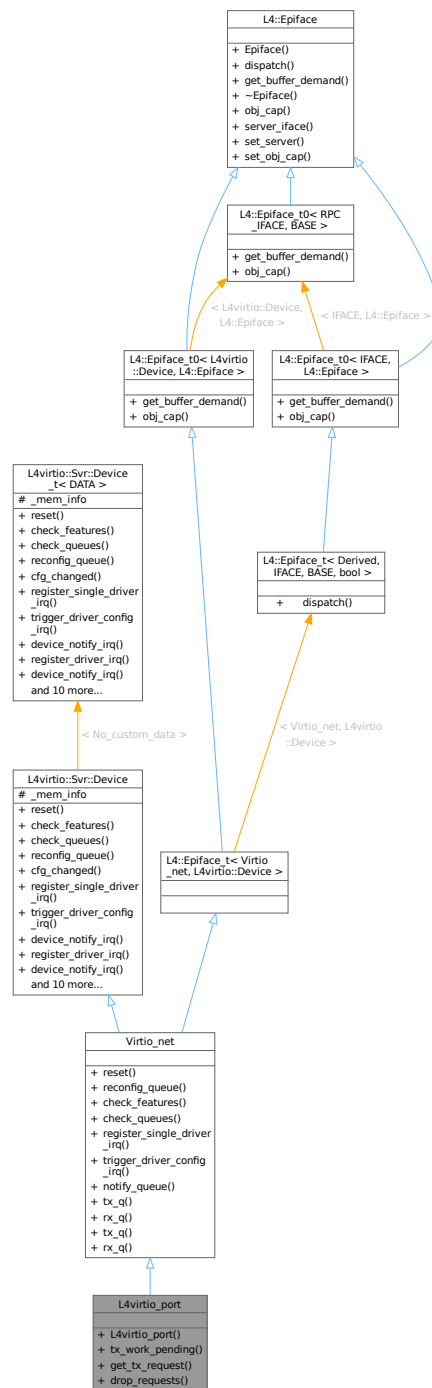
- l4/l4virtio/virtio\_net.h

## 16.459 L4virtio\_port Class Reference

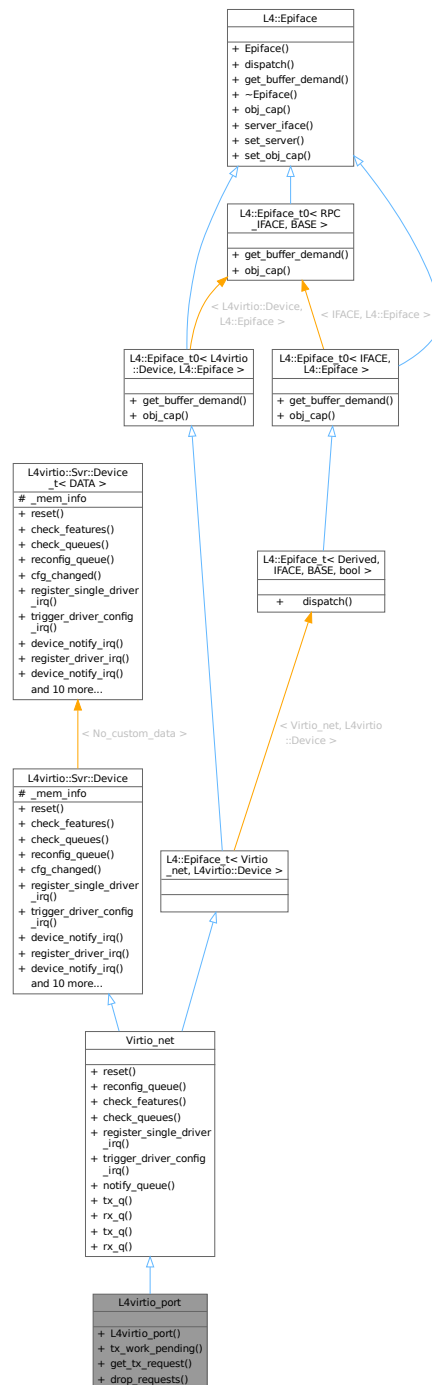
A Port on the Virtio Net Switch.

```
#include <port_l4virtio.h>
```

Inheritance diagram for L4virtio\_port:



Collaboration diagram for L4virtio\_port:



## Public Member Functions

- **L4virtio\_port** (unsigned vq\_max, unsigned num\_ds, char const \*name, [l4\\_uint8\\_t](#) const \*mac)  
*Create a Virtio net port object.*
- bool **tx\_work\_pending** () const  
*Check whether there is any work pending on the transmission queue.*
- std::optional< [Virtio\\_net\\_request](#) > **get\_tx\_request** ()



*Get one request from the transmission queue.*

- void **drop\_requests** ()

*Drop all requests pending in the transmission queue.*

## Public Member Functions inherited from Virtio\_net

- void **reset** () override  
*reset callback, called for doing a device reset*
- int **reconfig\_queue** (unsigned index) override  
*callback for client queue-config request*
- bool **check\_features** () override  
*callback for checking the subset of accepted features*
- bool **check\_queues** () override  
*Check whether both virtqueues are ready.*
- void **register\_single\_driver\_irq** () override  
*Save the \_kick\_guest\_irq that the client sent via device\_notification\_irq().*
- void **trigger\_driver\_config\_irq** () override  
*callback for triggering configuration change notification IRQ*
- void **notify\_queue** (L4virtio::Svr::Virtqueue \*queue)  
*Trigger the \_kick\_guest\_irq IRQ.*
- Virtqueue \* **tx\_q** ()  
*Getter for the transmission queue.*
- Virtqueue \* **rx\_q** ()  
*Getter for the receive queue.*
- Virtqueue const \* **tx\_q** () const  
*Getter for the transmission queue.*
- Virtqueue const \* **rx\_q** () const  
*Getter for the receive queue.*

## Public Member Functions inherited from L4virtio::Svr::Device\_t< No\_custom\_data >

- virtual void **cfg\_changed** (unsigned)  
*callback for client device configuration changes*
- virtual L4::Cap< L4::Irq > **device\_notify\_irq** () const  
*callback to gather the device notification IRQ (old-style)*
- virtual void **register\_driver\_irq** (unsigned idx)  
*Callback for registering an notification IRQ (multi IRQ).*
- virtual L4::Cap< L4::Irq > **device\_notify\_irq** (unsigned idx)  
*Callback to gather the device notification IRQ (multi IRQ).*
- virtual unsigned **num\_events\_supported** () const  
*Return the highest notification index supported.*
- **Device\_t** (Dev\_config \*dev\_config)  
*Make a device for the given config.*
- **Mem\_list** const \* **mem\_info** () const  
*Get the memory region list used for this device.*
- void **reset\_queue\_config** (unsigned idx, unsigned num\_max, bool inc\_generation=false)  
*Trigger reset for the configuration space for queue idx.*
- void **init\_mem\_info** (unsigned num)  
*Initialize the memory region list to the given maximum.*

- void `device_error` ()  
*Transition device into `DEVICE_NEEDS_RESET` state.*
- bool `setup_queue` (`Virtqueue` \*q, unsigned qn, unsigned num\_max)  
*Enable/disable the specified queue.*
- bool `handle_mem_cmd_write` ()  
*Check for a value in the `cmd` register and handle a write.*
- void `enable_trusted_ds_validation` ()  
*Enable trusted dataspace validation.*
- void `add_trusted_dataspaces` (std::shared\_ptr< Ds\_vector const > ds)  
*Provide a list of trusted dataspaces that can be used for validation.*

## Public Member Functions inherited from `L4::Epiface_t0< L4virtio::Device, L4::Epiface >`

- `Type_info::Demand` `get_buffer_demand` () const  
*Get the server-side buffer demand based in `IFACE`.*
- `Cap< L4virtio::Device >` `obj_cap` () const  
*Get the (typed) capability to this object.*

## Public Member Functions inherited from `L4::Epiface`

- `Epiface` ()  
*Make a server object.*
- virtual `~Epiface` ()=0  
*Destroy the object.*
- Stored\_cap `obj_cap` () const  
*Get the capability to the kernel object belonging to this object.*
- `Server_iface` \* `server_iface` () const  
*Get pointer to server interface at which the object is currently registered.*
- int `set_server` (`Server_iface` \*srv, `Cap< void >` cap, bool managed=false)  
*Set server registration info for the object.*
- void `set_obj_cap` (`Cap< void >` const &cap)  
*Deprecated server registration function.*

## Additional Inherited Members

## Public Types inherited from `L4::Epiface_t0< L4virtio::Device, L4::Epiface >`

- typedef `L4virtio::Device` `Interface`  
*Data type of the IPC interface definition.*

## Public Types inherited from `L4::Epiface`

- typedef `lpc_svr::Server_iface` `Server_iface`  
*Type for abstract server interface.*
- typedef `lpc_svr::Server_iface::Demand` `Demand`  
*Type for server-side receive buffer demand.*

## Protected Attributes inherited from [L4virtio::Svr::Device\\_t< No\\_custom\\_data >](#)

- [Mem\\_list\\_mem\\_info](#)  
*Memory region list.*

### 16.459.1 Detailed Description

A Port on the Virtio Net Switch.

A Port object gets created by `Virtio_factory::op_create()`. This function actually only instantiates objects of the types `Switch_port` and `Monitor_port`. The created Port registers itself at the switch's server. Usually, the IPC call for port creation comes from `ned`. To finalize the setup, the client has to initialize the port during the virtio initialization phase. To do this, the client registers a dataspace for queues and buffers and provides an IRQ to notify the client on incoming network requests.

Definition at line 36 of file [port\\_l4virtio.h](#).

### 16.459.2 Member Function Documentation

#### 16.459.2.1 drop\_requests()

```
void L4virtio_port::drop_requests () [inline]
```

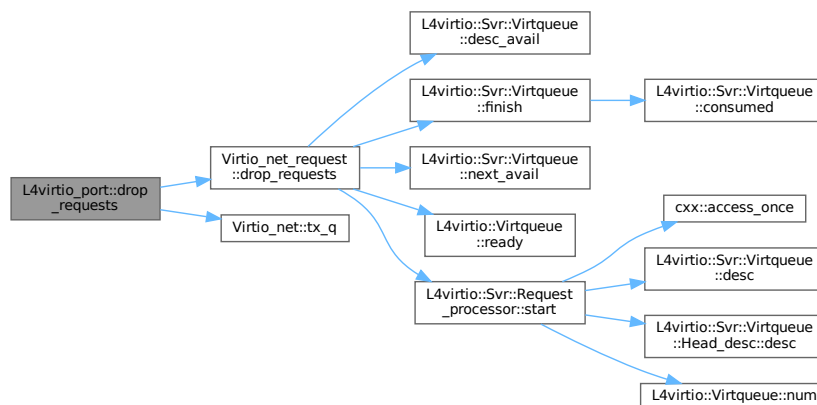
Drop all requests pending in the transmission queue.

This is used for monitor ports, which are not allowed to send packets.

Definition at line 103 of file [port\\_l4virtio.h](#).

References [Virtio\\_net\\_request::drop\\_requests\(\)](#), and [Virtio\\_net::tx\\_q\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

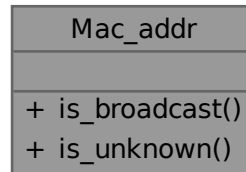
- `pkg/virtio-net-switch/server/switch/port_l4virtio.h`

## 16.460 Mac\_addr Class Reference

A wrapper class around the value of a MAC address.

```
#include <mac_addr.h>
```

Collaboration diagram for Mac\_addr:



### Public Member Functions

- bool **is\_broadcast** () const  
*Check if MAC address is a broadcast or multicast address.*
- bool **is\_unknown** () const  
*Check if the MAC address is not yet known.*

### 16.460.1 Detailed Description

A wrapper class around the value of a MAC address.

Definition at line 19 of file [mac\\_addr.h](#).

The documentation for this class was generated from the following file:

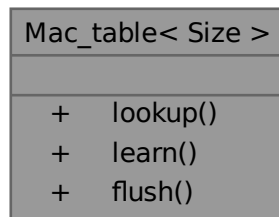
- [pkg/virtio-net-switch/server/switch/mac\\_addr.h](#)

## 16.461 Mac\_table< Size > Class Template Reference

[Mac\\_table](#) manages a 1:n association between ports and MAC addresses.

```
#include <mac_table.h>
```

Collaboration diagram for Mac\_table< Size >:



### Public Member Functions

- Port\_iface \* [lookup](#) (Mac\_addr dst, l4\_uint16\_t vlan\_id) const  
*Find the destination port for a MAC address and VLAN id.*
- void [learn](#) (Mac\_addr src, Port\_iface \*port, l4\_uint16\_t vlan\_id)  
*Learn a MAC address (add it to the MAC table).*
- void [flush](#) (Port\_iface \*port)  
*Flush all associations with a given port.*

## 16.461.1 Detailed Description

```
template<std::size_t Size = 1024U>
class Mac_table< Size >
```

[Mac\\_table](#) manages a 1:n association between ports and MAC addresses.

There are different types of devices which might be attached to a port. For a normal device the switch sees exactly one MAC address per port - the MAC address of the device attached to it. But there might be other devices like software bridges attached to the port sending packets with different MAC addresses to the port. Therefore the switch has to manage a 1:n association between ports and MAC addresses. The MAC table manages this association.

When a packet comes in we need to find the destination port for the packet and therefore perform a lookup based on the MAC address.

To prevent unbounded growth of the lookup table, the number of entries is limited. Replacement is done on a round-robin basis. If the capacity was reached, the oldest entry is evicted.

Definition at line 40 of file [mac\\_table.h](#).

## 16.461.2 Member Function Documentation

### 16.461.2.1 flush()

```
template<std::size_t Size = 1024U>
void Mac_table< Size >::flush (
 Port_iface * port) [inline]
```

Flush all associations with a given port.

### Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>port</i> | Pointer to port that is to be flushed |
|-------------|---------------------------------------|

This function removes all references to a given port from the MAC table. Since we manage a 1:n association between ports and MAC addresses there might be more than one entry for a given port and we have to iterate over the whole array to delete every reference to the port.

Definition at line 129 of file [mac\\_table.h](#).

### 16.461.2.2 learn()

```
template<std::size_t Size = 1024U>
void Mac_table< Size >::learn (
 Mac_addr src,
 Port_iface * port,
 14_uint16_t vlan_id) [inline]
```

Learn a MAC address (add it to the MAC table).

#### Parameters

|                |                                                                      |
|----------------|----------------------------------------------------------------------|
| <i>src</i>     | MAC address                                                          |
| <i>port</i>    | Pointer to the port object that can be used to reach MAC address src |
| <i>vlan_id</i> | VLAN id of the packet destination.                                   |

Will evict the oldest learned address from the table if the maximum capacity was reached and if the MAC address was not known yet. The source port of the table entry is always updated to cope with clients that move between ports.

Definition at line 78 of file [mac\\_table.h](#).

References [L4\\_UNLIKELY](#), and [lookup\(\)](#).

Here is the call graph for this function:



### 16.461.2.3 lookup()

```
template<std::size_t Size = 1024U>
Port_iface * Mac_table< Size >::lookup (
 Mac_addr dst,
 14_uint16_t vlan_id) const [inline]
```

Find the destination port for a MAC address and VLAN id.

#### Parameters

|                           |             |
|---------------------------|-------------|
| <i>dst</i>                | MAC address |
| <i>vlan</i><br><i>_id</i> | VLAN id     |

### Return values

|                |                                    |
|----------------|------------------------------------|
| <i>nullptr</i> | The MAC address is not known (yet) |
| <i>other</i>   | Pointer to the destination port    |

Definition at line 58 of file [mac\\_table.h](#).

Referenced by [learn\(\)](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

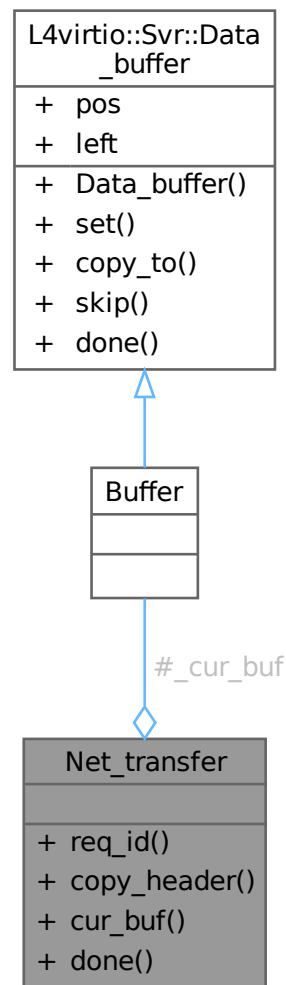
- `pkg/virtio-net-switch/server/switch/mac_table.h`

## 16.462 Net\_transfer Class Reference

A network request to only a single destination.

```
#include <request.h>
```

Collaboration diagram for Net\_transfer:



### Public Member Functions

- void const \* **req\_id** () const  
*Identifier for the underlying Net\_request, used for logging purposes.*
- virtual void **copy\_header** (Virtio\_net::Hdr \*dst\_header) const =0  
*Populate the virtio-net header for the destination.*
- **Buffer** & **cur\_buf** ()  
*Buffer containing (a part of) the packet data.*
- virtual bool **done** ()=0  
*Check whether the transfer has been completed, i.e.*

### 16.462.1 Detailed Description

A network request to only a single destination.



A `Net_request` can have multiple destinations (being a broadcast request, for example). That is why it is processed by multiple `Net_transfers`, each representing the delivery to a single destination port.

`Port_iface::handle_request` uses the `Net_transfer` to move one packet to the destination of the request.

Definition at line 33 of file [request.h](#).

## 16.462.2 Member Function Documentation

### 16.462.2.1 `cur_buf()`

```
Buffer & Net_transfer::cur_buf () [inline]
```

`Buffer` containing (a part of) the packet data.

Once emptied, a call to `done ()` might replenish the buffer, in case the net request consisted of multiple chained buffers.

Definition at line 54 of file [request.h](#).

### 16.462.2.2 `done()`

```
virtual bool Net_transfer::done () [pure virtual]
```

Check whether the transfer has been completed, i.e.

the entire packet data has been copied.

#### Return values

|              |                                                         |
|--------------|---------------------------------------------------------|
| <i>false</i> | There is remaining packet data that needs to be copied. |
| <i>true</i>  | The entire packet data has been copied.                 |

#### Exceptions

|                                               |                                     |
|-----------------------------------------------|-------------------------------------|
| <a href="#">L4virtio::Svr::Bad_descriptor</a> | Exception raised in SRC port queue. |
|-----------------------------------------------|-------------------------------------|

The documentation for this class was generated from the following file:

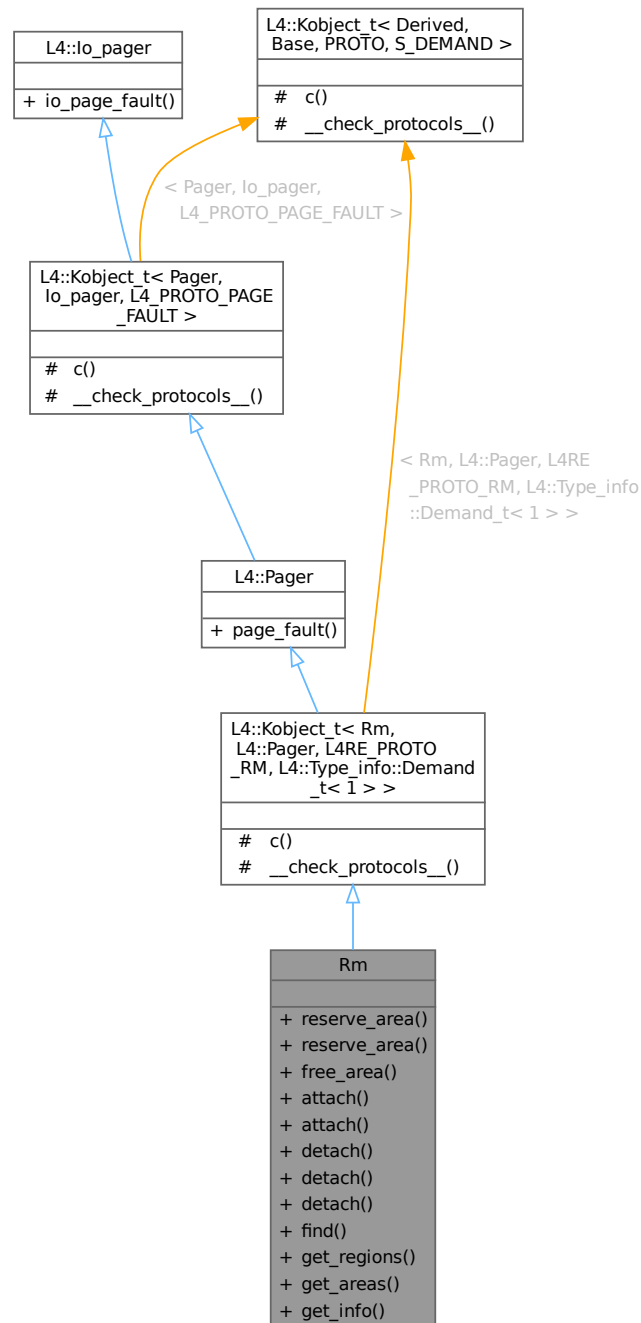
- `pkg/virtio-net-switch/server/switch/request.h`

## 16.463 Rm Class Reference

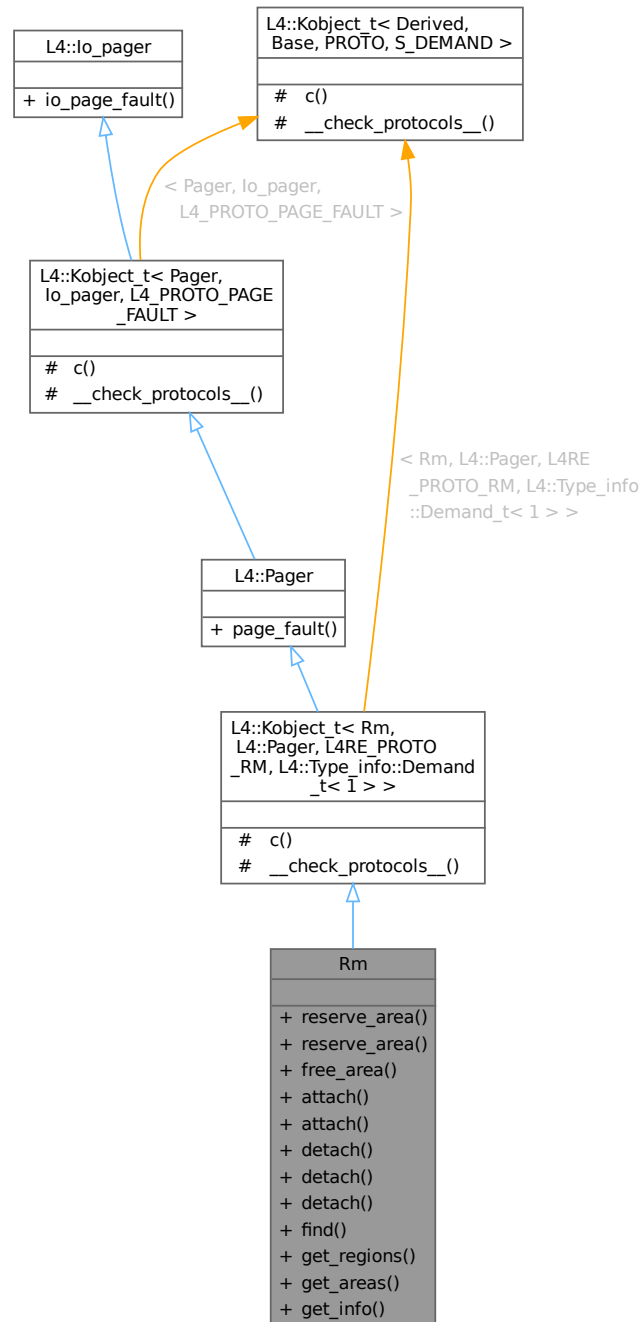
Region map.

```
#include <l4/re/rm>
```

Inheritance diagram for Rm:



Collaboration diagram for Rm:



## Data Structures

- struct [F](#)  
*Rm flags definitions.*
- class [Unique\\_region](#)  
*Unique region.*
- struct [Region](#)

*A region is a range of virtual addresses which is backed by content.*

- struct [Area](#)

*An area is a range of virtual addresses which is reserved, see [L4Re::Rm::reserve\\_area\(\)](#).*

## Public Types

- enum [Detach\\_result](#) { [Detached\\_ds](#) = 0 , [Kept\\_ds](#) = 1 , [Split\\_ds](#) = 2 , [Detach\\_result\\_mask](#) = 3 , [Detach\\_again](#) = 4 }
- Result values for detach operation.*
- enum [Region\\_flag\\_shifts](#) { [Caching\\_shift](#) = Dataspace::F::Caching\_shift }
- Region flag shifts.*
- enum [Detach\\_flags](#) { [Detach\\_exact](#) = 1 , [Detach\\_overlap](#) = 2 , [Detach\\_keep](#) = 4 }
- Flags for detach operation.*

## Public Member Functions

- long [reserve\\_area](#) ([l4\\_addr\\_t](#) \*start, unsigned long size, Flags flags=[Flags\(0\)](#), unsigned char align=[L4\\_PAGESHIFT](#)) const noexcept
- Reserve the given area in the region map.*
- template<typename T>  
long [reserve\\_area](#) (T \*\*start, unsigned long size, Flags flags=[Flags\(0\)](#), unsigned char align=[L4\\_PAGESHIFT](#)) const noexcept
- Reserve the given area in the region map.*
- long [free\\_area](#) ([l4\\_addr\\_t](#) addr)
- Free an area from the region map.*
- long [attach](#) ([l4\\_addr\\_t](#) \*start, unsigned long size, Flags flags, [L4::lpc::Cap](#)< Dataspace > mem, Offset offs=0, unsigned char align=[L4\\_PAGESHIFT](#), [L4::Cap](#)< [L4::Task](#) > const task=[L4::Cap](#)< [L4::Task](#) >::Invalid, char const \*name=nullptr, Offset backing\_offset=0) const noexcept
- Attach a data space to a region.*
- template<typename T>  
long [attach](#) (T \*\*start, unsigned long size, Flags flags, [L4::lpc::Cap](#)< Dataspace > mem, Offset offs=0, unsigned char align=[L4\\_PAGESHIFT](#), [L4::Cap](#)< [L4::Task](#) > const task=[L4::Cap](#)< [L4::Task](#) >::Invalid, char const \*name=nullptr, Offset backing\_offset=0) const noexcept
- Attach a data space to a region.*
- int [detach](#) ([l4\\_addr\\_t](#) addr, [L4::Cap](#)< Dataspace > \*mem, [L4::Cap](#)< [L4::Task](#) > const &task=This\_task) const noexcept
- Detach and unmap a region from the address space.*
- int [detach](#) (void \*addr, [L4::Cap](#)< Dataspace > \*mem, [L4::Cap](#)< [L4::Task](#) > const &task=This\_task) const noexcept
- Detach and unmap a region from the address space.*
- int [detach](#) ([l4\\_addr\\_t](#) start, unsigned long size, [L4::Cap](#)< Dataspace > \*mem, [L4::Cap](#)< [L4::Task](#) > const &task) const noexcept
- Detach and unmap all parts of the regions within the specified interval.*
- long [find](#) ([l4\\_addr\\_t](#) \*addr, unsigned long \*size, Offset \*offset, [L4Re::Rm::Flags](#) \*flags, [L4::Cap](#)< Dataspace > \*m) noexcept
- Find a region given an address and size.*
- long [get\\_regions](#) ([l4\\_addr\\_t](#) start, [L4::lpc::Ret\\_array](#)< [Region](#) > regions)
- Return the list of regions whose starting addresses are higher or equal to start in the address space managed by this region map.*
- long [get\\_areas](#) ([l4\\_addr\\_t](#) start, [L4::lpc::Ret\\_array](#)< [Area](#) > areas)
- Return the list of areas whose starting addresses are higher or equal to start in the address space managed by this region map.*
- long [get\\_info](#) ([l4\\_addr\\_t](#) addr, [L4::lpc::String](#)< char > &name, Offset &backing\_offset)
- Return auxiliary information of a region.*

## Public Member Functions inherited from [L4::Pager](#)

- [l4\\_msgtag\\_t page\\_fault](#) ([l4\\_umword\\_t](#) pfa, [l4\\_umword\\_t](#) pc, [L4::lpc::Rcv\\_fpage](#) rwin, [L4::lpc::Opt<L4::lpc::Snd\\_fpage & > fp](#))

*Page-fault protocol message.*

## Public Member Functions inherited from [L4::io\\_pager](#)

- [l4\\_msgtag\\_t io\\_page\\_fault](#) ([l4\\_fpage\\_t](#) io\_pfa, [l4\\_umword\\_t](#) pc, [L4::lpc::Rcv\\_fpage](#) rwin, [L4::lpc::Opt<L4::lpc::Snd\\_fpage & > fp](#))

*IO page fault protocol message.*

## Additional Inherited Members

### Protected Types inherited from

[L4::Kobject\\_t< Rm, L4::Pager, L4RE\\_PROTO\\_RM, L4::Type\\_info::Demand\\_t< 1 > >](#)

- typedef [Rm](#) **Class**  
*The target interface type (inheriting from [Kobject\\_t](#)).*
- typedef Typeid::Iface< PROTO, [Rm](#) > **\_\_Iface**  
*The interface description for the derived class.*
- typedef Typeid::Merge\_list< Typeid::Iface\_list< **\_\_Iface** >, typename [L4::Pager::\\_\\_Iface\\_list](#) > **\_\_Iface\_list**  
*The list of all RPC interfaces provided directly or through inheritance.*

### Protected Types inherited from

[L4::Kobject\\_t< Pager, io\\_pager, L4\\_PROTO\\_PAGE\\_FAULT >](#)

- typedef [Pager](#) **Class**  
*The target interface type (inheriting from [Kobject\\_t](#)).*
- typedef Typeid::Iface< PROTO, [Pager](#) > **\_\_Iface**  
*The interface description for the derived class.*
- typedef Typeid::Merge\_list< Typeid::Iface\_list< **\_\_Iface** >, typename [io\\_pager::\\_\\_Iface\\_list](#) > **\_\_Iface\_list**  
*The list of all RPC interfaces provided directly or through inheritance.*

### Protected Member Functions inherited from

[L4::Kobject\\_t< Rm, L4::Pager, L4RE\\_PROTO\\_RM, L4::Type\\_info::Demand\\_t< 1 > >](#)

- [L4::Cap< Class > c](#) () const noexcept  
*Get the capability to ourselves.*

### Protected Member Functions inherited from

[L4::Kobject\\_t< Pager, io\\_pager, L4\\_PROTO\\_PAGE\\_FAULT >](#)

- [L4::Cap< Class > c](#) () const noexcept  
*Get the capability to ourselves.*

**Static Protected Member Functions inherited from****[L4::Kobject\\_t< Rm, L4::Pager, L4RE\\_PROTO\\_RM, L4::Type\\_info::Demand\\_t< 1 > >](#)**

- static void **\_\_check\_protocols\_\_** () noexcept

*Helper to check for protocol conflicts.***Static Protected Member Functions inherited from****[L4::Kobject\\_t< Pager, lo\\_pager, L4\\_PROTO\\_PAGE\\_FAULT >](#)**

- static void **\_\_check\_protocols\_\_** () noexcept

*Helper to check for protocol conflicts.***16.463.1 Detailed Description**[Region](#) map.

See also

[Region map API](#) .Definition at line [81](#) of file [rm](#).**16.463.2 Member Enumeration Documentation****16.463.2.1 Detach\_flags**enum [L4Re::Rm::Detach\\_flags](#)

Flags for detach operation.

**Enumerator**

|                                |                                                                                  |
|--------------------------------|----------------------------------------------------------------------------------|
| <a href="#">Detach_exact</a>   | Do an unmap of the exact region given.                                           |
| <a href="#">Detach_overlap</a> | Do an unmap of all overlapping regions.                                          |
| <a href="#">Detach_keep</a>    | Do not free the detached data space, ignore the <a href="#">F::Detach_free</a> . |

Definition at line [221](#) of file [rm](#).**16.463.2.2 Detach\_result**enum [L4Re::Rm::Detach\\_result](#)

Result values for detach operation.

**Enumerator**

|              |                                  |
|--------------|----------------------------------|
| Detached_ds  | Detached data sapce.             |
| Kept_ds      | Kept data space.                 |
| Split_ds     | Splitted data space, and done.   |
| Detach_again | Detached data space, more to do. |

Definition at line 89 of file [rm](#).

### 16.463.2.3 Region\_flag\_shifts

```
enum L4Re::Rm::Region_flag_shifts
```

[Region](#) flag shifts.

#### Enumerator

|               |                                         |
|---------------|-----------------------------------------|
| Caching_shift | Start of <a href="#">Rm</a> cache bits. |
|---------------|-----------------------------------------|

Definition at line 101 of file [rm](#).

## 16.463.3 Member Function Documentation

### 16.463.3.1 attach() [1/2]

```
long L4Re::Rm::attach (
 l4_addr_t * start,
 unsigned long size,
 Rm::Flags flags,
 L4::Ipc::Cap< Dataspace > mem,
 Rm::Offset offs = 0,
 unsigned char align = L4_PAGESHIFT,
 L4::Cap< L4::Task > const task = L4::Cap<L4::Task>::Invalid,
 char const * name = nullptr,
 Rm::Offset backing_offset = 0) const [noexcept]
```

Attach a data space to a region.

#### Parameters

|         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in, out | <i>start</i> | Virtual start address where the region manager shall attach the data space. Will be rounded down to the nearest start of a page. If <a href="#">L4Re::Rm::F::Search_addr</a> is given this value is used as the start address to search for a free virtual memory region and the resulting address is returned here. If <a href="#">L4Re::Rm::F::In_area</a> is given the value is used as a selector for the area (see <a href="#">L4Re::Rm::reserve_area</a> ) to attach the data space to. |
|         | <i>size</i>  | Size of the data space to attach (in bytes). Will be rounded up to the nearest multiple of the page size.                                                                                                                                                                                                                                                                                                                                                                                     |

|  |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <i>flags</i>          | The flags control how and with which rights the dataspace is attached to the region. See <a href="#">L4Re::Rm::F::Attach_flags</a> and <a href="#">L4Re::Rm::F::Region_flags</a> . The caller must specify the desired rights of the attached region explicitly. The default set of rights is empty. If the <a href="#">F::Eager_map</a> flag is set this function may also return <a href="#">L4Re::Dataspace::map</a> error codes if the mapping fails. |
|  | <i>mem</i>            | Data space.                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|  | <i>offs</i>           | Offset into the data space to use.                                                                                                                                                                                                                                                                                                                                                                                                                        |
|  | <i>align</i>          | Alignment of the virtual region, log2-size, default: a page ( <a href="#">L4_PAGESHIFT</a> ). This is only meaningful if the <a href="#">L4Re::Rm::F::Search_addr</a> flag is used.                                                                                                                                                                                                                                                                       |
|  | <i>task</i>           | Optional destination task of mapping if <a href="#">F::Eager_map</a> flag was passed. If invalid, the mapping is established in the current task. This parameter is only useful if the region manager is for a foreign task.                                                                                                                                                                                                                              |
|  | <i>name</i>           | Optional name of the region.                                                                                                                                                                                                                                                                                                                                                                                                                              |
|  | <i>backing_offset</i> | Optional value describing an offset into the backing store of this region.                                                                                                                                                                                                                                                                                                                                                                                |

### Return values

|                   |                                                                    |
|-------------------|--------------------------------------------------------------------|
| 0                 | Success                                                            |
| -L4_ENOENT        | No area could be found (see <a href="#">L4Re::Rm::F::In_area</a> ) |
| -L4_EPERM         | Operation not allowed.                                             |
| -L4_EINVAL        |                                                                    |
| -L4_EADDRNOTAVAIL | The given address is not available.                                |
| <0                | IPC errors                                                         |

Makes the whole or parts of a data space visible in the virtual memory of the corresponding task. The corresponding region in the virtual address space is backed with the contents of the dataspace.

### Note

When searching for a free place in the virtual address space, the space between *start* and the end of the virtual address space is searched.

There is no region object created, instead the region is defined by a virtual address within this range (see [L4Re::Rm::find](#)).

Definition at line 34 of file [rm\\_impl.h](#).

### 16.463.3.2 attach() [2/2]

```
template<typename T>
long L4Re::Rm::attach (
 T ** start,
 unsigned long size,
 Flags flags,
 L4::Ipc::Cap< Dataspace > mem,
 Offset offs = 0,
 unsigned char align = L4_PAGESHIFT,
 L4::Cap< L4::Task > const task = L4::Cap<L4::Task>::Invalid,
```



```
char const * name = nullptr,
Offset backing_offset = 0) const [inline], [noexcept]
```

Attach a data space to a region.

#### Parameters

---

|                |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>in, out</i> | <i>start</i>          | Virtual start address where the region manager shall attach the data space. Will be rounded down to the nearest start of a page. If <a href="#">L4Re::Rm::F::Search_addr</a> is given this value is used as the start address to search for a free virtual memory region and the resulting address is returned here. If <a href="#">L4Re::Rm::F::In_area</a> is given the value is used as a selector for the area (see <a href="#">L4Re::Rm::reserve_area</a> ) to attach the data space to. |
|                | <i>size</i>           | Size of the data space to attach (in bytes). Will be rounded up to the nearest multiple of the page size.                                                                                                                                                                                                                                                                                                                                                                                     |
|                | <i>flags</i>          | The flags control how and with which rights the dataspace is attached to the region. See <a href="#">L4Re::Rm::F::Attach_flags</a> and <a href="#">L4Re::Rm::F::Region_flags</a> . The caller must specify the desired rights of the attached region explicitly. The default set of rights is empty. If the <a href="#">F::Eager_map</a> flag is set this function may also return <a href="#">L4Re::Dataspace::map</a> error codes if the mapping fails.                                     |
|                | <i>mem</i>            | Data space.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|                | <i>offs</i>           | Offset into the data space to use.                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|                | <i>align</i>          | Alignment of the virtual region, log2-size, default: a page ( <a href="#">L4_PAGESHIFT</a> ). This is only meaningful if the <a href="#">L4Re::Rm::F::Search_addr</a> flag is used.                                                                                                                                                                                                                                                                                                           |
|                | <i>task</i>           | Optional destination task of mapping if <a href="#">F::Eager_map</a> flag was passed. If invalid, the mapping is established in the current task. This parameter is only useful if the region manager is for a foreign task.                                                                                                                                                                                                                                                                  |
|                | <i>name</i>           | Optional name of the region.                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|                | <i>backing_offset</i> | Optional value describing an offset into the backing store of this region.                                                                                                                                                                                                                                                                                                                                                                                                                    |

### Return values

|                   |                                                                    |
|-------------------|--------------------------------------------------------------------|
| 0                 | Success                                                            |
| -L4_ENOENT        | No area could be found (see <a href="#">L4Re::Rm::F::In_area</a> ) |
| -L4_EPERM         | Operation not allowed.                                             |
| -L4_EINVAL        |                                                                    |
| -L4_EADDRNOTAVAIL | The given address is not available.                                |
| <0                | IPC errors                                                         |

Makes the whole or parts of a data space visible in the virtual memory of the corresponding task. The corresponding region in the virtual address space is backed with the contents of the dataspace.

### Note

When searching for a free place in the virtual address space, the space between *start* and the end of the virtual address space is searched.

There is no region object created, instead the region is defined by a virtual address within this range (see [L4Re::Rm::find](#)).

Definition at line 409 of file [rm](#).

### 16.463.3.3 detach() [1/3]

```
int L4Re::Rm::detach (
 l4_addr_t addr,
```

```

L4::Cap< Dataspace > * mem,
L4::Cap< L4::Task > const & task = This_task) const [inline], [noexcept]

```

Detach and unmap a region from the address space.

### Parameters

|     |             |                                                                                                                                                                    |
|-----|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <i>addr</i> | Virtual address of region, any address within the region is valid.                                                                                                 |
| out | <i>mem</i>  | Dataspace that is affected. Give 0 if not interested.                                                                                                              |
|     | <i>task</i> | This argument specifies the task where the pages are unmapped. Provide <a href="#">L4::Cap&lt;L4::Task&gt;::Invalid</a> for none. The default is the current task. |

### Return values

|                                         |                  |
|-----------------------------------------|------------------|
| <a href="#">L4Re::Rm::Detach_result</a> | On success.      |
| <code>-L4_ENOENT</code>                 | No region found. |
| <code>&lt;0</code>                      | IPC errors       |

Frees a region in the virtual address space given by *addr* (address type). The corresponding part of the address space is now available again.

Definition at line 765 of file [rm](#).

#### 16.463.3.4 detach() [2/3]

```

int L4Re::Rm::detach (
 l4_addr_t start,
 unsigned long size,
 L4::Cap< Dataspace > * mem,
 L4::Cap< L4::Task > const & task) const [inline], [noexcept]

```

Detach and unmap all parts of the regions within the specified interval.

### Parameters

|     |              |                                                                                                                                                                    |
|-----|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <i>start</i> | Start of area to detach, must be within region.                                                                                                                    |
|     | <i>size</i>  | Size of of area to detach (in bytes).                                                                                                                              |
| out | <i>mem</i>   | Dataspace that is affected. Give 0 if not interested.                                                                                                              |
|     | <i>task</i>  | This argument specifies the task where the pages are unmapped. Provide <a href="#">L4::Cap&lt;L4::Task&gt;::Invalid</a> for none. The default is the current task. |

### Return values

|                                         |                  |
|-----------------------------------------|------------------|
| <a href="#">L4Re::Rm::Detach_result</a> | On success.      |
| <code>-L4_ENOENT</code>                 | No region found. |
| <code>&lt;0</code>                      | IPC errors       |

Frees all regions within the interval given by start and size. If a region overlaps the start or the end of the interval this region is only detached partly. If the interval is within one region the original region is split up into two separate regions.

Definition at line 778 of file [rm](#).

### 16.463.3.5 detach() [3/3]

```
int L4Re::Rm::detach (
 void * addr,
 L4::Cap< Dataspace > * mem,
 L4::Cap< L4::Task > const & task = This_task) const [inline], [noexcept]
```

Detach and unmap a region from the address space.

#### Parameters

|     |             |                                                                                                                                                                    |
|-----|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <i>addr</i> | Virtual address of region, any address within the region is valid.                                                                                                 |
| out | <i>mem</i>  | Dataspace that is affected. Give 0 if not interested.                                                                                                              |
|     | <i>task</i> | This argument specifies the task where the pages are unmapped. Provide <a href="#">L4::Cap&lt;L4::Task&gt;::Invalid</a> for none. The default is the current task. |

#### Return values

|                                         |                  |
|-----------------------------------------|------------------|
| <a href="#">L4Re::Rm::Detach_result</a> | On success.      |
| <a href="#">-L4_ENOENT</a>              | No region found. |
| <a href="#">&lt;0</a>                   | IPC errors       |

Frees a region in the virtual address space given by addr (address type). The corresponding part of the address space is now available again.

Definition at line 770 of file [rm](#).

### 16.463.3.6 find()

```
long L4Re::Rm::find (
 l4_addr_t * addr,
 unsigned long * size,
 Offset * offset,
 L4Re::Rm::Flags * flags,
 L4::Cap< Dataspace > * m) [inline], [noexcept]
```

Find a region given an address and size.

#### Parameters

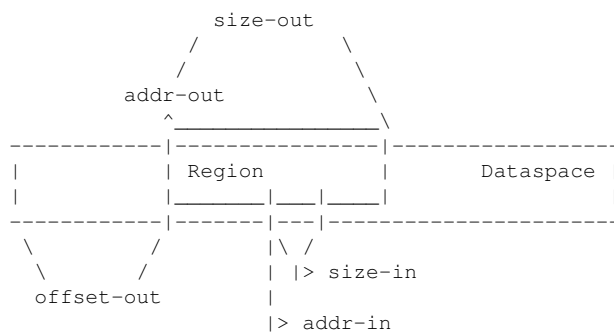
|         |             |                                                                     |
|---------|-------------|---------------------------------------------------------------------|
| in, out | <i>addr</i> | Address to look for. Returns the start address of the found region. |
|---------|-------------|---------------------------------------------------------------------|

|         |               |                                                                                                       |
|---------|---------------|-------------------------------------------------------------------------------------------------------|
| in, out | <i>size</i>   | Size of the area to look for (in bytes). Returns the size of the found region (in bytes).             |
| out     | <i>offset</i> | Offset at the beginning of the region within the associated dataspace.                                |
| out     | <i>flags</i>  | <a href="#">Region flags</a> , see <a href="#">F::Region_flags</a> (and <a href="#">F::In_area</a> ). |
| out     | <i>m</i>      | Associated dataspace or paging service.                                                               |

### Return values

|            |                        |
|------------|------------------------|
| 0          | Success                |
| -L4_EPERM  | Operation not allowed. |
| -L4_ENOENT | No region found.       |
| <0         | IPC errors             |

This function returns the properties of the region that contains the area described by the `addr` and `size` parameter. If no such region is found but a reserved area, the area is returned and [F::In\\_area](#) is set in `flags`. Note, in the case of an area the `offset` and `m` return values are invalid.



### Note

The value of the `size` input parameter should be 1 to assure that a region can be determined unambiguously.

Definition at line [672](#) of file [rm](#).

### 16.463.3.7 free\_area()

```
long L4Re::Rm::free_area (
 l4_addr_t addr)
```

Free an area from the region map.

### Parameters

|             |                                     |
|-------------|-------------------------------------|
| <i>addr</i> | An address within the area to free. |
|-------------|-------------------------------------|

### Return values

|            |                |
|------------|----------------|
| 0          | Success        |
| -L4_ENOENT | No area found. |
| <0         | IPC errors     |

**Note**

The data spaces that are attached to that area are not detached by this operation.

**See also**

[reserve\\_area\(\)](#) for more information about areas.

**16.463.3.8 get\_areas()**

```
long L4Re::Rm::get_areas (
 l4_addr_t start,
 L4::Ipc::Ret_array< Area > areas)
```

Return the list of areas whose starting addresses are higher or equal to `start` in the address space managed by this region map.

**Parameters**

|     |              |                                                |
|-----|--------------|------------------------------------------------|
|     | <i>start</i> | Virtual address from where to start searching. |
| out | <i>areas</i> | List of areas found in this region map.        |

**Return values**

|          |                                                           |
|----------|-----------------------------------------------------------|
| $\geq 0$ | Number of returned areas in the <code>areas</code> array. |
| <0       | IPC errors                                                |

**Note**

The returned list of areas might not be complete and the caller shall use the function repeatedly with a start address one larger than the end address of the last area from the previous call.

**16.463.3.9 get\_info()**

```
long L4Re::Rm::get_info (
 l4_addr_t addr,
 L4::Ipc::String< char > & name,
 Offset & backing_offset)
```

Return auxiliary information of a region.

This is a debugging feature and might not be available.

**Parameters**

|     |                       |                                |
|-----|-----------------------|--------------------------------|
|     | <i>addr</i>           | Virtual address of the region. |
| out | <i>name</i>           | Name of the region.            |
| out | <i>backing_offset</i> | Backing offset information.    |

### Return values

|            |                         |
|------------|-------------------------|
| 0          | Success                 |
| -L4_ENOENT | Region not found.       |
| -L4_ENOSYS | Function not available. |
| <0         | IPC errors              |

#### 16.463.3.10 get\_regions()

```
long L4Re::Rm::get_regions (
 l4_addr_t start,
 L4::Ipc::Ret_array< Region > regions)
```

Return the list of regions whose starting addresses are higher or equal to *start* in the address space managed by this region map.

### Parameters

|     |                |                                                |
|-----|----------------|------------------------------------------------|
|     | <i>start</i>   | Virtual address from where to start searching. |
| out | <i>regions</i> | List of regions found in this region map.      |

### Return values

|     |                                                         |
|-----|---------------------------------------------------------|
| >=0 | Number of returned regions in the <i>regions</i> array. |
| <0  | IPC errors                                              |

### Note

The returned list of regions might not be complete and the caller shall use the function repeatedly with a start address one larger than the end address of the last region from the previous call.

#### 16.463.3.11 reserve\_area() [1/2]

```
long L4Re::Rm::reserve_area (
 l4_addr_t * start,
 unsigned long size,
 Flags flags = Flags(0),
 unsigned char align = L4_PAGESHIFT) const [inline], [noexcept]
```

Reserve the given area in the region map.

### Parameters

|                |              |                                                                                                                             |
|----------------|--------------|-----------------------------------------------------------------------------------------------------------------------------|
| <i>in, out</i> | <i>start</i> | The virtual start address of the area to reserve. Returns the start address of the area.                                    |
|                | <i>size</i>  | The size of the area to reserve (in bytes).                                                                                 |
|                | <i>flags</i> | Flags for the reserved area (see <a href="#">L4Re::Rm::F::Region_flags</a> and <a href="#">L4Re::Rm::F::Attach_flags</a> ). |
|                | <i>align</i> | Alignment of area if searched as bits (log2 value).                                                                         |

### Return values

|                   |                                    |
|-------------------|------------------------------------|
| 0                 | Success                            |
| -L4_EADDRNOTAVAIL | The given area cannot be reserved. |
| <0                | IPC errors                         |

This function reserves an area within the virtual address space managed by the region map. There are two kinds of areas available:

- Reserved areas (*flags* = [L4Re::Rm::F::Reserved](#)), where no data spaces can be attached
- Special purpose areas (*flags* = 0), where data spaces can be attached to the area via the [L4Re::Rm::F::In\\_area](#) flag and a start address within the area itself.

### Note

When searching for a free place in the virtual address space (with *flags* = [L4Re::Rm::F::Search\\_addr](#)), the space between *start* and the end of the virtual address space is searched.

Definition at line 280 of file [rm](#).

### 16.463.3.12 reserve\_area() [2/2]

```
template<typename T>
long L4Re::Rm::reserve_area (
 T ** start,
 unsigned long size,
 Flags flags = Flags(0),
 unsigned char align = L4_PAGESHIFT) const [inline], [noexcept]
```

Reserve the given area in the region map.

### Parameters

|                |              |                                                                                                         |
|----------------|--------------|---------------------------------------------------------------------------------------------------------|
| <i>in, out</i> | <i>start</i> | The virtual start address of the area to reserve. Returns the start address of the area.                |
|                | <i>size</i>  | The size of the area to reserve (in bytes).                                                             |
|                | <i>flags</i> | Flags for the reserved area (see <a href="#">F::Region_flags</a> and <a href="#">F::Attach_flags</a> ). |
|                | <i>align</i> | Alignment of area if searched as bits (log2 value).                                                     |

### Return values



|                   |                                    |
|-------------------|------------------------------------|
| 0                 | Success                            |
| -L4_EADDRNOTAVAIL | The given area cannot be reserved. |
| <0                | IPC errors                         |

For more information, please refer to the analogous function

See also

[L4Re::Rm::reserve\\_area](#).

Definition at line 306 of file [rm](#).

The documentation for this class was generated from the following files:

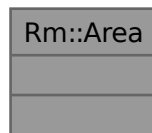
- [l4/re/rm](#)
- [l4/re/impl/rm\\_impl.h](#)

## 16.464 Rm::Area Struct Reference

An area is a range of virtual addresses which is reserved, see [L4Re::Rm::reserve\\_area\(\)](#).

```
#include <rm>
```

Collaboration diagram for Rm::Area:



### 16.464.1 Detailed Description

An area is a range of virtual addresses which is reserved, see [L4Re::Rm::reserve\\_area\(\)](#).

See also

[Region map API](#)

Definition at line 699 of file [rm](#).

The documentation for this struct was generated from the following file:

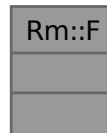
- [l4/re/rm](#)

## 16.465 Rm::F Struct Reference

Rm flags definitions.

```
#include <rm>
```

Collaboration diagram for Rm::F:



### Public Types

- enum [Attach\\_flags](#) : l4\_uint32\_t {  
[Search\\_addr](#) = 0x20000 , [In\\_area](#) = 0x40000 , [Eager\\_map](#) = 0x80000 , [No\\_eager\\_map](#) = 0x100000 ,  
[Attach\\_mask](#) = 0x1f0000 }  
*Flags for attach operation.*
- enum [Region\\_flags](#) : l4\_uint16\_t {  
[Rights\\_mask](#) = 0x0f , [R](#) = Dataspace::F::R , [W](#) = Dataspace::F::W , [X](#) = Dataspace::F::X ,  
[RW](#) = Dataspace::F::RW , [RX](#) = Dataspace::F::RX , [RWX](#) = Dataspace::F::RWX , [Kernel](#) = 0x100 ,  
[Detach\\_free](#) = 0x200 , [Pager](#) = 0x400 , [Reserved](#) = 0x800 , [Caching\\_mask](#) = Dataspace::F::Caching\_mask ,  
[Cache\\_normal](#) = Dataspace::F::Normal , [Cache\\_buffered](#) = Dataspace::F::Bufferable , [Cache\\_uncached](#) =  
Dataspace::F::Uncacheable , [Ds\\_map\\_mask](#) = 0xff ,  
[Region\\_flags\\_mask](#) = 0xffff }  
*Region flags (permissions, cacheability, special).*

### 16.465.1 Detailed Description

Rm flags definitions.

Definition at line 108 of file [rm](#).

### 16.465.2 Member Enumeration Documentation

#### 16.465.2.1 Attach\_flags

```
enum L4Re::Rm::F::Attach_flags : l4_uint32_t
```

Flags for attach operation.

#### Enumerator

|              |                                                   |
|--------------|---------------------------------------------------|
| Search_addr  | Search for a suitable address range.              |
| In_area      | Search only in area, or map into area.            |
| Eager_map    | Eagerly map the attached data space in.           |
| No_eager_map | Prevent eager mapping of the attached data space. |
| Attach_mask  | Mask of all attach flags.                         |

Definition at line 111 of file [rm](#).

### 16.465.2.2 Region\_flags

```
enum L4Re::Rm::F::Region_flags : l4_uint16_t
```

[Region](#) flags (permissions, cacheability, special).

#### Enumerator

|                   |                                                                                                           |
|-------------------|-----------------------------------------------------------------------------------------------------------|
| Rights_mask       | <a href="#">Region</a> rights.                                                                            |
| R                 | Readable region.                                                                                          |
| W                 | Writable region.                                                                                          |
| X                 | Executable region.                                                                                        |
| RW                | Readable and writable region.                                                                             |
| RX                | Readable and executable region.                                                                           |
| RWX               | Readable, writable and executable region.                                                                 |
| Kernel            | Kernel-provided memory (KUMEM).                                                                           |
| Detach_free       | Free the portion of the data space after detach.                                                          |
| Pager             | <a href="#">Region</a> has a pager.                                                                       |
| Reserved          | <a href="#">Region</a> is reserved (blocked).                                                             |
| Caching_mask      | Mask of all <a href="#">Rm</a> cache bits.                                                                |
| Cache_normal      | Cache bits for normal cacheable memory. This is the default if no other cache-related flag was specified. |
| Cache_buffered    | Cache bits for buffered (write combining) memory.                                                         |
| Cache_uncached    | Cache bits for uncached memory.                                                                           |
| Ds_map_mask       | Mask for all bits for cache options and rights.                                                           |
| Region_flags_mask | Mask of all region flags.                                                                                 |

Definition at line 128 of file [rm](#).

The documentation for this struct was generated from the following file:

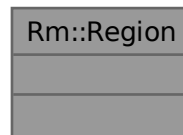
- [l4/re/rm](#)

## 16.466 Rm::Region Struct Reference

A region is a range of virtual addresses which is backed by content.

```
#include <rm>
```

Collaboration diagram for Rm::Region:



### 16.466.1 Detailed Description

A region is a range of virtual addresses which is backed by content.

See also

[Region map API](#)

Definition at line 686 of file [rm](#).

The documentation for this struct was generated from the following file:

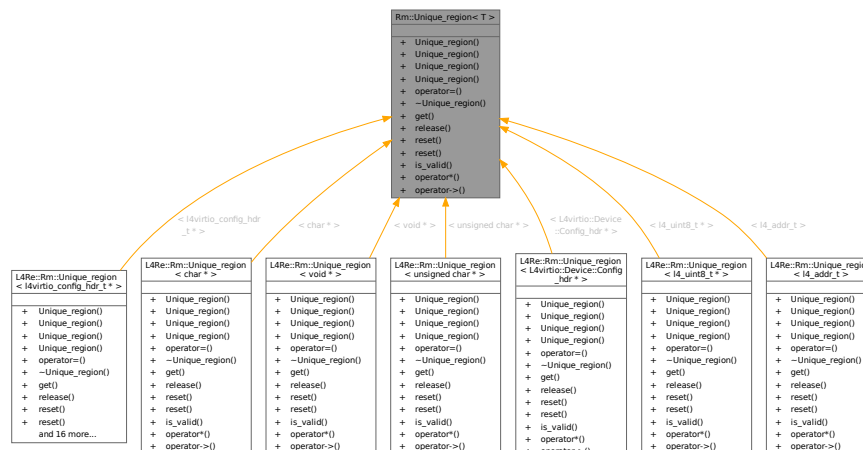
- [l4/re/rm](#)

## 16.467 Rm::Unique\_region< T > Class Template Reference

Unique region.

```
#include <rm>
```

Inheritance diagram for Rm::Unique\_region< T >:



Collaboration diagram for Rm::Unique\_region< T >:

| Rm::Unique_region< T >                                                                                                                                                                                                                                                                                                                                      |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>+ Unique_region()</li> <li>+ Unique_region()</li> <li>+ Unique_region()</li> <li>+ Unique_region()</li> <li>+ operator=()</li> <li>+ ~Unique_region()</li> <li>+ get()</li> <li>+ release()</li> <li>+ reset()</li> <li>+ reset()</li> <li>+ is_valid()</li> <li>+ operator*()</li> <li>+ operator-&gt;()</li> </ul> |

## Public Member Functions

- **Unique\_region** () noexcept  
*Construct an invalid [Unique\\_region](#).*
- **Unique\_region** (T addr) noexcept  
*Construct a [Unique\\_region](#) from an address.*
- **Unique\_region** (T addr, L4::Cap< Rm > const &rm) noexcept  
*Construct a valid [Unique\\_region](#) from an address and a region manager.*
- **Unique\_region** (Unique\_region &&o) noexcept  
*Move-Construct a [Unique\\_region](#).*
- **Unique\_region** & operator= (Unique\_region &&o) noexcept  
*Move-assign a [Unique\\_region](#).*
- **~Unique\_region** () noexcept  
*Destructor.*
- T **get** () const noexcept  
*Return the address.*
- T **release** () noexcept  
*Return the address and invalidate the [Unique\\_region](#).*
- void **reset** (T addr, L4::Cap< Rm > const &rm) noexcept  
*Set new address and region manager.*
- void **reset** () noexcept  
*Make the [Unique\\_region](#) invalid.*
- bool **is\_valid** () const noexcept  
*Check if the [Unique\\_region](#) is valid.*

- **T operator\* ()** const noexcept  
*Dereference the address.*
- **T operator-> ()** const noexcept  
*Member access for the address.*

### 16.467.1 Detailed Description

```
template<typename T>
class Rm::Unique_region< T >
```

Unique region.

Capture a single region with automatic detach on destruction and unique ownership. Stores the start address and the region-mapper capability internally. A unique region is valid precisely if the internal region-mapper capability is valid. The features for unique ownership and automatic detach are only active for valid unique regions.

Definition at line 435 of file [rm](#).

### 16.467.2 Constructor & Destructor Documentation

#### 16.467.2.1 Unique\_region() [1/3]

```
template<typename T>
L4Re::Rm::Unique_region< T >::Unique_region (
 T addr) [inline], [explicit], [noexcept]
```

Construct a [Unique\\_region](#) from an address.

No region manager is set.

#### Parameters

|             |                 |
|-------------|-----------------|
| <i>addr</i> | The new address |
|-------------|-----------------|

Definition at line 456 of file [rm](#).

#### 16.467.2.2 Unique\_region() [2/3]

```
template<typename T>
L4Re::Rm::Unique_region< T >::Unique_region (
 T addr,
 L4::Cap< Rm > const & rm) [inline], [noexcept]
```

Construct a valid [Unique\\_region](#) from an address and a region manager.

#### Parameters

|             |                    |
|-------------|--------------------|
| <i>addr</i> | The address        |
| <i>rm</i>   | The region manager |

Definition at line 465 of file [rm](#).

### 16.467.2.3 Unique\_region() [3/3]

```
template<typename T>
L4Re::Rm::Unique_region< T >::Unique_region (
 Unique_region< T > && o) [inline], [noexcept]
```

Move-Construct a [Unique\\_region](#).

#### Parameters

|          |                                    |
|----------|------------------------------------|
| <i>o</i> | L-value reference to other region. |
|----------|------------------------------------|

Definition at line 473 of file [rm](#).

### 16.467.2.4 ~Unique\_region()

```
template<typename T>
L4Re::Rm::Unique_region< T >::~~Unique_region () [inline], [noexcept]
```

Destructor.

If the region is valid, call [detach](#).

Definition at line 498 of file [rm](#).

## 16.467.3 Member Function Documentation

### 16.467.3.1 get()

```
template<typename T>
T L4Re::Rm::Unique_region< T >::get () const [inline], [noexcept]
```

Return the address.

#### Returns

the address

Definition at line 509 of file [rm](#).

**16.467.3.2 is\_valid()**

```
template<typename T>
bool L4Re::Rm::Unique_region< T >::is_valid () const [inline], [noexcept]
```

Check if the `Unique_region` is valid.

**Returns**

true iff the `Unique_region` is valid

Definition at line 549 of file `rm`.

**16.467.3.3 operator=()**

```
template<typename T>
Unique_region & L4Re::Rm::Unique_region< T >::operator= (
 Unique_region< T > && o) [inline], [noexcept]
```

Move-assign a `Unique_region`.

**Parameters**

|          |                                            |
|----------|--------------------------------------------|
| <i>o</i> | L-value reference to region to assign from |
|----------|--------------------------------------------|

Definition at line 481 of file `rm`.

**16.467.3.4 release()**

```
template<typename T>
T L4Re::Rm::Unique_region< T >::release () [inline], [noexcept]
```

Return the address and invalidate the `Unique_region`.

**Returns**

the address

Definition at line 517 of file `rm`.

**16.467.3.5 reset()**

```
template<typename T>
void L4Re::Rm::Unique_region< T >::reset (
 T addr,
 L4::Cap< Rm > const & rm) [inline], [noexcept]
```

Set new address and region manager.

**Parameters**



|             |                        |
|-------------|------------------------|
| <i>addr</i> | The new address        |
| <i>rm</i>   | The new region manager |

Definition at line 529 of file [rm](#).

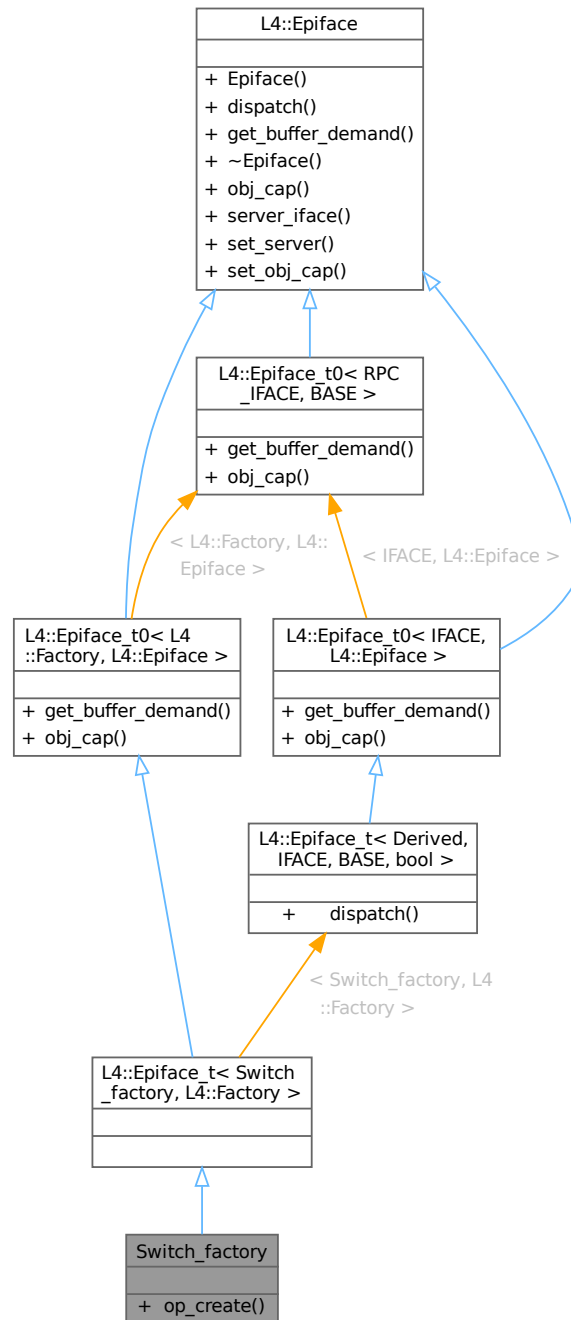
The documentation for this class was generated from the following file:

- [l4/re/rm](#)

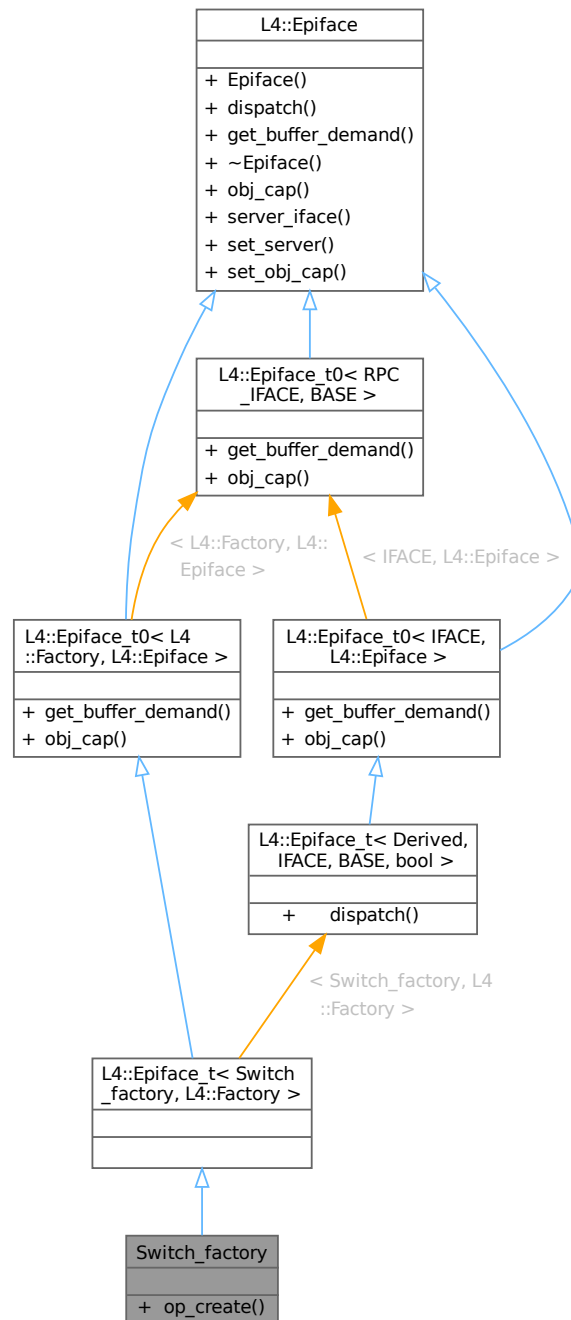
## 16.468 Switch\_factory Class Reference

The IPC interface for creating ports.

Inheritance diagram for Switch\_factory:



Collaboration diagram for Switch\_factory:



## Public Member Functions

- long `op_create` (L4::Factory::Rights, L4::lpc::Cap< void > &res, l4\_umword\_t type, L4::lpc::Varg\_list\_ref va)  
Handle factory protocol.

## Public Member Functions inherited from L4::Epiface\_t0< L4::Factory, L4::Epiface >

- Type\_info::Demand `get_buffer_demand` () const

*Get the server-side buffer demand based in IFACE.*

- `Cap< L4::Factory > obj_cap () const`

*Get the (typed) capability to this object.*

## Public Member Functions inherited from `L4::Epiface`

- `Epiface ()`

*Make a server object.*

- `virtual ~Epiface ()=0`

*Destroy the object.*

- `Stored_cap obj_cap () const`

*Get the capability to the kernel object belonging to this object.*

- `Server_iface * server_iface () const`

*Get pointer to server interface at which the object is currently registered.*

- `int set_server (Server_iface *srv, Cap< void > cap, bool managed=false)`

*Set server registration info for the object.*

- `void set_obj_cap (Cap< void > const &cap)`

*Deprecated server registration function.*

## Additional Inherited Members

## Public Types inherited from `L4::Epiface_t0< L4::Factory, L4::Epiface >`

- `typedef L4::Factory Interface`

*Data type of the IPC interface definition.*

## Public Types inherited from `L4::Epiface`

- `typedef ipc_svr::Server_iface Server_iface`

*Type for abstract server interface.*

- `typedef ipc_svr::Server_iface::Demand Demand`

*Type for server-side receive buffer demand.*

### 16.468.1 Detailed Description

The IPC interface for creating ports.

The Switch factory provides an IPC interface to create ports. Ports are the only option for a client to communicate with the switch and, thus, with other network devices.

The `Switch_factory` gets constructed when the net switch application gets started. It thereafter gets registered on the switch's server to serve IPC `create` calls.

Definition at line 114 of file `main.cc`.

## 16.468.2 Member Function Documentation

### 16.468.2.1 op\_create()

```
long Switch_factory::op_create (
 L4::Factory::Rights ,
 L4::Ipc::Cap< void > & res,
 l4_umword_t type,
 L4::Ipc::Varg_list_ref va) [inline]
```

Handle factory protocol.

This function is invoked after an incoming factory::create request and creates a new port or statistics interface if possible.

Definition at line 528 of file [main.cc](#).

References [L4\\_EINVAL](#).

The documentation for this class was generated from the following file:

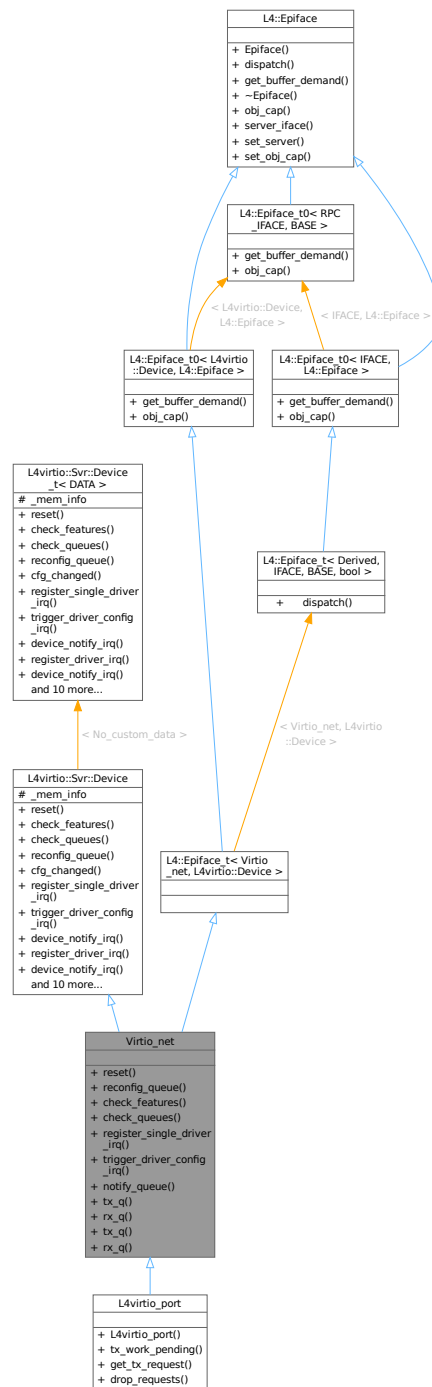
- pkg/virtio-net-switch/server/switch/main.cc

## 16.469 Virtio\_net Class Reference

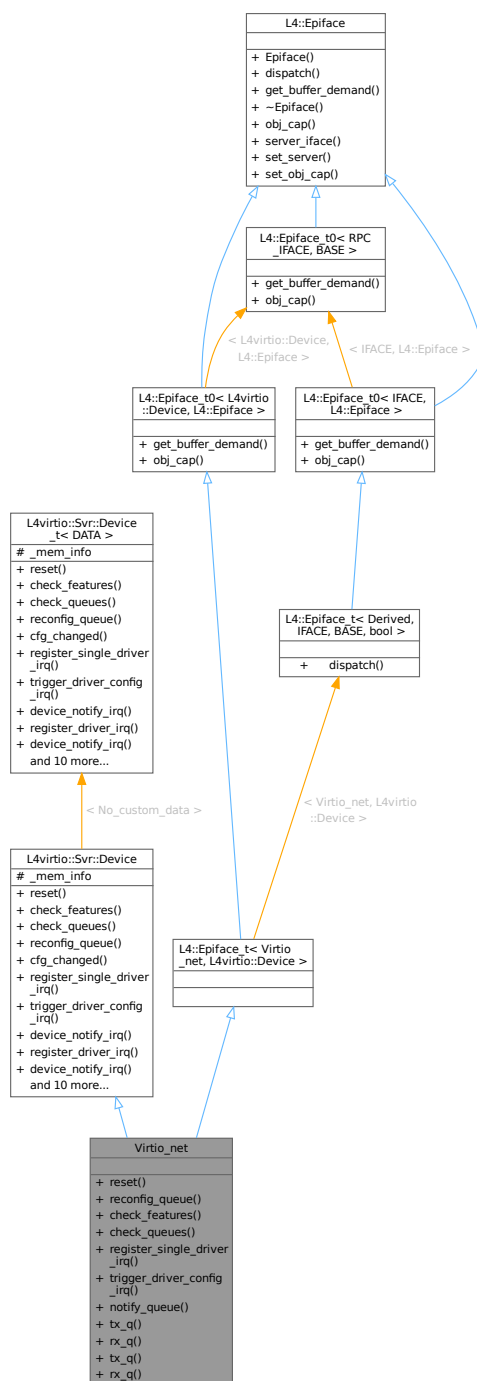
The Base class of a Port.

```
#include <virtio_net.h>
```

Inheritance diagram for Virtio\_net:



Collaboration diagram for Virtio\_net:



## Public Member Functions

- void **reset** () override  
*reset callback, called for doing a device reset*
- int **reconfig\_queue** (unsigned index) override  
*callback for client queue-config request*
- bool **check\_features** () override

- callback for checking the subset of accepted features*
- bool **check\_queues** () override
  - Check whether both virtqueues are ready.*
- void **register\_single\_driver\_irq** () override
  - Save the `_kick_guest_irq` that the client sent via `device_notification_irq()`.*
- void **trigger\_driver\_config\_irq** () override
  - callback for triggering configuration change notification IRQ*
- void **notify\_queue** (L4virtio::Svr::Virtqueue \*queue)
  - Trigger the `_kick_guest_irq` IRQ.*
- Virtqueue \* **tx\_q** ()
  - Getter for the transmission queue.*
- Virtqueue \* **rx\_q** ()
  - Getter for the receive queue.*
- Virtqueue const \* **tx\_q** () const
  - Getter for the transmission queue.*
- Virtqueue const \* **rx\_q** () const
  - Getter for the receive queue.*

## Public Member Functions inherited from L4virtio::Svr::Device\_t< No\_custom\_data >

- virtual void **cfg\_changed** (unsigned)
  - callback for client device configuration changes*
- virtual L4::Cap< L4::Irq > **device\_notify\_irq** () const
  - callback to gather the device notification IRQ (old-style)*
- virtual void **register\_driver\_irq** (unsigned idx)
  - Callback for registering an notification IRQ (multi IRQ).*
- virtual L4::Cap< L4::Irq > **device\_notify\_irq** (unsigned idx)
  - Callback to gather the device notification IRQ (multi IRQ).*
- virtual unsigned **num\_events\_supported** () const
  - Return the highest notification index supported.*
- **Device\_t** (Dev\_config \*dev\_config)
  - Make a device for the given config.*
- Mem\_list const \* **mem\_info** () const
  - Get the memory region list used for this device.*
- void **reset\_queue\_config** (unsigned idx, unsigned num\_max, bool inc\_generation=false)
  - Trigger reset for the configuration space for queue idx.*
- void **init\_mem\_info** (unsigned num)
  - Initialize the memory region list to the given maximum.*
- void **device\_error** ()
  - Transition device into DEVICE\_NEEDS\_RESET state.*
- bool **setup\_queue** (Virtqueue \*q, unsigned qn, unsigned num\_max)
  - Enable/disable the specified queue.*
- bool **handle\_mem\_cmd\_write** ()
  - Check for a value in the `cmd` register and handle a write.*
- void **enable\_trusted\_ds\_validation** ()
  - Enable trusted dataspace validation.*
- void **add\_trusted\_dataspaces** (std::shared\_ptr< Ds\_vector const > ds)
  - Provide a list of trusted dataspaces that can be used for validation.*



## Public Member Functions inherited from [L4::Epiface\\_t0](#)< [L4virtio::Device](#), [L4::Epiface](#) >

- [Type\\_info::Demand](#) **get\_buffer\_demand** () const  
*Get the server-side buffer demand based in IFACE.*
- [Cap](#)< [L4virtio::Device](#) > **obj\_cap** () const  
*Get the (typed) capability to this object.*

## Public Member Functions inherited from [L4::Epiface](#)

- **Epiface** ()  
*Make a server object.*
- virtual **~Epiface** ()=0  
*Destroy the object.*
- Stored\_cap [obj\\_cap](#) () const  
*Get the capability to the kernel object belonging to this object.*
- [Server\\_iface](#) \* **server\_iface** () const  
*Get pointer to server interface at which the object is currently registered.*
- int **set\_server** ([Server\\_iface](#) \*srv, [Cap](#)< void > cap, bool managed=false)  
*Set server registration info for the object.*
- void **set\_obj\_cap** ([Cap](#)< void > const &cap)  
*Deprecated server registration function.*

## Additional Inherited Members

## Public Types inherited from [L4::Epiface\\_t0](#)< [L4virtio::Device](#), [L4::Epiface](#) >

- typedef [L4virtio::Device](#) **Interface**  
*Data type of the IPC interface definition.*

## Public Types inherited from [L4::Epiface](#)

- typedef [lpc\\_svr::Server\\_iface](#) **Server\_iface**  
*Type for abstract server interface.*
- typedef [lpc\\_svr::Server\\_iface::Demand](#) **Demand**  
*Type for server-side receive buffer demand.*

## Protected Attributes inherited from [L4virtio::Svr::Device\\_t](#)< [No\\_custom\\_data](#) >

- [Mem\\_list](#) **\_mem\_info**  
*Memory region list.*

### 16.469.1 Detailed Description

The Base class of a Port.

This class provides the Virtio network protocol specific implementation aspects of a port.

[Virtio\\_net](#) comprises the virtqueues for both, the incoming and the outgoing network requests:

- The transmission queue, containing requests to be transmitted to other ports. The transmission queue is filled by the client, this port relates to.
- The receive queue, containing requests that have been transmitted from other ports. The receive queue is filled by the switch.

Definition at line 71 of file [virtio\\_net.h](#).

### 16.469.2 Member Function Documentation

#### 16.469.2.1 notify\_queue()

```
void Virtio_net::notify_queue (
 L4virtio::Svr::Virtqueue * queue) [inline]
```

Trigger the `_kick_guest_irq` IRQ.

This function gets called on the receiving port, when a request was successfully transmitted by the switch.

Definition at line 269 of file [virtio\\_net.h](#).

References [L4VIRTIO\\_IRQ\\_STATUS\\_VRING](#).

The documentation for this class was generated from the following file:

- `pkg/virtio-net-switch/server/switch/virtio_net.h`

## 16.470 Virtio\_net\_request Class Reference

Abstraction for a network request.

```
#include <request_l4virtio.h>
```

Collaboration diagram for `Virtio_net_request`:

| Virtio_net_request                                                                                                                                                                         |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>+ <code>dst_mac()</code></li> <li>+ <code>src_mac()</code></li> <li>+ <code>drop_requests()</code></li> <li>+ <code>get_request()</code></li> </ul> |

## Public Member Functions

- [Mac\\_addr dst\\_mac](#) () const  
*Get the Mac address of the destination port.*
- [Mac\\_addr src\\_mac](#) () const  
*Get the Mac address of the source port.*

## Static Public Member Functions

- static void [drop\\_requests](#) ([Virtio\\_net](#) \*dev, [L4virtio::Svr::Virtqueue](#) \*queue)  
*Drop all requests of a specific queue.*
- static std::optional< [Virtio\\_net\\_request](#) > [get\\_request](#) ([Virtio\\_net](#) \*dev, [L4virtio::Svr::Virtqueue](#) \*queue)  
*Construct a request from the next entry of a provided queue.*

## 16.470.1 Detailed Description

Abstraction for a network request.

A [Virtio\\_net\\_request](#) is constructed by the source port, using the static function [get\\_request](#) () as part of [Port\\_iface::get\\_tx\\_request](#) ().

On destruction, [finish](#) () will be called, which, will trigger the client IRQ of the source client.

Definition at line 35 of file [request\\_l4virtio.h](#).

## 16.470.2 Member Function Documentation

### 16.470.2.1 drop\_requests()

```
void Virtio_net_request::drop_requests (
 Virtio_net * dev,
 L4virtio::Svr::Virtqueue * queue) [inline], [static]
```

Drop all requests of a specific queue.

This function is used for example to drop all requests in the transmission queue of a monitor port, since monitor ports are not allowed to transmit data.

### Parameters

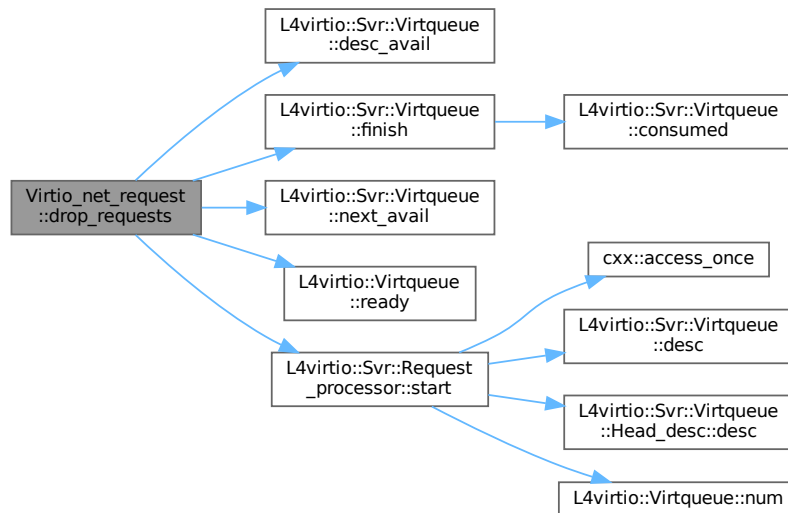
|              |                                    |
|--------------|------------------------------------|
| <i>dev</i>   | Port of the provided virtqueue.    |
| <i>queue</i> | Virtqueue to drop all requests of. |

Definition at line 172 of file [request\\_l4virtio.h](#).

References [L4virtio::Svr::Virtqueue::desc\\_avail](#)(), [L4virtio::Svr::Virtqueue::finish](#)(), [L4\\_UNLIKELY](#), [L4virtio::Svr::Virtqueue::next\\_avail](#)(), [L4virtio::Virtqueue::ready](#)(), and [L4virtio::Svr::Request\\_processor::start](#)().

Referenced by [L4virtio\\_port::drop\\_requests](#)().

Here is the call graph for this function:



Here is the caller graph for this function:



### 16.470.2.2 get\_request()

```

std::optional< Virtio_net_request > Virtio_net_request::get_request (
 Virtio_net * dev,
 L4virtio::Svr::Virtqueue * queue) [inline], [static]

```

Construct a request from the next entry of a provided queue.

#### Parameters

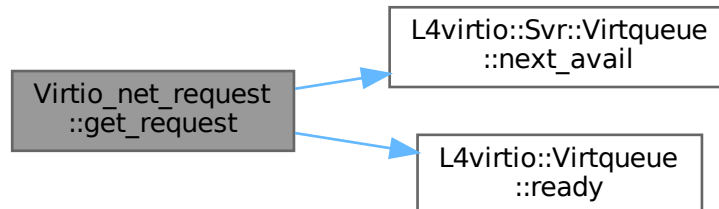
|              |                                       |
|--------------|---------------------------------------|
| <i>dev</i>   | Port of the provided virtqueue.       |
| <i>queue</i> | Virtqueue to extract next entry from. |

Definition at line 199 of file [request\\_l4virtio.h](#).

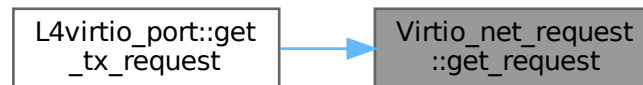
References [L4\\_UNLIKELY](#), [L4virtio::Svr::Virtqueue::next\\_avail\(\)](#), and [L4virtio::Virtqueue::ready\(\)](#).

Referenced by [L4virtio\\_port::get\\_tx\\_request\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

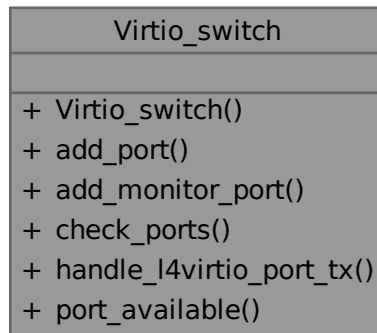
- `pkg/virtio-net-switch/server/switch/request_l4virtio.h`

## 16.471 Virtio\_switch Class Reference

The Virtio switch contains all ports and processes network requests.

```
#include <switch.h>
```

Collaboration diagram for Virtio\_switch:



### Public Member Functions

- [Virtio\\_switch](#) (unsigned max\_ports)  
*Create a switch with n ports.*
- bool [add\\_port](#) (Port\_iface \*port)  
*Add a port to the switch.*
- bool [add\\_monitor\\_port](#) (Port\_iface \*port)  
*Add a monitor port to the switch.*
- void [check\\_ports](#) ()  
*Check validity of ports.*
- bool [handle\\_l4virtio\\_port\\_tx](#) (L4virtio\_port \*port)  
*Handle TX queue of the given port.*
- int [port\\_available](#) (bool monitor)  
*Is there still a free port on this switch available?*

## 16.471.1 Detailed Description

The Virtio switch contains all ports and processes network requests.

A Port on its own is not capable to process an incoming network request because it has no knowledge about other ports. The processing of an incoming request therefore gets delegated to the switch.

The [Virtio\\_switch](#) is constructed at the start of the Virtio Net Switch application. The factory saves a reference to it to pass it to the [Kick\\_irq](#) on port creation.

Definition at line 33 of file [switch.h](#).

## 16.471.2 Constructor & Destructor Documentation

### 16.471.2.1 Virtio\_switch()

```
Virtio_switch::Virtio_switch (
 unsigned max_ports) [explicit]
```

Create a switch with n ports.

### Parameters

|                  |                                  |
|------------------|----------------------------------|
| <i>max_ports</i> | maximal number of provided ports |
|------------------|----------------------------------|

Definition at line 12 of file [switch.cc](#).

## 16.471.3 Member Function Documentation

### 16.471.3.1 add\_monitor\_port()

```
bool Virtio_switch::add_monitor_port (
 Port_iface * port)
```

Add a monitor port to the switch.

#### Parameters

|             |                                                        |
|-------------|--------------------------------------------------------|
| <i>port</i> | A pointer to an already constructed Port_iface object. |
|-------------|--------------------------------------------------------|

#### Return values

|              |                                      |
|--------------|--------------------------------------|
| <i>true</i>  | Port was added successfully.         |
| <i>false</i> | Switch was not able to add the port. |

Definition at line 54 of file [switch.cc](#).

### 16.471.3.2 add\_port()

```
bool Virtio_switch::add_port (
 Port_iface * port)
```

Add a port to the switch.

#### Parameters

|             |                                                        |
|-------------|--------------------------------------------------------|
| <i>port</i> | A pointer to an already constructed Port_iface object. |
|-------------|--------------------------------------------------------|

#### Return values

|              |                                      |
|--------------|--------------------------------------|
| <i>true</i>  | Port was added successfully.         |
| <i>false</i> | Switch was not able to add the port. |

Definition at line 29 of file [switch.cc](#).

References [Mac\\_addr::is\\_unknown\(\)](#).

Here is the call graph for this function:



### 16.471.3.3 check\_ports()

```
void Virtio_switch::check_ports ()
```

Check validity of ports.

Check whether all ports are still used and remove any unused (unreferenced) ports. Shall be invoked after an incoming cap deletion irq to remove ports without clients.

Definition at line 69 of file [switch.cc](#).

### 16.471.3.4 handle\_l4virtio\_port\_tx()

```
bool Virtio_switch::handle_l4virtio_port_tx (
 L4virtio_port * port)
```

Handle TX queue of the given port.

#### Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <i>port</i> | <a href="#">L4virtio_port</a> to handle pending TX work for. |
|-------------|--------------------------------------------------------------|

#### Return values

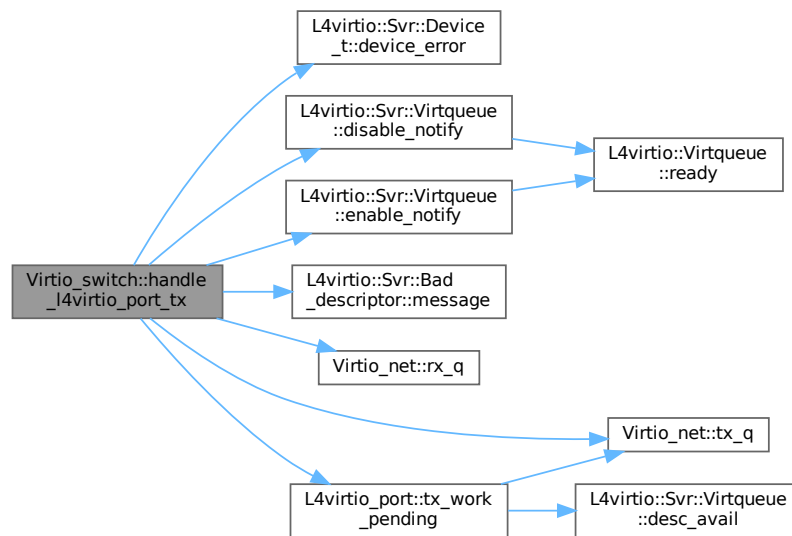
|              |                                                                                        |
|--------------|----------------------------------------------------------------------------------------|
| <i>false</i> | Port hit its TX burst limit, and thus a TX pending reschedule notification was queued. |
| <i>true</i>  | Port's entire TX queue was processed.                                                  |

Definition at line 194 of file [switch.cc](#).

References [L4virtio::Svr::Device\\_t< DATA >::device\\_error\(\)](#), [L4virtio::Svr::Virtqueue::disable\\_notify\(\)](#), [L4virtio::Svr::Virtqueue::enable\\_notify\(\)](#), [L4virtio::Svr::Bad\\_descriptor::message\(\)](#), [Virtio\\_net::rx\\_q\(\)](#), [Virtio\\_net::tx\\_q\(\)](#), and [L4virtio\\_port::tx\\_work\\_pending\(\)](#).



Here is the call graph for this function:



### 16.471.3.5 port\_available()

```
int Virtio_switch::port_available (
 bool monitor) [inline]
```

Is there still a free port on this switch available?

#### Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>monitor</i> | True if we look for a monitor slot. |
|----------------|-------------------------------------|

#### Return values

|          |                                |
|----------|--------------------------------|
| $\geq 0$ | The next available port index. |
| -1       | No port available.             |

Definition at line 148 of file [switch.h](#).

The documentation for this class was generated from the following files:

- pkg/virtio-net-switch/server/switch/switch.h
- pkg/virtio-net-switch/server/switch/switch.cc

## 16.472 Virtio\_vlan\_mangle Class Reference

Class for VLAN packet rewriting.

```
#include <vlan.h>
```

Collaboration diagram for Virtio\_vlan\_mangle:

| Virtio_vlan_mangle                                                                                                                                           |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>+ Virtio_vlan_mangle()</li> <li>+ copy_pkt()</li> <li>+ rewrite_hdr()</li> <li>+ add()</li> <li>+ remove()</li> </ul> |

### Public Member Functions

- [Virtio\\_vlan\\_mangle](#) ()  
*Default constructor.*
- [l4\\_uint32\\_t copy\\_pkt](#) ([Buffer](#) &dst, [Buffer](#) &src)  
*Copy packet from src to dst.*
- void [rewrite\\_hdr](#) ([Virtio\\_net::Hdr](#) \*hdr)  
*Rewrite the virtio network header.*

### Static Public Member Functions

- static constexpr [Virtio\\_vlan\\_mangle add](#) ([l4\\_uint16\\_t](#) tci)  
*Construct an object that adds a VLAN tag.*
- static constexpr [Virtio\\_vlan\\_mangle remove](#) ()  
*Construct an object that removes the VLAN tag.*

### 16.472.1 Detailed Description

Class for VLAN packet rewriting.

Definition at line 36 of file [vlan.h](#).

## 16.472.2 Constructor & Destructor Documentation

### 16.472.2.1 Virtio\_vlan\_mangle()

`Virtio_vlan_mangle::Virtio_vlan_mangle () [inline]`

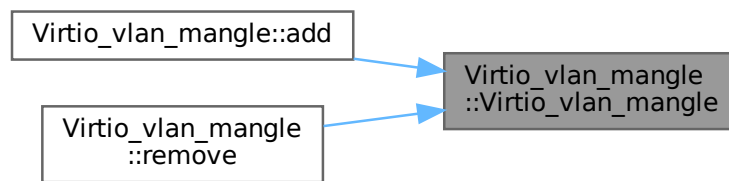
Default constructor.

The packet is not touched in any way.

Definition at line 52 of file [vlan.h](#).

Referenced by [add\(\)](#), and [remove\(\)](#).

Here is the caller graph for this function:



## 16.472.3 Member Function Documentation

### 16.472.3.1 add()

```
constexpr Virtio_vlan_mangle Virtio_vlan_mangle::add (
 14_uint16_t tci) [inline], [static], [constexpr]
```

Construct an object that adds a VLAN tag.

#### Parameters

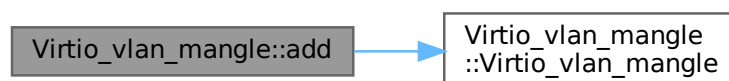
|            |                                       |
|------------|---------------------------------------|
| <i>tci</i> | The TCI field of the VLAN tag to add. |
|------------|---------------------------------------|

It is the callers responsibility to ensure that the packet is not already tagged.

Definition at line 64 of file [vlan.h](#).

References [Virtio\\_vlan\\_mangle\(\)](#).

Here is the call graph for this function:



### 16.472.3.2 copy\_pkt()

```
l4_uint32_t Virtio_vlan_mangle::copy_pkt (
 Buffer & dst,
 Buffer & src) [inline]
```

Copy packet from *src* to *dst*.

#### Parameters

|            |                           |
|------------|---------------------------|
| <i>src</i> | Source packet buffer      |
| <i>dst</i> | Destination packet buffer |

#### Returns

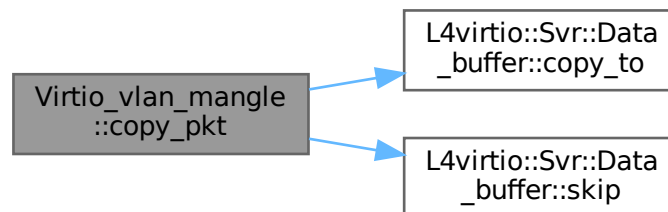
The number of bytes copied

Copy the data from *src* to *dst*, possibly rewriting parts of the packet. The method is expected to be called repeatedly until the source packet is finished. Partial copies are allowed (including reading nothing from the source buffer) as long as progress is made, i.e. repeatedly calling this function eventually consumes the source buffer.

Definition at line 93 of file [vlan.h](#).

References [L4virtio::Svr::Data\\_buffer::copy\\_to\(\)](#), [L4\\_LIKELY](#), [L4virtio::Svr::Data\\_buffer::left](#), [L4virtio::Svr::Data\\_buffer::pos](#), and [L4virtio::Svr::Data\\_buffer::skip\(\)](#).

Here is the call graph for this function:



### 16.472.3.3 remove()

```
constexpr Virtio_vlan_mangle Virtio_vlan_mangle::remove () [inline], [static], [constexpr]
```

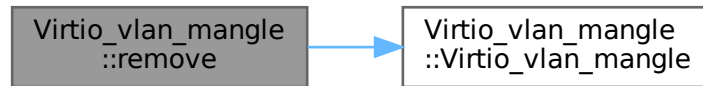
Construct an object that removes the VLAN tag.

This object assumes that the Ethernet packet has a VLAN tag and will slavishly remove the necessary bytes from the packet.

Definition at line 75 of file [vlan.h](#).

References [Virtio\\_vlan\\_mangle\(\)](#).

Here is the call graph for this function:



#### 16.472.3.4 `rewrite_hdr()`

```
void Virtio_vlan_mangle::rewrite_hdr (
 Virtio_net::Hdr * hdr) [inline]
```

Rewrite the virtio network header.

##### Parameters

|            |                                 |
|------------|---------------------------------|
| <i>hdr</i> | The virtio header of the packet |
|------------|---------------------------------|

This method is called exactly once for every virtio network packet. Any necessary changes to the header are done in-place.

Definition at line 142 of file [vlan.h](#).

References [L4\\_UNLIKELY](#).

The documentation for this class was generated from the following file:

- `pkg/virtio-net-switch/server/switch/vlan.h`



# Chapter 17

## File Documentation

### 17.1 asm\_access.h

```
00001 /*
00002 * Copyright (C) 2021, 2024 Kernkonzept GmbH.
00003 * Author(s): Jakub Jermar <jakub.jermar@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #pragma once
00009
00010 #include <l4/sys/l4int.h>
00011 #include <x86/l4/drivers/asm_access.h>
00012
00013 namespace Asm_access {
00014
00015 inline
00016 l4_uint64_t
00017 read(l4_uint64_t const *mem)
00018 {
00019 l4_uint64_t val;
00020
00021 asm volatile ("movq %[mem], %[val]" : [val] "=r" (val) : [mem] "m" (*mem));
00022
00023 return val;
00024 }
00025
00026 inline
00027 void
00028 write(l4_uint64_t val, l4_uint64_t *mem)
00029 {
00030 asm volatile ("movq %[val], %[mem]" : [mem] "=m" (*mem) : [val] "r" (val));
00031 }
00032 }
00033
```

### 17.2 asm\_access.h

```
00001 /*
00002 * Copyright (C) 2021, 2024 Kernkonzept GmbH.
00003 * Author(s): Jakub Jermar <jakub.jermar@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #pragma once
00009
00010 #include <l4/sys/l4int.h>
00011
00012 namespace Asm_access {
00013
00014 inline
00015 l4_uint8_t
00016 read(l4_uint8_t const *mem)
00017 {
```

```

00018 14_uint8_t val;
00019
00020 asm volatile ("ldrb %[val], %[mem]" : [val] "=r" (val) : [mem] "m" (*mem));
00021
00022 return val;
00023 }
00024
00025 inline
00026 14_uint16_t
00027 read(14_uint16_t const *mem)
00028 {
00029 14_uint16_t val;
00030
00031 asm volatile ("ldrh %[val], %[mem]" : [val] "=r" (val) : [mem] "m" (*mem));
00032
00033 return val;
00034 }
00035
00036 inline
00037 14_uint32_t
00038 read(14_uint32_t const *mem)
00039 {
00040 14_uint32_t val;
00041
00042 asm volatile ("ldr %[val], %[mem]" : [val] "=r" (val) : [mem] "m" (*mem));
00043
00044 return val;
00045 }
00046
00047 inline
00048 void
00049 write(14_uint8_t val, 14_uint8_t *mem)
00050 {
00051 asm volatile ("strb %[val], %[mem]" : [mem] "=m" (*mem) : [val] "r" (val));
00052 }
00053
00054 inline
00055 void
00056 write(14_uint16_t val, 14_uint16_t *mem)
00057 {
00058 asm volatile ("strh %[val], %[mem]" : [mem] "=m" (*mem) : [val] "r" (val));
00059 }
00060
00061 inline
00062 void
00063 write(14_uint32_t val, 14_uint32_t *mem)
00064 {
00065 asm volatile ("str %[val], %[mem]" : [mem] "=m" (*mem) : [val] "r" (val));
00066 }
00067
00068 }

```

## 17.3 asm\_access.h

```

00001 /*
00002 * Copyright (C) 2021, 2024 Kernkonzept GmbH.
00003 * Author(s): Jakub Jermar <jakub.jermar@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #pragma once
00009
00010 #include <14/sys/l4int.h>
00011
00012 namespace Asm_access {
00013
00014 inline
00015 14_uint8_t
00016 read(14_uint8_t const *mem)
00017 {
00018 14_uint8_t val;
00019
00020 asm volatile ("ldrb %w[val], %[mem]" : [val] "=r" (val) : [mem] "m" (*mem));
00021
00022 return val;
00023 }
00024
00025 inline
00026 14_uint16_t
00027 read(14_uint16_t const *mem)
00028 {
00029 14_uint16_t val;

```



```

00030
00031 asm volatile ("ldrh %w[val], %[mem]" : [val] "=r" (val) : [mem] "m" (*mem));
00032
00033 return val;
00034 }
00035
00036 inline
00037 l4_uint32_t
00038 read(l4_uint32_t const *mem)
00039 {
00040 l4_uint32_t val;
00041
00042 asm volatile ("ldr %w[val], %[mem]" : [val] "=r" (val) : [mem] "m" (*mem));
00043
00044 return val;
00045 }
00046
00047 inline
00048 l4_uint64_t
00049 read(l4_uint64_t const *mem)
00050 {
00051 l4_uint64_t val;
00052
00053 asm volatile ("ldr %[val], %[mem]" : [val] "=r" (val) : [mem] "m" (*mem));
00054
00055 return val;
00056 }
00057
00058 inline
00059 void
00060 write(l4_uint8_t val, l4_uint8_t *mem)
00061 {
00062 asm volatile ("strb %w[val], %[mem]" : [mem] "=m" (*mem) : [val] "r" (val));
00063 }
00064
00065 inline
00066 void
00067 write(l4_uint16_t val, l4_uint16_t *mem)
00068 {
00069 asm volatile ("strh %w[val], %[mem]" : [mem] "=m" (*mem) : [val] "r" (val));
00070 }
00071
00072 inline
00073 void
00074 write(l4_uint32_t val, l4_uint32_t *mem)
00075 {
00076 asm volatile ("str %w[val], %[mem]" : [mem] "=m" (*mem) : [val] "r" (val));
00077 }
00078
00079 inline
00080 void
00081 write(l4_uint64_t val, l4_uint64_t *mem)
00082 {
00083 asm volatile ("str %[val], %[mem]" : [mem] "=m" (*mem) : [val] "r" (val));
00084 }
00085
00086 }

```

## 17.4 asm\_access.h

```

00001 /*
00002 * Copyright (C) 2021, 2024 Kernkonzept GmbH.
00003 * Author(s): Jakub Jermar <jakub.jermar@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #pragma once
00009
00010 #include <l4/drivers/asm_access_gen.h>

```

## 17.5 asm\_access.h

```

00001 /*
00002 * Copyright (C) 2021, 2024 Kernkonzept GmbH.
00003 * Author(s): Jakub Jermar <jakub.jermar@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)

```

```

00006 */
00007
00008 #pragma once
00009
00010 #include <l4/drivers/asm_access_gen.h>

```

## 17.6 asm\_access.h

```

00001 /*
00002 * Copyright (C) 2021, 2024 Kernkonzept GmbH.
00003 * Author(s): Georg Kotheimer <georg.kotheimer@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #pragma once
00009
00010 #include <l4/sys/l4int.h>
00011
00012 namespace Asm_access {
00013
00014 inline
00015 l4_uint8_t
00016 read(l4_uint8_t const *mem)
00017 {
00018 l4_uint8_t val;
00019
00020 asm volatile ("lb %[val], %[mem]" : [val] "=r" (val) : [mem] "m" (*mem));
00021
00022 return val;
00023 }
00024
00025 inline
00026 l4_uint16_t
00027 read(l4_uint16_t const *mem)
00028 {
00029 l4_uint16_t val;
00030
00031 asm volatile ("lh %[val], %[mem]" : [val] "=r" (val) : [mem] "m" (*mem));
00032
00033 return val;
00034 }
00035
00036 inline
00037 l4_uint32_t
00038 read(l4_uint32_t const *mem)
00039 {
00040 l4_uint32_t val;
00041
00042 asm volatile ("lw %[val], %[mem]" : [val] "=r" (val) : [mem] "m" (*mem));
00043
00044 return val;
00045 }
00046
00047 inline
00048 void
00049 write(l4_uint8_t val, l4_uint8_t *mem)
00050 {
00051 asm volatile ("sb %[val], %[mem]" : [mem] "=m" (*mem) : [val] "r" (val));
00052 }
00053
00054 inline
00055 void
00056 write(l4_uint16_t val, l4_uint16_t *mem)
00057 {
00058 asm volatile ("sh %[val], %[mem]" : [mem] "=m" (*mem) : [val] "r" (val));
00059 }
00060
00061 inline
00062 void
00063 write(l4_uint32_t val, l4_uint32_t *mem)
00064 {
00065 asm volatile ("sw %[val], %[mem]" : [mem] "=m" (*mem) : [val] "r" (val));
00066 }
00067
00068 #if __riscv_xlen == 64
00069
00070 inline
00071 l4_uint64_t
00072 read(l4_uint64_t const *mem)
00073 {
00074 l4_uint64_t val;
00075

```

```

00076 asm volatile ("ld %[val], %[mem]" : [val] "=r" (val) : [mem] "m" (*mem));
00077
00078 return val;
00079 }
00080
00081 inline
00082 void
00083 write(l4_uint64_t val, l4_uint64_t *mem)
00084 {
00085 asm volatile ("sd %[val], %[mem]" : [mem] "=m" (*mem) : [val] "r" (val));
00086 }
00087
00088 #endif
00089
00090 }

```

## 17.7 asm\_access.h

```

00001 /*
00002 * Copyright (C) 2021, 2024 Kernkonzept GmbH.
00003 * Author(s): Jakub Jermar <jakub.jermar@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #pragma once
00009
00010 #include <l4/drivers/asm_access_gen.h>

```

## 17.8 asm\_access.h

```

00001 /*
00002 * Copyright (C) 2021, 2024 Kernkonzept GmbH.
00003 * Author(s): Jakub Jermar <jakub.jermar@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #pragma once
00009
00010 #include <l4/sys/l4int.h>
00011
00012 namespace Asm_access {
00013
00014 inline
00015 l4_uint8_t
00016 read(l4_uint8_t const *mem)
00017 {
00018 l4_uint8_t val;
00019
00020 asm volatile ("movb %[mem], %[val]" : [val] "=q" (val) : [mem] "m" (*mem));
00021
00022 return val;
00023 }
00024
00025 inline
00026 l4_uint16_t
00027 read(l4_uint16_t const *mem)
00028 {
00029 l4_uint16_t val;
00030
00031 asm volatile ("movw %[mem], %[val]" : [val] "=r" (val) : [mem] "m" (*mem));
00032
00033 return val;
00034 }
00035
00036 inline
00037 l4_uint32_t
00038 read(l4_uint32_t const *mem)
00039 {
00040 l4_uint32_t val;
00041
00042 asm volatile ("movl %[mem], %[val]" : [val] "=r" (val) : [mem] "m" (*mem));
00043
00044 return val;
00045 }
00046
00047 inline

```

```

00048 void
00049 write(l4_uint8_t val, l4_uint8_t *mem)
00050 {
00051 asm volatile ("movb %[val], %[mem]" : [mem] "=m" (*mem) : [val] "qi" (val));
00052 }
00053
00054 inline
00055 void
00056 write(l4_uint16_t val, l4_uint16_t *mem)
00057 {
00058 asm volatile ("movw %[val], %[mem]" : [mem] "=m" (*mem) : [val] "ri" (val));
00059 }
00060
00061 inline
00062 void
00063 write(l4_uint32_t val, l4_uint32_t *mem)
00064 {
00065 asm volatile ("movl %[val], %[mem]" : [mem] "=m" (*mem) : [val] "ri" (val));
00066 }
00067
00068 }

```

## 17.9 asm\_access\_gen.h

```

00001 /*
00002 * Copyright (C) 2021, 2024 Kernkonzept GmbH.
00003 * Author(s): Jakub Jermar <jakub.jermar@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/sys/l4int.h>
00010 #include <l4/cxx/type_traits>
00011
00012 namespace Asm_access {
00013
00014 template <typename T>
00015 struct is_supported_type
00016 {
00017 static const bool value = cxx::is_same<T, l4_uint8_t>::value
00018 || cxx::is_same<T, l4_uint16_t>::value
00019 || cxx::is_same<T, l4_uint32_t>::value
00020 || cxx::is_same<T, l4_uint64_t>::value;
00021 };
00022
00023 template <typename T>
00024 inline
00025 typename cxx::enable_if<is_supported_type<T>::value, T>::type
00026 read(T const *mem)
00027 {
00028 return *reinterpret_cast<volatile T const *>(mem);
00029 }
00030
00031 template <typename T>
00032 inline
00033 typename cxx::enable_if<is_supported_type<T>::value, void>::type
00034 write(T val, T *mem)
00035 {
00036 *reinterpret_cast<volatile T *>(mem) = val;
00037 }
00038
00039 }

```

## 17.10 hw\_mmio\_register\_block

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * Copyright (C) 2014-2021, 2023-2024 Kernkonzept GmbH.
00004 * Author(s): Alexander Warg <alexander.warg@kernkonzept.com>
00005 * Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 #pragma once
00010
00011 #include <l4/drivers/hw_register_block>
00012 #include <l4/drivers/asm_access.h>

```

```

00013
00014 namespace L4drivers {
00015
00016 class Mmio_register_block_base
00017 {
00018 protected:
00019 l4_addr_t _base;
00020 l4_addr_t _shift;
00021
00022 public:
00023 explicit Mmio_register_block_base(l4_addr_t base = 0, l4_addr_t shift = 0)
00024 : _base(base), _shift(shift) {}
00025
00026 template< typename T >
00027 T read(l4_addr_t reg) const
00028 { return Asm_access::read(reinterpret_cast<T const *>(_base + (reg « _shift))); }
00029
00030 template< typename T >
00031 void write(T value, l4_addr_t reg) const
00032 { Asm_access::write(value, reinterpret_cast<T *>(_base + (reg « _shift))); }
00033
00034 void set_base(l4_addr_t base) { _base = base; }
00035 void set_shift(l4_addr_t shift) { _shift = shift; }
00036 };
00037
00042 template< unsigned MAX_BITS = 32 >
00043 struct Mmio_register_block
00044 : Register_block_impl<Mmio_register_block<MAX_BITS>, MAX_BITS>,
00045 Mmio_register_block_base
00046 {
00047 explicit Mmio_register_block(l4_addr_t base = 0, l4_addr_t shift = 0)
00048 : Mmio_register_block_base(base, shift) {}
00049 };
00050
00051 }

```

## 17.11 hw\_register\_block

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * Copyright (C) 2014-2021, 2023-2024 Kernkonzept GmbH.
00004 * Author(s): Alexander Warg <alexander.warg@kernkonzept.com>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include <l4/sys/types.h>
00011 #include <l4/cxx/type_traits>
00012
00013 namespace L4drivers {
00014
00015
00062
00063
00071 template< unsigned MAX_BITS = 32 >
00072 struct Register_block_base;
00073
00074 template<>
00075 struct Register_block_base<8>
00076 {
00077 virtual l4_uint8_t do_read_8(l4_addr_t reg) const = 0;
00078 virtual void do_write_8(l4_uint8_t value, l4_addr_t reg) = 0;
00079 virtual ~Register_block_base() = 0;
00080 };
00081
00082 inline Register_block_base<8>::~Register_block_base() {}
00083
00084 template<>
00085 struct Register_block_base<16> : Register_block_base<8>
00086 {
00087 virtual l4_uint16_t do_read_16(l4_addr_t reg) const = 0;
00088 virtual void do_write_16(l4_uint16_t value, l4_addr_t reg) = 0;
00089 };
00090
00091 template<>
00092 struct Register_block_base<32> : Register_block_base<16>
00093 {
00094 virtual l4_uint32_t do_read_32(l4_addr_t reg) const = 0;
00095 virtual void do_write_32(l4_uint32_t value, l4_addr_t reg) = 0;
00096 };
00097
00098 template<>

```

```

00099 struct Register_block_base<64> : Register_block_base<32>
00100 {
00101 virtual l4_uint64_t do_read_64(l4_addr_t reg) const = 0;
00102 virtual void do_write_64(l4_uint64_t value, l4_addr_t reg) = 0;
00103 };
00104 #undef REGBLK_READ_TEMPLATE
00105 #undef REGBLK_WRITE_TEMPLATE
00106
00107 template<typename CHIL>
00108 struct Register_block_modify_mixin
00109 {
00110 template< typename T >
00111 T modify(T clear_bits, T set_bits, l4_addr_t reg) const
00112 {
00113 CHIL const *c = static_cast<CHIL const *>(this);
00114 T r = (c->template read<T>(reg) & ~clear_bits) | set_bits;
00115 c->template write<T>(r, reg);
00116 return r;
00117 }
00118
00119 template< typename T >
00120 T set(T set_bits, l4_addr_t reg) const
00121 { return this->template modify<T>(T(0), set_bits, reg); }
00122
00123 template< typename T >
00124 T clear(T clear_bits, l4_addr_t reg) const
00125 { return this->template modify<T>(clear_bits, T(0), reg); }
00126 };
00127
00128 #define REGBLK_READ_TEMPLATE(sz) \
00129 template< typename T > \
00130 typename cxx::enable_if<sizeof(T) == (sz / 8), T>::type read(l4_addr_t reg) const \
00131 { \
00132 union X { T t; l4_uint##sz##_t v; } m; \
00133 m.v = _b->do_read_##sz (reg); \
00134 return m.t; \
00135 }
00136
00137 #define REGBLK_WRITE_TEMPLATE(sz) \
00138 template< typename T > \
00139 void write(T value, l4_addr_t reg, typename cxx::enable_if<sizeof(T) == (sz / 8), T>::type = T()) \
00140 const \
00141 { \
00142 union X { T t; l4_uint##sz##_t v; } m; \
00143 m.t = value; \
00144 _b->do_write_##sz(m.v, reg); \
00145 }
00146
00155 template< typename BLOCK >
00156 class Register_block_tmpl
00157 : public Register_block_modify_mixin<Register_block_tmpl<BLOCK> >
00158 {
00159 private:
00160 BLOCK *_b;
00161
00162 public:
00163 Register_block_tmpl(BLOCK *blk) : _b(blk) {}
00164 Register_block_tmpl() = default;
00165
00166 operator BLOCK * () const { return _b; }
00167
00168 REGBLK_READ_TEMPLATE(8)
00169 REGBLK_WRITE_TEMPLATE(8)
00170 REGBLK_READ_TEMPLATE(16)
00171 REGBLK_WRITE_TEMPLATE(16)
00172 REGBLK_READ_TEMPLATE(32)
00173 REGBLK_WRITE_TEMPLATE(32)
00174 REGBLK_READ_TEMPLATE(64)
00175 REGBLK_WRITE_TEMPLATE(64)
00176 };
00177
00178 #undef REGBLK_READ_TEMPLATE
00179 #undef REGBLK_WRITE_TEMPLATE
00180
00181 namespace __Type_helper {
00182 template<unsigned> struct Unsigned;
00183 template<> struct Unsigned<8> { typedef l4_uint8_t type; };
00184 template<> struct Unsigned<16> { typedef l4_uint16_t type; };
00185 template<> struct Unsigned<32> { typedef l4_uint32_t type; };
00186 template<> struct Unsigned<64> { typedef l4_uint64_t type; };
00187 }
00188
00189 template< unsigned BITS, typename BLOCK >
00200 class Ro_register_tmpl
00201

```

```

00202 {
00203 protected:
00204 BLOCK _b;
00205 unsigned _o;
00206
00207 public:
00208 typedef typename __Type_helper::Unsigned<BITS>::type value_type;
00209
00210 Ro_register_tmpl(BLOCK const &blk, unsigned offset) : _b(blk), _o(offset) {}
00211 Ro_register_tmpl() = default;
00212
00217 operator value_type () const
00218 { return _b.template read<value_type>(_o); }
00219
00224 value_type read() const
00225 { return _b.template read<value_type>(_o); }
00226 };
00227
00228
00236 template< unsigned BITS, typename BLOCK >
00237 class Register_tmpl : public Ro_register_tmpl<BITS, BLOCK>
00238 {
00239 public:
00240 typedef typename Ro_register_tmpl<BITS, BLOCK>::value_type value_type;
00241
00242 Register_tmpl(BLOCK const &blk, unsigned offset)
00243 : Ro_register_tmpl<BITS, BLOCK>(blk, offset)
00244 {}
00245
00246 Register_tmpl() = default;
00247
00252 Register_tmpl &operator = (value_type val)
00253 { this->_b.template write<value_type>(val, this->_o); return *this; }
00254
00259 void write(value_type val)
00260 { this->_b.template write<value_type>(val, this->_o); }
00261
00274 value_type set(value_type set_bits)
00275 { return this->_b.template set<value_type>(set_bits, this->_o); }
00276
00290 value_type clear(value_type clear_bits)
00291 { return this->_b.template clear<value_type>(clear_bits, this->_o); }
00292
00308 value_type modify(value_type clear_bits, value_type set_bits)
00309 { return this->_b.template modify<value_type>(clear_bits, set_bits, this->_o); }
00310 };
00311
00312
00324 template<
00325 unsigned MAX_BITS,
00326 typename BLOCK = Register_block_tmpl<
00327 Register_block_base<MAX_BITS>
00328 >
00329 >
00330 class Register_block
00331 {
00332 private:
00333 template< unsigned B, typename BLK > friend class Register_block;
00334 template< unsigned B, typename BLK > friend class Ro_register_block;
00335 typedef BLOCK Block;
00336 Block _b;
00337
00338 public:
00339 Register_block() = default;
00340 Register_block(Block const &blk) : _b(blk) {}
00341 Register_block &operator = (Block const &blk)
00342 { _b = blk; return *this; }
00343
00344 template< unsigned BITS >
00345 Register_block(Register_block<BITS> blk) : _b(blk._b) {}
00346
00347 typedef Register_tmpl<MAX_BITS, Block> Register;
00348 typedef Ro_register_tmpl<MAX_BITS, Block> Ro_register;
00349
00356 template< unsigned BITS >
00357 Ro_register_tmpl<BITS, Block> r(unsigned offset) const
00358 { return Ro_register_tmpl<BITS, Block>(this->_b, offset); }
00359
00365 Ro_register operator [] (unsigned offset) const
00366 { return this->r<MAX_BITS>(offset); }
00367
00368
00375 template< unsigned BITS >
00376 Register_tmpl<BITS, Block> r(unsigned offset)
00377 { return Register_tmpl<BITS, Block>(this->_b, offset); }
00378
00385 Register operator [] (unsigned offset)

```

```

00386 { return this->r<MAX_BITS>(offset); }
00387 };
00388
00398 template<
00399 unsigned MAX_BITS,
00400 typename BLOCK = Register_block_tmpl<
00401 Register_block_base<MAX_BITS> const
00402 >
00403 >
00404 class Ro_register_block
00405 {
00406 private:
00407 template< unsigned B, typename BLK > friend class Ro_register_block;
00408 typedef BLOCK Block;
00409 Block _b;
00410
00411 public:
00412 Ro_register_block() = default;
00413 Ro_register_block(BLOCK const &blk) : _b(blk) {}
00414
00415 template< unsigned BITS >
00416 Ro_register_block(Register_block<BITS> const &blk) : _b(blk._b) {}
00417
00418 typedef Ro_register_tmpl<MAX_BITS, Block> Ro_register;
00419 typedef Ro_register Register;
00420
00426 Ro_register operator [] (unsigned offset) const
00427 { return Ro_register(this->_b, offset); }
00428
00435 template< unsigned BITS >
00436 Ro_register_tmpl<BITS, Block> r(unsigned offset) const
00437 { return Ro_register_tmpl<BITS, Block>(this->_b, offset); }
00438 };
00439
00440
00454 template< typename BASE, unsigned MAX_BITS = 32 >
00455 struct Register_block_impl;
00456
00457 #define REGBLK_IMPL_RW_TEMPLATE(sz, ...) \
00458 l4_uint##sz##_t do_read_##sz(l4_addr_t reg) const override \
00459 { return static_cast<BASE const *>(this)->template read<l4_uint##sz##_t>(reg); } \
00460 \
00461 void do_write_##sz(l4_uint##sz##_t value, l4_addr_t reg) override \
00462 { static_cast<BASE*>(this)->template write<l4_uint##sz##_t>(value, reg); }
00463
00464
00465 template< typename BASE >
00466 struct Register_block_impl<BASE, 8> : public Register_block_base<8>
00467 {
00468 REGBLK_IMPL_RW_TEMPLATE(8);
00469 };
00470
00471 template< typename BASE >
00472 struct Register_block_impl<BASE, 16> : public Register_block_base<16>
00473 {
00474 REGBLK_IMPL_RW_TEMPLATE(8);
00475 REGBLK_IMPL_RW_TEMPLATE(16);
00476 };
00477
00478 template< typename BASE >
00479 struct Register_block_impl<BASE, 32> : public Register_block_base<32>
00480 {
00481 REGBLK_IMPL_RW_TEMPLATE(8);
00482 REGBLK_IMPL_RW_TEMPLATE(16);
00483 REGBLK_IMPL_RW_TEMPLATE(32);
00484 };
00485
00486 template< typename BASE >
00487 struct Register_block_impl<BASE, 64> : public Register_block_base<64>
00488 {
00489 REGBLK_IMPL_RW_TEMPLATE(8);
00490 REGBLK_IMPL_RW_TEMPLATE(16);
00491 REGBLK_IMPL_RW_TEMPLATE(32);
00492 REGBLK_IMPL_RW_TEMPLATE(64);
00493 };
00494
00495 #undef REGBLK_IMPL_RW_TEMPLATE
00496
00497 }

```

## 17.12 io\_regblock.h

```
00001 /*
```



```

00002 * Copyright (C) 2012 Technische Universität Dresden.
00003 * Author(s): Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 namespace L4
00010 {
00011 class Io_register_block
00012 {
00013 public:
00017 virtual unsigned char read8(unsigned long reg) const = 0;
00018
00022 virtual unsigned short read16(unsigned long reg) const = 0;
00023
00027 virtual unsigned int read32(unsigned long reg) const = 0;
00028
00029 /*
00030 * \brief Read register with an 8 byte access.
00031 */
00032 //virtual unsigned long long read64(unsigned long reg) const = 0;
00033
00037 virtual void write8(unsigned long reg, unsigned char value) const = 0;
00038
00042 virtual void write16(unsigned long reg, unsigned short value) const = 0;
00043
00047 virtual void write32(unsigned long reg, unsigned int value) const = 0;
00048
00049 /*
00050 * \brief Write register with an 8 byte access.
00051 */
00052 //virtual void write64(unsigned long reg, unsigned long long value) const = 0;
00053
00057 virtual unsigned long addr(unsigned long reg) const = 0;
00058
00064 virtual void delay() const = 0;
00065
00066 virtual ~Io_register_block() = 0;
00067
00075 template< typename R >
00076 R read(unsigned long reg) const
00077 {
00078 static_assert(sizeof(R) == 1 || sizeof(R) == 2 || sizeof(R) == 4,
00079 "Invalid size");
00080
00081 switch (sizeof(R))
00082 {
00083 case 1: return read8(reg);
00084 case 2: return read16(reg);
00085 case 4: return read32(reg);
00086 };
00087 }
00088
00096 template< typename R >
00097 void write(unsigned long reg, R value) const
00098 {
00099 static_assert(sizeof(R) == 1 || sizeof(R) == 2 || sizeof(R) == 4,
00100 "Invalid size");
00101
00102 switch (sizeof(R))
00103 {
00104 case 1: write8(reg, value); return;
00105 case 2: write16(reg, value); return;
00106 case 4: write32(reg, value); return;
00107 };
00108 }
00109
00120 template< typename R >
00121 R modify(unsigned long reg, R clear_bits, R set_bits) const
00122 {
00123 R r = (read<R>(reg) & ~clear_bits) | set_bits;
00124 write(reg, r);
00125 return r;
00126 }
00127
00134 template< typename R >
00135 R set(unsigned long reg, R set_bits) const
00136 {
00137 return modify<R>(reg, 0, set_bits);
00138 }
00139
00146 template< typename R >
00147 R clear(unsigned long reg, R clear_bits) const
00148 {
00149 return modify<R>(reg, clear_bits, 0);
00150 }

```

```

00151
00152 };
00153
00154 inline Io_register_block::~Io_register_block() {}
00155
00156
00157 class Io_register_block_mmio : public Io_register_block
00158 {
00159 private:
00160 template< typename R >
00161 R _read(unsigned long reg) const
00162 { return *reinterpret_cast<volatile R *>(_base + (reg « _shift)); }
00163
00164 template< typename R >
00165 void _write(unsigned long reg, R val) const
00166 { *reinterpret_cast<volatile R *>(_base + (reg « _shift)) = val; }
00167
00168 public:
00169 Io_register_block_mmio(unsigned long base, unsigned char shift = 0)
00170 : _base(base), _shift(shift)
00171 {}
00172
00173 unsigned long addr(unsigned long reg) const override
00174 { return _base + (reg « _shift); }
00175
00176 unsigned char read8(unsigned long reg) const override
00177 { return _read<unsigned char>(reg); }
00178 unsigned short read16(unsigned long reg) const override
00179 { return _read<unsigned short>(reg); }
00180 unsigned int read32(unsigned long reg) const override
00181 { return _read<unsigned int>(reg); }
00182
00183 void write8(unsigned long reg, unsigned char val) const override
00184 { _write(reg, val); }
00185 void write16(unsigned long reg, unsigned short val) const override
00186 { _write(reg, val); }
00187 void write32(unsigned long reg, unsigned int val) const override
00188 { _write(reg, val); }
00189
00190 void delay() const override
00191 {}
00192
00193 private:
00194 unsigned long _base;
00195 unsigned char _shift;
00196 };
00197
00198 template<typename ACCESS_TYPE>
00199 class Io_register_block_mmio_fixed_width : public Io_register_block
00200 {
00201 private:
00202 template< typename R >
00203 R _read(unsigned long reg) const
00204 { return *reinterpret_cast<volatile ACCESS_TYPE *>(_base + (reg « _shift)); }
00205
00206 template< typename R >
00207 void _write(unsigned long reg, R val) const
00208 { *reinterpret_cast<volatile ACCESS_TYPE *>(_base + (reg « _shift)) = val; }
00209
00210 public:
00211 Io_register_block_mmio_fixed_width(unsigned long base, unsigned char shift = 0)
00212 : _base(base), _shift(shift)
00213 {}
00214
00215 unsigned long addr(unsigned long reg) const
00216 { return _base + (reg « _shift); }
00217
00218 unsigned char read8(unsigned long reg) const override
00219 { return _read<unsigned char>(reg); }
00220 unsigned short read16(unsigned long reg) const override
00221 { return _read<unsigned short>(reg); }
00222 unsigned int read32(unsigned long reg) const override
00223 { return _read<unsigned int>(reg); }
00224
00225 void write8(unsigned long reg, unsigned char val) const override
00226 { _write(reg, val); }
00227 void write16(unsigned long reg, unsigned short val) const override
00228 { _write(reg, val); }
00229 void write32(unsigned long reg, unsigned int val) const override
00230 { _write(reg, val); }
00231
00232 void delay() const override
00233 {}
00234
00235 private:
00236 unsigned long _base;
00237 unsigned char _shift;

```

```
00238 };
00239 }
```

## 17.13 io\_regblock\_port.h

```
00001 /*
00002 * Copyright (C) 2012 Technische Universität Dresden.
00003 * Author(s): Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include "io_regblock.h"
00010
00011 namespace L4
00012 {
00013 class Io_register_block_port : public Io_register_block
00014 {
00015 public:
00016 Io_register_block_port(unsigned long base)
00017 : _base(base)
00018 {}
00019
00020 unsigned long addr(unsigned long reg) const override { return _base + reg; }
00021
00022 unsigned char read8(unsigned long reg) const override
00023 {
00024 unsigned char val;
00025 asm volatile("inb %w1, %b0" : "=a" (val) : "Nd" (_base + reg));
00026 return val;
00027 }
00028
00029 unsigned short read16(unsigned long reg) const override
00030 {
00031 unsigned short val;
00032 asm volatile("inw %w1, %w0" : "=a" (val) : "Nd" (_base + reg));
00033 return val;
00034 }
00035
00036 unsigned int read32(unsigned long reg) const override
00037 {
00038 unsigned int val;
00039 asm volatile("in %w1, %0" : "=a" (val) : "Nd" (_base + reg));
00040 return val;
00041 }
00042
00043 void write8(unsigned long reg, unsigned char val) const override
00044 { asm volatile("outb %b0, %w1" : : "a" (val), "Nd" (_base + reg)); }
00045
00046 void write16(unsigned long reg, unsigned short val) const override
00047 { asm volatile("outw %w0, %w1" : : "a" (val), "Nd" (_base + reg)); }
00048
00049 void write32(unsigned long reg, unsigned int val) const override
00050 { asm volatile("out %0, %w1" : : "a" (val), "Nd" (_base + reg)); }
00051
00052 void delay() const override
00053 { asm volatile ("outb %al, $0x80"); }
00054
00055 private:
00056 unsigned long _base;
00057 };
00058 }
```

## 17.14 poll\_timeout\_counter.h

```
00001 /*
00002 * Copyright (C) 2012 Technische Universität Dresden.
00003 * Author(s): Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 namespace L4 {
00010
00034 class Poll_timeout_counter
00035 {
```

```

00036 public:
00042 Poll_timeout_counter(unsigned counter_val)
00043 {
00044 set(counter_val);
00045 }
00046
00053 void set(unsigned counter_val)
00054 {
00055 _c = counter_val;
00056 }
00057
00061 bool test(bool expression = true)
00062 {
00063 if (!expression)
00064 return false;
00065
00066 if (_c)
00067 {
00068 --_c;
00069 return true;
00070 }
00071
00072 return false;
00073 }
00074
00081 bool timed_out() const { return _c == 0; }
00082
00083 private:
00084 unsigned _c;
00085 };
00086
00087 }

```

## 17.15 uart\_16550.h

```

00001 /*
00002 * Copyright (C) 2008-2021 Technische Universität Dresden.
00003 * Copyright (C) 2023-2024 Kernkonzept GmbH.
00004 * Author(s): Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00005 * Alexander Warg <alexander.warg@os.inf.tu-dresden.de>
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 #pragma once
00010
00011 #include "uart_base.h"
00012
00013 namespace L4 {
00014
00015 class Uart_16550 : public Uart
00016 {
00017 protected:
00018 enum Registers
00019 {
00020 TRB = 0x00, // Transmit/Receive Buffer (read/write)
00021 BRD_LOW = 0x00, // Baud Rate Divisor LSB if bit 7 of LCR is set (read/write)
00022 IER = 0x01, // Interrupt Enable Register (read/write)
00023 BRD_HIGH = 0x01, // Baud Rate Divisor MSB if bit 7 of LCR is set (read/write)
00024 IIR = 0x02, // Interrupt Identification Register (read only)
00025 FCR = 0x02, // 16550 FIFO Control Register (write only)
00026 LCR = 0x03, // Line Control Register (read/write)
00027 MCR = 0x04, // Modem Control Register (read/write)
00028 LSR = 0x05, // Line Status Register (read only)
00029 MSR = 0x06, // Modem Status Register (read only)
00030 SPR = 0x07, // Scratch Pad Register (read/write)
00031 };
00032
00033 enum Register_value_iir
00034 {
00035 IIR_BUSY = 7,
00036 };
00037
00038 enum Register_value_lsr
00039 {
00040 LSR_DR = 0x01, // Receiver data ready
00041 LSR_THRE = 0x20, // Transmit hold register empty
00042 LSR_TSRE = 0x40, // Transmitter empty
00043 };
00044
00045 enum Init_values
00046 {
00047 #ifndef UART_16550_INIT_MCR
00048 Init_mcr = UART_16550_INIT_MCR,

```

```

00049 #else
00050 Init_mcr = 0,
00051 #endif
00052 #ifdef UART_16550_INIT_IER
00053 Init_ier = UART_16550_INIT_IER,
00054 #else
00055 Init_ier = 0,
00056 #endif
00057 #ifdef UART_16550_INIT_FCR
00058 Init_fcr = UART_16550_INIT_FCR,
00059 #else
00060 Init_fcr = 0,
00061 #endif
00062 };
00063
00064 public:
00065 enum
00066 {
00067 PAR_NONE = 0x00,
00068 PAR_EVEN = 0x18,
00069 PAR_ODD = 0x08,
00070 DAT_5 = 0x00,
00071 DAT_6 = 0x01,
00072 DAT_7 = 0x02,
00073 DAT_8 = 0x03,
00074 STOP_1 = 0x00,
00075 STOP_2 = 0x04,
00076
00077 MODE_8N1 = PAR_NONE | DAT_8 | STOP_1,
00078 MODE_7E1 = PAR_EVEN | DAT_7 | STOP_1,
00079
00080 // these two values are to leave either mode
00081 // or baud rate unchanged on a call to change_mode
00082 MODE_NC = 0x1000000,
00083 BAUD_NC = 0x1000000,
00084
00085 Base_rate_x86 = 115200,
00086 Base_rate_pxa = 921600,
00087 };
00088
00089 explicit Uart_16550(unsigned long base_rate, unsigned char init_flags = 0,
00090 unsigned char ier_bits = Init_ier,
00091 unsigned char mcr_bits = Init_mcr,
00092 unsigned char fcr_bits = Init_fcr)
00093 : _base_rate(base_rate), _init_flags(init_flags), _mcr_bits(mcr_bits),
00094 _ier_bits(ier_bits), _fcr_bits(fcr_bits)
00095 {}
00096
00097 bool startup(Io_register_block const *regs) override;
00098 void shutdown() override;
00099 bool change_mode(Transfer_mode m, Baud_rate r) override;
00100 int tx_avail() const;
00101 void wait_tx_done() const;
00102 inline void out_char(char c) const;
00103 int write(char const *s, unsigned long count,
00104 bool blocking = true) const override;
00105
00106 bool enable_rx_irq(bool enable = true) override;
00107 int char_avail() const override;
00108 int get_char(bool blocking = true) const override;
00109
00110 private:
00111 unsigned long _base_rate;
00112 unsigned char _init_flags;
00113 unsigned char _mcr_bits;
00114 unsigned char _ier_bits;
00115 unsigned char _fcr_bits;
00116 };
00117
00118 } // namespace L4

```

## 17.16 uart\_16550\_dw.h

```

00001 /*
00002 * Copyright (C) 2015 Technische Universität Dresden.
00003 * Copyright (C) 2023-2024 Kernkonzept GmbH.
00004 * Author(s): Adam Lackorzynski <adam@l4re.org>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include "uart_16550.h"

```

```

00011
00012 namespace L4 {
00013
00014 class Uart_16550_dw : public Uart_16550
00015 {
00016 public:
00017 explicit Uart_16550_dw(unsigned long base_rate)
00018 : Uart_16550(base_rate)
00019 {}
00020
00021 void irq_ack() override;
00022 };
00023
00024 } // namespace L4

```

## 17.17 uart\_apb.h

```

00001 /*
00002 * Copyright (C) 2017, 2019, 2023-2024 Kernkonzept GmbH.
00003 * Author(s): Georg Kotheimer <georg.kotheimer@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include "uart_base.h"
00010
00011 namespace L4 {
00012
00017 class Uart_apb : public Uart
00018 {
00019 public:
00021 Uart_apb(unsigned freq) : _freq(freq) {}
00022 bool startup(Io_register_block const *) override;
00023 void shutdown() override;
00024 bool change_mode(Transfer_mode m, Baud_rate r) override;
00025 int tx_avail() const;
00026 void wait_tx_done() const;
00027 inline void out_char(char c) const;
00028 int write(char const *s, unsigned long count,
00029 bool blocking = true) const override;
00030
00031 bool enable_rx_irq(bool enable) override;
00032 int char_avail() const override;
00033 int get_char(bool blocking = true) const override;
00034
00035 private:
00036 void set_rate(Baud_rate r);
00037 unsigned _freq;
00038 };
00039
00040 } // namespace L4

```

## 17.18 uart\_base.h

```

00001 /*
00002 * Copyright (C) 2009-2012 Technische Universität Dresden.
00003 * Copyright (C) 2023-2024 Kernkonzept GmbH.
00004 * Author(s): Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include <stddef.h>
00011 #include <l4/drivers/io_regblock.h>
00012
00013 #include "poll_timeout_counter.h"
00014
00015 namespace L4 {
00016
00020 class Uart
00021 {
00022 protected:
00023 unsigned _mode;
00024 unsigned _rate;
00025 Io_register_block const *_regs;
00026

```

```

00027 public:
00028 void *operator new (size_t, void* a)
00029 { return a; }
00030
00031 public:
00032 typedef unsigned Transfer_mode;
00033 typedef unsigned Baud_rate;
00034
00035 Uart()
00036 : _mode(~0U), _rate(~0U)
00037 {}
00038
00046 virtual bool startup(Io_register_block const *regs) = 0;
00047
00048 virtual ~Uart() {}
00049
00053 virtual void shutdown() = 0;
00054
00066 virtual bool change_mode(Transfer_mode m, Baud_rate r) = 0;
00067
00078 virtual int write(char const *s, unsigned long count,
00079 bool blocking = true) const = 0;
00080
00086 Transfer_mode mode() const { return _mode; }
00087
00093 Baud_rate rate() const { return _rate; }
00094
00095
00103 virtual bool enable_rx_irq(bool = true) { return false; }
00104
00108 virtual void irq_ack() {}
00109
00116 virtual int char_avail() const = 0;
00117
00126 virtual int get_char(bool blocking = true) const = 0;
00127
00128 protected:
00140 template <typename Uart_driver, bool Timeout_guard = true>
00141 int generic_write(char const *s, unsigned long count,
00142 bool blocking = true) const
00143 {
00144 auto *self = static_cast<Uart_driver const*>(this);
00145
00146 unsigned long c;
00147 for (c = 0; c < count; ++c)
00148 {
00149 if (!blocking && !self->tx_avail())
00150 break;
00151
00152 if constexpr (Timeout_guard)
00153 {
00154 Poll_timeout_counter i(3000000);
00155 while (i.test(!self->tx_avail()))
00156 ;
00157 }
00158 else
00159 {
00160 while (!self->tx_avail())
00161 ;
00162 }
00163
00164 self->out_char(*s++);
00165 }
00166
00167 if (blocking)
00168 self->wait_tx_done();
00169
00170 return c;
00171 }
00172 };
00173
00174 } // namespace L4

```

## 17.19 uart\_cadence.h

```

00001 /*
00002 * Copyright (C) 2013 Technische Universität Dresden.
00003 * Copyright (C) 2023-2024 Kernkonzept GmbH.
00004 * Author(s): Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once

```

```

00009
00010 #include "uart_base.h"
00011
00012 namespace L4 {
00013
00014 class Uart_cadence : public Uart
00015 {
00016 public:
00017 explicit Uart_cadence(unsigned base_rate) : _base_rate(base_rate) {}
00018 bool startup(Io_register_block const *) override;
00019 void shutdown() override;
00020 bool change_mode(Transfer_mode m, Baud_rate r) override;
00021 int tx_avail() const;
00022 void wait_tx_done() const {}
00023 inline void out_char(char c) const;
00024 int write(char const *s, unsigned long count,
00025 bool blocking = true) const override;
00026
00027 bool enable_rx_irq(bool) override;
00028 void irq_ack() override;
00029 int char_avail() const override;
00030 int get_char(bool blocking = true) const override;
00031
00032 private:
00033 unsigned _base_rate;
00034 };
00035
00036 } // namespace L4

```

## 17.20 uart\_dcc-v6.h

```

00001 /*
00002 * Copyright (C) 2009 Technische Universität Dresden.
00003 * Copyright (C) 2023-2024 Kernkonzept GmbH.
00004 * Author(s): Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include "uart_base.h"
00011
00012 namespace L4 {
00013
00014 class Uart_dcc_v6 : public Uart
00015 {
00016 public:
00017 explicit Uart_dcc_v6() {}
00018 explicit Uart_dcc_v6(unsigned /*base_rate*/) {}
00019 bool startup(Io_register_block const *) override;
00020 void shutdown() override;
00021 bool change_mode(Transfer_mode m, Baud_rate r) override;
00022 int tx_avail() const;
00023 void wait_tx_done() const;
00024 inline void out_char(char c) const;
00025 int write(char const *s, unsigned long count,
00026 bool blocking = true) const override;
00027
00028 int char_avail() const override;
00029 int get_char(bool blocking = true) const override;
00030
00031 private:
00032 unsigned get_status() const;
00033 };
00034
00035 } // namespace L4

```

## 17.21 uart\_dm.h

```

00001 /*
00002 * Copyright (C) 2021-2022 Stephan Gerhold <stephan@gerhold.net>
00003 * Copyright (C) 2022-2024 Kernkonzept GmbH.
00004 * Author(s): Stephan Gerhold <stephan@gerhold.net>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009

```



```

00010 #include "uart_base.h"
00011
00012 namespace L4 {
00013
00014 class Uart_dm : public Uart
00015 {
00016 public:
00017 explicit Uart_dm(unsigned /*base_rate*/) {}
00018 bool startup(Io_register_block const *) override;
00019 void shutdown() override;
00020 bool change_mode(Transfer_mode m, Baud_rate r) override;
00021 int tx_avail() const;
00022 void wait_tx_done() const;
00023 inline void out_char(char c) const;
00024 int write(char const *s, unsigned long count,
00025 bool blocking = true) const override;
00026
00027 bool enable_rx_irq(bool enable = true) override;
00028 int char_avail() const override;
00029 int get_char(bool blocking = true) const override;
00030 };
00031
00032 } // namespace L4

```

## 17.22 uart\_dummy.h

```

00001 /*
00002 * Copyright 2009 Technische Universität Dresden.
00003 * Copyright (C) 2023-2024 Kernkonzept GmbH.
00004 * Author(s): Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include "uart_base.h"
00011
00012 namespace L4 {
00013
00014 class Uart_dummy : public Uart
00015 {
00016 public:
00017 explicit Uart_dummy() {}
00018 explicit Uart_dummy(unsigned /*base_rate*/) {}
00019 bool startup(Io_register_block const *) override { return true; }
00020 void shutdown() override {}
00021 bool change_mode(Transfer_mode, Baud_rate) override { return true; }
00022 inline void out_char(char /*ch*/) const {}
00023 int write(char const * /*str*/, unsigned long /*count*/,
00024 bool /*blocking*/ = true) const override
00025 { return 0; }
00026
00027 int char_avail() const override { return false; }
00028 int get_char(bool /*blocking*/ = true) const override { return 0; }
00029 };
00030
00031 } // namespace L4

```

## 17.23 uart\_geni.h

```

00001 /*
00002 * Copyright (C) 2022-2024 Kernkonzept GmbH.
00003 * Author(s): Stephan Gerhold <stephan.gerhold@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include "uart_base.h"
00010
00011 namespace L4 {
00012
00013 class Uart_geni : public Uart
00014 {
00015 public:
00016 explicit Uart_geni(unsigned /*base_rate*/) {}
00017 bool startup(Io_register_block const *) override;
00018 void shutdown() override;

```

```

00019 bool change_mode(Transfer_mode m, Baud_rate r) override;
00020 int tx_avail() const;
00021 void wait_tx_done() const;
00022 void out_char(char c) const;
00023 int write(char const *s, unsigned long count,
00024 bool blocking = true) const override;
00025
00026 bool enable_rx_irq(bool enable = true) override;
00027 void irq_ack() override;
00028 int char_avail() const override;
00029 int get_char(bool blocking = true) const override;
00030 };
00031
00032 } // namespace L4

```

## 17.24 uart\_imx.h

```

00001 /*
00002 * Copyright (C) 2008-2009 Technische Universität Dresden.
00003 * Copyright (C) 2023-2024 Kernkonzept GmbH.
00004 * Author(s): Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include "uart_base.h"
00011
00012 namespace L4 {
00013
00014 class Uart_imx : public Uart
00015 {
00016 public:
00017 enum platform_type
00018 {
00019 Type_imx21,
00020 Type_imx35,
00021 Type_imx51,
00022 Type_imx6,
00023 Type_imx7,
00024 Type_imx8,
00025 };
00026 explicit Uart_imx(enum platform_type type) : _type(type) {}
00027 explicit Uart_imx(enum platform_type type, unsigned /*base_rate*/)
00028 : _type(type) {}
00029 bool startup(Io_register_block const *) override;
00030 void shutdown() override;
00031 bool change_mode(Transfer_mode m, Baud_rate r) override;
00032 int tx_avail() const;
00033 void wait_tx_done() const;
00034 inline void out_char(char c) const;
00035 int write(char const *s, unsigned long count,
00036 bool blocking = true) const override;
00037
00038 bool enable_rx_irq(bool enable = true) override;
00039 int char_avail() const override;
00040 int get_char(bool blocking = true) const override;
00041
00042 private:
00043 enum platform_type _type;
00044 };
00045
00046 class Uart_imx21 : public Uart_imx
00047 {
00048 public:
00049 Uart_imx21() : Uart_imx(Type_imx21) {}
00050 explicit Uart_imx21(unsigned base_rate) : Uart_imx(Type_imx21, base_rate) {}
00051 };
00052
00053 class Uart_imx35 : public Uart_imx
00054 {
00055 public:
00056 Uart_imx35() : Uart_imx(Type_imx35) {}
00057 explicit Uart_imx35(unsigned base_rate) : Uart_imx(Type_imx35, base_rate) {}
00058 };
00059
00060 class Uart_imx51 : public Uart_imx
00061 {
00062 public:
00063 Uart_imx51() : Uart_imx(Type_imx51) {}
00064 explicit Uart_imx51(unsigned base_rate) : Uart_imx(Type_imx51, base_rate) {}
00065 };
00066

```

```

00067 class Uart_imx6 : public Uart_imx
00068 {
00069 public:
00070 Uart_imx6() : Uart_imx(Type_imx6) {}
00071 explicit Uart_imx6(unsigned base_rate) : Uart_imx(Type_imx6, base_rate) {}
00072
00073 void irq_ack() override;
00074 };
00075
00076 class Uart_imx7 : public Uart_imx
00077 {
00078 public:
00079 Uart_imx7() : Uart_imx(Type_imx7) {}
00080 explicit Uart_imx7(unsigned base_rate) : Uart_imx(Type_imx7, base_rate) {}
00081 };
00082
00083 class Uart_imx8 : public Uart_imx
00084 {
00085 public:
00086 Uart_imx8() : Uart_imx(Type_imx8) {}
00087 explicit Uart_imx8(unsigned base_rate) : Uart_imx(Type_imx8, base_rate) {}
00088 };
00089
00090 } // namespace L4

```

## 17.25 uart\_leon3.h

```

00001 /*
00002 * Copyright (C) 2011 Technische Universität Dresden.
00003 * Copyright (C) 2023-2024 Kernkonzept GmbH.
00004 * Author(s): Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00005 * Björn Döbel <doebel@os.inf.tu-dresden.de>
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 #pragma once
00010
00011 #include "uart_base.h"
00012
00013 namespace L4 {
00014
00015 class Uart_leon3 : public Uart
00016 {
00017 public:
00018 explicit Uart_leon3() {}
00019 explicit Uart_leon3(unsigned /*base_rate*/) {}
00020 bool startup(Io_register_block const *) override;
00021 void shutdown() override;
00022 bool change_mode(Transfer_mode m, Baud_rate r) override;
00023 int tx_avail() const;
00024 void wait_tx_done() const;
00025 inline void out_char(char c) const;
00026 int write(char const *s, unsigned long count,
00027 bool blocking = true) const override;
00028
00029 bool enable_rx_irq(bool = true) override;
00030 int char_avail() const override;
00031 int get_char(bool blocking = true) const override;
00032 };
00033
00034 } // namespace L4

```

## 17.26 uart\_linflex.h

```

00001 /*
00002 * Copyright (C) 2018 Technische Universität Dresden.
00003 * Copyright (C) 2023-2024 Kernkonzept GmbH.
00004 * Author(s): Adam Lackorzynski <adam@l4re.org>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include "uart_base.h"
00011
00012 namespace L4 {
00013
00014 class Uart_linflex : public Uart

```

```

00015 {
00016 public:
00017 explicit Uart_linflex(unsigned) {}
00018 bool startup(Io_register_block const *) override;
00019 void shutdown() override;
00020 bool change_mode(Transfer_mode m, Baud_rate r) override;
00021 int tx_avail() const;
00022 void wait_tx_done() const;
00023 inline void out_char(char c) const;
00024 int write(char const *s, unsigned long count,
00025 bool blocking = true) const override;
00026
00027 bool enable_rx_irq(bool enable = true) override;
00028 int char_avail() const override;
00029 int get_char(bool blocking = true) const override;
00030
00031 private:
00032 void set_uartcr(bool fifo);
00033 bool is_tx_fifo_enabled() const;
00034 bool is_rx_fifo_enabled() const;
00035 };
00036
00037 } // namespace L4

```

## 17.27 uart\_lpuart.h

```

00001 /*
00002 * Copyright (C) 2019, 2023-2024 Kernkonzept GmbH.
00003 * Author(s): Adam Lackorzynski <adam@l4re.org>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include "uart_base.h"
00010
00011 namespace L4 {
00012
00013 class Uart_lpuart : public Uart
00014 {
00015 public:
00016 Uart_lpuart(unsigned freq = 0) : _freq(freq) {}
00017 bool startup(Io_register_block const *) override;
00018 void shutdown() override;
00019 bool change_mode(Transfer_mode m, Baud_rate r) override;
00020 int tx_avail() const;
00021 void wait_tx_done() const {}
00022 inline void out_char(char c) const;
00023 int write(char const *s, unsigned long count,
00024 bool blocking = true) const override;
00025
00026 bool enable_rx_irq(bool enable = true) override;
00027 int char_avail() const override;
00028 int get_char(bool blocking = true) const override;
00029
00030 private:
00031 unsigned _freq;
00032 };
00033
00034 } // namespace L4

```

## 17.28 uart\_mvebu.h

```

00001 /*
00002 * Copyright (C) 2017, 2023-2024 Kernkonzept GmbH.
00003 * Author(s): Adam Lackorzynski <adam@l4re.org>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include "uart_base.h"
00010
00011 namespace L4 {
00012
00013 class Uart_mvebu : public Uart
00014 {
00015 public:

```

```

00016 explicit Uart_mvebu(unsigned baserate) : _baserate(baserate) {}
00017 bool startup(Io_register_block const *) override;
00018 void shutdown() override;
00019 bool change_mode(Transfer_mode m, Baud_rate r) override;
00020 int tx_avail() const;
00021 void wait_tx_done() const {}
00022 inline void out_char(char c) const;
00023 int write(char const *s, unsigned long count,
00024 bool blocking = true) const override;
00025
00026 bool enable_rx_irq(bool enable = true) override;
00027 int char_avail() const override;
00028 int get_char(bool blocking = true) const override;
00029
00030 private:
00031 unsigned _baserate;
00032 };
00033
00034 } // namespace L4

```

## 17.29 uart\_of.h

```

00001 /*
00002 * Copyright (C) 2009 Technische Universität Dresden.
00003 * Copyright (C) 2023-2024 Kernkonzept GmbH.
00004 * Author(s): Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include "uart_base.h"
00011 #include <stdarg.h>
00012 #include <string.h>
00013 #include <l4/drivers/of.h>
00014
00015 namespace L4 {
00016
00017 class Uart_of : public Uart, public L4_drivers::Of
00018 {
00019 private:
00020 ihandle_t _serial;
00021
00022 public:
00023 Uart_of() : Of(), _serial(0) {}
00024 explicit Uart_of(unsigned /*base_rate*/) : Of(), _serial(0) {}
00025 bool startup(Io_register_block const *) override;
00026 void shutdown() override;
00027 bool change_mode(Transfer_mode m, Baud_rate r) override;
00028 int tx_avail() const;
00029 void out_char(char c) const;
00030 int write(char const *s, unsigned long count,
00031 bool blocking = true) const override;
00032
00033 int char_avail() const override;
00034 int get_char(bool blocking = true) const override;
00035 };
00036
00037 } // namespace L4

```

## 17.30 uart\_omap35x.h

```

00001 /*
00002 * Copyright (C) 2009 Technische Universität Dresden.
00003 * Copyright (C) 2023-2024 Kernkonzept GmbH.
00004 * Author(s): Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include "uart_base.h"
00011
00012 namespace L4 {
00013
00014 class Uart_omap35x : public Uart
00015 {
00016 public:

```

```

00017 explicit Uart_omap35x() {}
00018 explicit Uart_omap35x(unsigned /*base_rate*/) {}
00019 bool startup(Io_register_block const *) override;
00020 void shutdown() override;
00021 bool change_mode(Transfer_mode m, Baud_rate r) override;
00022 int tx_avail() const;
00023 void wait_tx_done() const;
00024 inline void out_char(char c) const;
00025 int write(char const *s, unsigned long count,
00026 bool blocking = true) const override;
00027
00028 bool enable_rx_irq(bool) override;
00029 int char_avail() const override;
00030 int get_char(bool blocking = true) const override;
00031 };
00032
00033 } // namespace L4

```

## 17.31 uart\_pl011.h

```

00001 /*
00002 * Copyright (C) 2009 Technische Universität Dresden.
00003 * Copyright (C) 2023-2024 Kernkonzept GmbH.
00004 * Author(s): Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include "uart_base.h"
00011
00012 namespace L4 {
00013
00014 class Uart_pl011 : public Uart
00015 {
00016 public:
00017 Uart_pl011(unsigned freq) : _freq(freq) {}
00018 bool startup(Io_register_block const *) override;
00019 void shutdown() override;
00020 bool change_mode(Transfer_mode m, Baud_rate r) override;
00021 int tx_avail() const;
00022 void wait_tx_done() const;
00023 inline void out_char(char c) const;
00024 int write(char const *s, unsigned long count,
00025 bool blocking = true) const override;
00026
00027 bool enable_rx_irq(bool enable) override;
00028 int char_avail() const override;
00029 int get_char(bool blocking = true) const override;
00030
00031 private:
00032 void set_rate(Baud_rate r);
00033 unsigned _freq;
00034 };
00035
00036 } // namespace L4

```

## 17.32 uart\_s3c2410.h

```

00001 /*
00002 * Copyright (C) 2009-2012 Technische Universität Dresden.
00003 * Copyright (C) 2023-2024 Kernkonzept GmbH.
00004 * Author(s): Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include "uart_base.h"
00011
00012 namespace L4 {
00013
00014 class Uart_s3c : public Uart
00015 {
00016 protected:
00017 enum Uart_type
00018 {
00019 Type_24xx, Type_64xx, Type_s5pv210,

```

```

00020 };
00021
00022 Uart_type type() const { return _type; }
00023
00024 public:
00025 explicit Uart_s3c(Uart_type type) : _type(type) {}
00026 explicit Uart_s3c(Uart_type type, unsigned /*base_rate*/) : _type(type) {}
00027 bool startup(Io_register_block const *) override;
00028 void shutdown() override;
00029 bool change_mode(Transfer_mode m, Baud_rate r) override;
00030 int tx_avail() const;
00031 void wait_tx_done() const;
00032 inline void out_char(char c) const;
00033 int write(char const *s, unsigned long count,
00034 bool blocking = true) const override;
00035 void fifo_reset();
00036
00037 int char_avail() const override;
00038 int get_char(bool blocking = true) const override;
00039
00040 protected:
00041 virtual void ack_rx_irq() const = 0;
00042 virtual void wait_for_empty_tx_fifo() const = 0;
00043 virtual unsigned is_rx_fifo_non_empty() const = 0;
00044 virtual unsigned is_tx_fifo_not_full() const = 0;
00045
00046 private:
00047 Uart_type _type;
00048 };
00049
00050 class Uart_s3c2410 : public Uart_s3c
00051 {
00052 public:
00053 Uart_s3c2410() : Uart_s3c(Type_24xx) {}
00054 explicit Uart_s3c2410(unsigned base_rate) : Uart_s3c(Type_24xx, base_rate) {}
00055
00056 protected:
00057 void ack_rx_irq() const override {}
00058 void wait_for_empty_tx_fifo() const override;
00059 unsigned is_rx_fifo_non_empty() const override;
00060 unsigned is_tx_fifo_not_full() const override;
00061
00062 void auto_flow_control(bool on);
00063 };
00064
00065 class Uart_s3c64xx : public Uart_s3c
00066 {
00067 public:
00068 Uart_s3c64xx() : Uart_s3c(Type_64xx) {}
00069 explicit Uart_s3c64xx(unsigned base_rate) : Uart_s3c(Type_64xx, base_rate) {}
00070
00071 protected:
00072 void ack_rx_irq() const override;
00073 void wait_for_empty_tx_fifo() const override;
00074 unsigned is_rx_fifo_non_empty() const override;
00075 unsigned is_tx_fifo_not_full() const override;
00076 };
00077
00078 class Uart_s5pv210 : public Uart_s3c
00079 {
00080 public:
00081 Uart_s5pv210() : Uart_s3c(Type_s5pv210) {}
00082 explicit Uart_s5pv210(unsigned base_rate) : Uart_s3c(Type_s5pv210, base_rate) {}
00083
00084 protected:
00085 void ack_rx_irq() const override;
00086 void wait_for_empty_tx_fifo() const override;
00087 unsigned is_rx_fifo_non_empty() const override;
00088 unsigned is_tx_fifo_not_full() const override;
00089 };
00090
00091 } // namespace L4

```

## 17.33 uart\_sa1000.h

```

00001 /*
00002 * Copyright (C) 2008-2012 Technische Universität Dresden.
00003 * Copyright (C) 2023-2024 Kernkonzept GmbH.
00004 * Author(s): Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00005 * Alexander Warg <alexander.warg@os.inf.tu-dresden.de>
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */

```

```

00009 #pragma once
00010
00011 #include "uart_base.h"
00012
00013 namespace L4 {
00014
00015 class Uart_sal000 : public Uart
00016 {
00017 public:
00018 explicit Uart_sal000() {}
00019 explicit Uart_sal000(unsigned /*base_rate*/) {}
00020 bool startup(Io_register_block const *) override;
00021 void shutdown() override;
00022 bool change_mode(Transfer_mode m, Baud_rate r) override;
00023 int tx_avail() const;
00024 void wait_tx_done() const;
00025 inline void out_char(char c) const;
00026 int write(char const *s, unsigned long count,
00027 bool blocking = true) const override;
00028
00029 int char_avail() const override;
00030 int get_char(bool blocking = true) const override;
00031 };
00032
00033 } // namespace L4

```

## 17.34 uart\_sbi.h

```

00001 /*
00002 * Copyright (C) 2021, 2024 Kernkonzept GmbH.
00003 * Author(s): Georg Kotheimer <georg.kotheimer@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include "uart_base.h"
00010
00011 namespace L4 {
00012
00013 class Uart_sbi : public Uart
00014 {
00015 public:
00016 Uart_sbi();
00017 explicit Uart_sbi(unsigned /*base_rate*/) : Uart_sbi() {}
00018 bool startup(Io_register_block const *) override;
00019 void shutdown() override;
00020 bool change_mode(Transfer_mode m, Baud_rate r) override;
00021 int tx_avail() const { return true; }
00022 void wait_tx_done() const {}
00023 inline void out_char(char c) const;
00024 int write(char const *s, unsigned long count,
00025 bool blocking = true) const override;
00026
00027 int char_avail() const override;
00028 int get_char(bool blocking = true) const override;
00029
00030 private:
00031 mutable int _bufchar;
00032 };
00033
00034 } // namespace L4

```

## 17.35 uart\_sh.h

```

00001 /*
00002 * Copyright (C) 2016 Technische Universität Dresden.
00003 * Copyright (C) 2023-2024 Kernkonzept GmbH.
00004 * Author(s): Adam Lackorzynski <adam@l4re.org>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include "uart_base.h"
00011
00012 namespace L4 {
00013

```



```

00014 class Uart_sh : public Uart
00015 {
00016 public:
00017 explicit Uart_sh() {}
00018 explicit Uart_sh(unsigned /*base_rate*/) {}
00019 bool startup(Io_register_block const *) override;
00020 void shutdown() override;
00021 bool change_mode(Transfer_mode m, Baud_rate r) override;
00022 int tx_avail() const;
00023 void wait_tx_done() const {}
00024 inline void out_char(char c) const;
00025 int write(char const *s, unsigned long count,
00026 bool blocking = true) const override;
00027
00028 bool enable_rx_irq(bool enable = true) override;
00029 void irq_ack() override;
00030 int char_avail() const override;
00031 int get_char(bool blocking = true) const override;
00032 };
00033
00034 } // namespace L4

```

## 17.36 uart\_sifive.h

```

00001 /*
00002 * Copyright (C) 2021-2024 Kernkonzept GmbH.
00003 * Author(s): Georg Kotheimer <georg.kotheimer@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include "uart_base.h"
00010
00011 namespace L4 {
00012
00013 class Uart_sifive : public Uart
00014 {
00015 public:
00016 Uart_sifive(unsigned freq) : _freq(freq), _bufchar(-1) {}
00017 bool startup(Io_register_block const *) override;
00018 void shutdown() override;
00019 bool change_mode(Transfer_mode m, Baud_rate r) override;
00020 int tx_avail() const;
00021 void wait_tx_done() const;
00022 inline void out_char(char c) const;
00023 int write(char const *s, unsigned long count,
00024 bool blocking = true) const override;
00025
00026 bool enable_rx_irq(bool enable) override;
00027 int char_avail() const override;
00028 int get_char(bool blocking = true) const override;
00029
00030 private:
00031 unsigned _freq;
00032 mutable int _bufchar;
00033 };
00034
00035 } // namespace L4

```

## 17.37 uart\_tegra-tcu.h

```

00001 /*
00002 * Copyright (C) 2025 Kernkonzept GmbH.
00003 * Author(s): Adam Lackorzynski <adam@l4re.org>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include "uart_base.h"
00010
00011 namespace L4 {
00012
00013 class Uart_tegra_tcu : public Uart
00014 {
00015 public:
00016 Uart_tegra_tcu() {}

```

```

00017 Uart_tegra_tcu(unsigned long) {}
00018 bool startup(Io_register_block const *tx_mbox) override;
00019 void set_rx_mbox(Io_register_block const *rx_mbox);
00020 void shutdown() override;
00021 bool change_mode(Transfer_mode m, Baud_rate r) override;
00022 int tx_avail() const;
00023 void wait_tx_done() const {}
00024 inline void out_char(char c) const;
00025 int write(char const *s, unsigned long count,
00026 bool blocking = true) const override;
00027
00028 int char_avail() const override;
00029 int get_char(bool blocking = true) const override;
00030
00031 private:
00032 mutable unsigned _current_rx_val = 0u;
00033 Io_register_block const *_rx_mbox = nullptr;
00034 };
00035
00036 } // namespace L4

```

## 17.38 tutorial.lua

```

00001 local _doc = [=[
00002
00003 Tutorial lua script for Ned
00004 =====
00005
00006 Firstly we have a set of definitions available. Some come from 'ned.lua'
00007 embedded script and others from the C++ bindings within Ned. The whole L4
00008 functionality is in the lua module "L4" (use 'local L4 = require("L4");').
00009 The L4 module classes and functions cope with L4 capabilities and
00010 their invocations, provide a set of constants and access to the L4Re environment of
00011 the running program. Finally, of course it can also start L4 applications.
00012
00013 L4 Capabilities
00014 =====
00015
00016 The L4 module defines a user data type for capabilities. A capability in lua
00017 carries a typed L4 capability and is accompanied with a set of type specific
00018 methods that may be called on the object behind the capability. There also
00019 exists a way to cast a capability to a capability to a different type of
00020 object using L4.cast(type, cap).
00021
00022 L4.cast(type, cap)
00023
00024 Returns a cap transformed to a capability of the given type, whereas type
00025 is either the fully qualified C++ name of the class encapsulating the object
00026 or the L4 protocol ID assigned to all L4Re and L4 system objects.
00027 If the type is unknown then nil is returned.
00028
00029 Generic capabilities provide the following methods:
00030
00031 is_valid()
00032
00033 Returns true if the capability is not the invalid cap (L4_INVALID_CAP), and
00034 false if it is the invalid cap.
00035
00036
00037 L4Re::Namespace object
00038 =====
00039
00040 There is a lua type for name spaces that has the following methods:
00041
00042 query(name), or q(name)
00043
00044 Returns a closure (function) that initiates a query for the given name
00045 within the name space. The function takes no arguments and returns
00046 a capability if successful or nil if name is not found.
00047
00048
00049 link(name), or l(name)
00050
00051 Returns a function that can create a link to the given object in the name
00052 space if put into another name space. The function takes two parameters
00053 the target name space and the name in the target name space.
00054 Loader:create_namespace and Loader:fill_namespace calls this function
00055 when things are really put into an L4Re::Namespace.
00056
00057
00058 register(name, cap), or r(name, cap)
00059
00060 Registers the given object capability under the given name. cap can

```

```

00061 be either a real capability (note query returns a function), a string, or
00062 nil. If it is a capability it is just put into the name space.
00063 In the case cap is a string a placeholder will be put into the name space
00064 that will be replaced with a real capability later by some program.
00065 And if nil is use the name will be deleted from the name space.
00066
00067
00068 L4::Factory object
00069 =====
00070
00071 The factory object provides an interface to the generic create method of a
00072 factory.
00073
00074 create(proto, ...)
00075
00076 This method calls the factory to create an object of the given type,
00077 via the L4 protocol number (see L4.Proto table for more) all further
00078 arguments are passed to the factory.
00079
00080
00081 Access to the L4Re Env capabilities
00082 =====
00083
00084 The L4 module defines a table L4.Env that contains the capabilities
00085 of the L4Re::Env::env() environment. Namely:
00086
00087 factory The kernel factory of Ned
00088 log The log object of Ned
00089 user_factory The factory provided to Ned, including memory
00090 parent The parent of Ned
00091 rm The region map of Ned
00092 scheduler The scheduler of Ned
00093
00094
00095 Some useful constants
00096 =====
00097
00098 L4.Proto table contains the most important protocol values for
00099 L4 and L4Re objects.
00100
00101 Namespace
00102 Goos
00103 Rm
00104 Irq
00105 Sigma0
00106 Factory
00107 Log
00108 Scheduler
00109
00110 The L4.Info table contains the following functions:
00111
00112 Kip.str() The banner string found in the kernel info page
00113 arch() Architecture name, such as: x86, amd64, arm, ppc32
00114 platform() Platform name, such as: pc, ux, realview, beagleboard
00115 mword_bits() Number of native machine word bits (32, 64)
00116
00117
00118 Support for starting L4 programs
00119 =====
00120
00121 The L4 module defines two classes that are useful for starting L4 applications.
00122 The class L4.Loader that encapsulates a fairly high level policy
00123 that is useful for starting a whole set of processes. And the class L4.App_env
00124 that encapsulates a more fine-grained policy.
00125
00126 L4.Loader
00127 -----
00128
00129 The class L4.Loader encapsulates the policy for starting programs with the
00130 basic building blocks for the application coming from a dedicated loader,
00131 such as Moe or a Loader instance. These building blocks are a region map (Rm),
00132 a scheduler, a memory allocator, and a logging facility.
00133 A L4.Loader object is typically used to start multiple applications. There
00134 is a L4.default_loader instance of L4.Loader that uses the L4.Env.mem_alloc
00135 factory of the current Ned instance to create the objects for a new program.
00136 However you may also use a more restricted factory for applications and
00137 instantiate a loader for them. The L4.Loader objects can already be used
00138 to start a program with L4.Loader:start(app_spec, cmd, ...). Where app_spec
00139 is a table containing parameters for the new application. cmd is the
00140 command to run and the remaining arguments are the command-line options for
00141 the application.
00142
00143]==]
00144
00145 L4.default_loader:start({}, "rom/hello");
00146
00147 local _doc = [=[

```

```

00148
00149 This statement does the following:
00150 1. Create a new environment for the application
00151 2. Add the rom name-space into the new environment (thus shares Ned's
00152 'rom' directory with the new program).
00153 3. Creates all the building blocks for the new process and starts the
00154 'l4re' support kernel in the new process which in turn starts 'rom/hello'
00155 in the new process.
00156
00157 Using the app_spec parameter you can modify the behavior in two ways. There are
00158 two supported options 'caps' for providing more capabilities for the
00159 application. And 'log' for modifying the logger tag and color.
00160
00161]==]
00162
00163 local my_caps = {
00164 fb = L4.Env.vesa;
00165 };
00166
00167 L4.default_loader:start({caps = my_caps, log = {"APP", "blue"}}, "rom/hello");
00168
00169 local _doc = [=[
00170
00171 This snippet creates a caps template (my_caps) and uses it for the
00172 new process and also sets user-defined log tags. The L4.Loader:start method,
00173 however, automatically adds the 'rom' directory to the caps environment if
00174 not already specified in the template.
00175
00176 Environment variables may be given as a table in the third argument to
00177 start. Argument to the program are given space separated after the program
00178 name within a single string.
00179
00180]==]
00181
00182 L4.default_loader:start({}, "rom/program arg1 " .. arg2, { LD_DEBUG = 1 });
00183
00184 local _doc = [=[
00185
00186 L4.default_loader:startv is a variant of the start function that takes the
00187 arguments of the program as a single argument each. If the last argument to
00188 startv is a table it is interpreted as environment variables for the program.
00189 The above example would translate to:
00190
00191]==]
00192
00193 L4.default_loader:startv({}, "rom/program", "arg1", arg2, { LD_DEBUG = 1 });
00194
00195 local _doc = [=[
00196
00197 To create a new L4.Loader instance you may use a generic factory for all
00198 building blocks or set individual factories.
00199
00200]==]
00201
00202 l = L4.Loader.new({
00203 mem = L4.Env.user_factory:create(L4.Proto.Factory, 512*1024):m("rs")
00204 });
00205
00206 local _doc = [=[
00207
00208 Creates a loader instance that uses the newly created 512 KB factory for
00209 all building blocks. To set individual factories use the options:
00210 'mem' as memory allocator for the new processes and as
00211 default factory for all objects not explicitly set to a
00212 different factory
00213 'log_fab' for creating log objects.
00214 'ns_fab' for creating name-space objects.
00215 'rm_fab' for creating region-map objects.
00216
00217
00218
00219 L4.App_env
00220 -----
00221
00222 L4.App_env provides a more fine-grained control for a single process or for a
00223 limited number of processes. L4.App_env uses an L4.Loader object as basic
00224 facility. However you can override the memory allocator 'mem' for the new
00225 process as well as the kernel factory 'factory', the log capability etc.
00226
00227]==]
00228
00229 local e = L4.App_env.new({
00230 loader = l,
00231 mem = L4.Env.user_factory:create(L4.Proto.Factory, 128*1024):m("rs")
00232 });
00233
00234 e:start("rom/hello");

```

## 17.39 cmd\_control

```

00001 // -*- Mode: C++ -*-
00002 // vim:ft=cpp
00003 /*
00004 * Copyright (C) 2016, 2024 Kernkonzept GmbH.
00005 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 #pragma once
00010
00011 #include <l4/sys/cxx/ipc_epiface>
00012 #include <l4/sys/cxx/ipc_string>
00013
00014 namespace L4Re { namespace Ned {
00015
00016 class Cmd_control : public L4::Kobject_0t<Cmd_control>
00017 {
00018 L4_INLINE_RPC_NF(long, execute, (L4::Ipc::String<> cmd,
00019 L4::Ipc::Array<char> &result));
00020
00021 public:
00022 long execute(L4::Ipc::String<> cmd) noexcept
00023 {
00024 L4::Ipc::Array<char> res(0, NULL);
00025 return execute_t::call(c(), cmd, res);
00026 }
00027
00028 long execute(L4::Ipc::String<> cmd,
00029 L4::Ipc::String<char> *result) noexcept
00030 {
00031 L4::Ipc::Array<char> res(result->length, result->data);
00032 long r = execute_t::call(c(), cmd, res);
00033 if (r >= 0)
00034 result->length = res.length;
00035 return r;
00036 }
00037
00038 typedef L4::Typeid::Rpc<execute_t> Rpc;
00039 };
00040 } } // namespace

```

## 17.40 debug.h

```

00001 /*
00002 * (c) 2013-2014 Alexander Warg <warg@os.inf.tu-dresden.de>
00003 * economic rights: Technische Universität Dresden (Germany)
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/re/util/debug>
00010
00011 struct Err : L4Re::Util::Err
00012 {
00013 Err(Level l = Fatal) : L4Re::Util::Err(l, "VSwitch") {}
00014 };
00015
00016 class Dbg : public L4Re::Util::Dbg
00017 {
00018 enum
00019 {
00020 Verbosity_shift = 4,
00021 Verbosity_mask = (1UL « Verbosity_shift) - 1
00022 };
00023
00024 public:
00025 enum Verbosity : unsigned long
00026 {
00027 Quiet = 0,
00028 Warn = 1,
00029 Info = 2,
00030 Debug = 4,
00031 Trace = 8,
00032 Max_verbosity = 8
00033 };
00034
00035 enum Component
00036 {

```

```

00041 Core = 0,
00042 Virtio,
00043 Port,
00044 Request,
00045 Queue,
00046 Packet,
00047 Max_component
00048 };
00049
00050 #ifndef NDEBUG
00051
00052 static_assert(Max_component * Verbosity_shift <= sizeof(level) * 8,
00053 "Too many components for level mask");
00054 static_assert((Max_verbosity & Verbosity_mask) == Max_verbosity,
00055 "Verbosity_shift too small for verbosity levels");
00056
00062 static void set_verbosity(unsigned mask)
00063 {
00064 for (unsigned i = 0; i < Max_component; ++i)
00065 set_verbosity(i, mask);
00066 }
00067
00074 static void set_verbosity(unsigned c, unsigned mask)
00075 {
00076 level &= ~(Verbosity_mask << (Verbosity_shift * c));
00077 level |= (mask & Verbosity_mask) << (Verbosity_shift * c);
00078 }
00079
00089 static bool is_active(unsigned c, unsigned mask)
00090 { return level & (mask & Verbosity_mask) << (Verbosity_shift * c); }
00091
00098 using L4Re::Util::Dbg::is_active;
00099
00100 Dbg(Component c = Core, Verbosity v = Warn, char const *subsys = "")
00101 : L4Re::Util::Dbg(v << (Verbosity_shift * c), "SWI", subsys)
00102 {}
00103
00104 #else
00105
00106 static void set_verbosity(unsigned) {}
00107 static void set_verbosity(unsigned, unsigned) {}
00108
00109 static bool is_active(unsigned, unsigned) { return false; }
00110 using L4Re::Util::Dbg::is_active;
00111
00112 Dbg(Component c = Core, Verbosity v = Warn, char const *subsys = "")
00113 : L4Re::Util::Dbg(v << (Verbosity_shift * c), "", subsys)
00114 {}
00115
00116 #endif
00117 };

```

## 17.41 debug.h

```

00001 /*
00002 * Copyright (C) 2018, 2024 Kernkonzept GmbH.
00003 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/re/util/debug>
00010
00011 namespace Block_device {
00012
00013 class Err : public L4Re::Util::Err
00014 {
00015 public:
00016 explicit
00017 Err(Level l = Normal) : L4Re::Util::Err(l, "") {}
00018 };
00019
00020 class Dbg : public L4Re::Util::Dbg
00021 {
00022 public:
00023 enum Level
00024 {
00025 Blk_warn = 1,
00026 Blk_info = 2,
00027 Blk_trace = 4,
00028 Blk_steptrace = 8
00029 };

```

```

00030 public:
00031 Dbg(unsigned long l = Blk_info, char const *subsys = "")
00032 : L4Re::Util::Dbg(l, "libblock", subsys) {}
00033
00034 static Dbg warn(char const *subsys = "")
00035 { return Dbg(Blk_warn, subsys); }
00036
00037 static Dbg info(char const *subsys = "")
00038 { return Dbg(Blk_info, subsys); }
00039
00040 static Dbg trace(char const *subsys = "")
00041 { return Dbg(Blk_trace, subsys); }
00042
00043 static Dbg steptrace(char const *subsys = "")
00044 { return Dbg(Blk_steptrace, subsys); }
00045 };
00046
00047 } // name space
00048

```

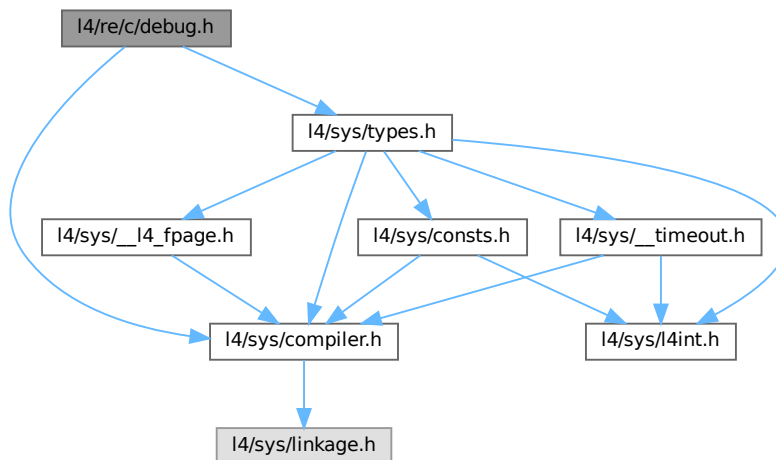
## 17.42 l4/re/c/debug.h File Reference

Debug C interface.

```

#include <l4/sys/compiler.h>
#include <l4/sys/types.h>
Include dependency graph for debug.h:

```



### Functions

- `L4_BEGIN_DECLS` long `l4re_debug_obj_debug` (`l4_cap_idx_t` srv, unsigned long function) `L4_NOTHROW`  
Call debug function of `L4Re` service.

### 17.42.1 Detailed Description

Debug C interface.

Definition in file `debug.h`.

## 17.43 debug.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #pragma once
00012
00017
00018 #include <14/sys/compiler.h>
00019 #include <14/sys/types.h>
00020
00021 L4_BEGIN_DECLS
00022
00030 L4_CV long
00031 l4re_debug_obj_debug(l4_cap_idx_t srv, unsigned long function) L4_NOTHROW;
00032
00033 L4_END_DECLS

```

## 17.44 filter.cc

```

00001 /*
00002 * Copyright (C) 2016-2017, 2024 Kernkonzept GmbH.
00003 * Author(s): Jean Wolter <jean.wolter@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #include "filter.h"
00008
00009 /* This is an example filter and therefore rather verbose. A real
00010 filter would not produce any output */
00011
00012 bool
00013 filter(const uint8_t *buf, size_t size)
00014 {
00015 // Packet large enough to apply filter condition?
00016 if (size <= 13)
00017 return false;
00018
00019 uint16_t ether_type = (uint16_t)*(buf + 12) << 8
00020 | (uint16_t)*(buf + 13);
00021 char const *protocol;
00022 switch (ether_type)
00023 {
00024 case 0x0800: protocol = "IPv4"; break;
00025 case 0x0806: protocol = "ARP"; break;
00026 case 0x8100: protocol = "Vlan"; break;
00027 case 0x86dd: protocol = "IPv6"; break;
00028 case 0x8863: protocol = "PPPoE Discovery"; break;
00029 case 0x8864: protocol = "PPPoE Session"; break;
00030 default: protocol = nullptr;
00031 }
00032 if (protocol)
00033 printf("%s\n", protocol);
00034 else
00035 printf("%04x\n", ether_type);
00036
00037 if (ether_type == 0x0806)
00038 {
00039 printf("Do not filter arp\n");
00040 return false;
00041 }
00042
00043 return true;
00044 }

```

## 17.45 filter.h

```

00001 /*
00002 * Copyright (C) 2016-2017, 2023-2024 Kernkonzept GmbH.
00003 * Author(s): Jean Wolter <jean.wolter@kernkonzept.com>

```



```

00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include "request.h"
00010 #include <l4/bid_config.h>
00011
00024 #ifndef CONFIG_VNS_PORT_FILTER
00025 bool filter(const uint8_t *buf, size_t size);
00026 #else
00027 inline bool filter(const uint8_t *, size_t)
00028 {
00029 // default implementation filtering out no packets, see filter.cc for
00030 // other examples
00031 return false;
00032 }
00033 #endif
00034
00043 inline bool filter_request(Net_request const &req)
00044 {
00045 size_t size;
00046 const uint8_t *buf = req.buffer(&size);
00047 return filter(buf, size);
00048 }

```

## 17.46 mac\_addr.h

```

00001 /*
00002 * Copyright (C) 2016-2017, 2020, 2022-2024 Kernkonzept GmbH.
00003 * Author(s): Jean Wolter <jean.wolter@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <cstring>
00010 #include <inttypes.h>
00011
00019 class Mac_addr
00020 {
00021 public:
00022 enum
00023 {
00024 Addr_length = 6,
00025 Addr_unknown = 0ULL
00026 };
00027
00028 explicit Mac_addr(char const *_src)
00029 {
00030 /* A mac address is 6 bytes long, it is transmitted in big endian
00031 order over the network. For our internal representation we
00032 focus on easy testability of broadcast/multicast and reorder
00033 the bytes that the most significant byte becomes the least
00034 significant one. */
00035 unsigned char const *src = reinterpret_cast<unsigned char const *>(_src);
00036 _mac = ((uint64_t)src[0]) | (((uint64_t)src[1]) << 8)
00037 | (((uint64_t)src[2]) << 16) | (((uint64_t)src[3]) << 24)
00038 | (((uint64_t)src[4]) << 32) | (((uint64_t)src[5]) << 40);
00039 }
00040
00041 static Mac_addr from_uncached(char volatile const *src)
00042 { return Mac_addr(src); }
00043
00044 explicit Mac_addr(uint64_t mac) : _mac{mac} {}
00045
00046 Mac_addr(Mac_addr const &other) : _mac{other._mac} {}
00047
00049 bool is_broadcast() const
00050 {
00051 /* There are broadcast and multicast addresses, both are supposed
00052 to be delivered to all station and the local network (layer 2).
00053
00054 Broadcast address is FF:FF:FF:FF:FF:FF, multicast addresses have
00055 the LSB of the first octet set. Since this holds for both
00056 broadcast and multicast we test for the multicast bit here.
00057
00058 In our internal representation we store the bytes in reverse
00059 order, so we test the least significant bit of the least
00060 significant byte.
00061 */
00062 return _mac & 1;
00063 }

```

```

00064
00066 bool is_unknown() const
00067 { return _mac == Addr_unknown; }
00068
00069 bool operator == (Mac_addr const &other) const
00070 { return _mac == other._mac; }
00071
00072 bool operator != (Mac_addr const &other) const
00073 { return _mac != other._mac; }
00074
00075 bool operator < (Mac_addr const &other) const
00076 { return _mac < other._mac; }
00077
00078 Mac_addr& operator = (Mac_addr const &other)
00079 { _mac = other._mac; return *this; }
00080
00081 Mac_addr& operator = (uint64_t mac)
00082 { _mac = mac; return *this; }
00083
00084 template<typename T>
00085 void print(T &stream) const
00086 {
00087 stream.cprintf("%02x:%02x:%02x:%02x:%02x",
00088 (int)(_mac & 0xff), (int)((_mac >> 8) & 0xff),
00089 (int)((_mac >> 16) & 0xff), (int)((_mac >> 24) & 0xff),
00090 (int)((_mac >> 32) & 0xff), (int)((_mac >> 40) & 0xff));
00091 }
00092
00093 void to_array(unsigned char mac[6]) const
00094 {
00095 mac[0] = _mac & 0xffU;
00096 mac[1] = (_mac >> 8) & 0xffU;
00097 mac[2] = (_mac >> 16) & 0xffU;
00098 mac[3] = (_mac >> 24) & 0xffU;
00099 mac[4] = (_mac >> 32) & 0xffU;
00100 mac[5] = (_mac >> 40) & 0xffU;
00101 }
00102
00103 private:
00104 explicit Mac_addr(char volatile const *_src)
00105 {
00106 /* A mac address is 6 bytes long, it is transmitted in big endian
00107 order over the network. For our internal representation we
00108 focus on easy testability of broadcast/multicast and reorder
00109 the bytes that the most significant byte becomes the least
00110 significant one. */
00111 volatile unsigned char const *src = reinterpret_cast<volatile unsigned char const *>(_src);
00112 _mac = ((uint64_t)src[0]) | (((uint64_t)src[1]) << 8)
00113 | (((uint64_t)src[2]) << 16) | (((uint64_t)src[3]) << 24)
00114 | (((uint64_t)src[4]) << 32) | (((uint64_t)src[5]) << 40);
00115 }
00116
00117 uint64_t _mac;
00118 };

```

## 17.47 mac\_table.h

```

00001 /*
00002 * Copyright (C) 2016-2017, 2020, 2022-2024 Kernkonzept GmbH.
00003 * Author(s): Jean Wolter <jean.wolter@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include "mac_addr.h"
00010 #include "port.h"
00011
00012 #include <array>
00013 #include <map>
00014 #include <tuple>
00015 #include <algorithm>
00020
00039 template<std::size_t Size = 1024U>
00040 class Mac_table
00041 {
00042 public:
00043 Mac_table()
00044 : _mac_table(),
00045 _entries(),
00046 _rr_index(0U)
00047 {}
00048

```

```

00058 Port_iface *lookup(Mac_addr dst, l4_uint16_t vlan_id) const
00059 {
00060 auto entry = _mac_table.find(std::tuple(dst, vlan_id));
00061 return (entry != _mac_table.end()) ? entry->second->port : nullptr;
00062 }
00063
00078 void learn(Mac_addr src, Port_iface *port, l4_uint16_t vlan_id)
00079 {
00080 Dbg info(Dbg::Port, Dbg::Info);
00081
00082 if (L4_UNLIKELY(info.is_active()))
00083 {
00084 // check whether we already know about src mac and vlan_id
00085 auto *p = lookup(src, vlan_id);
00086 if (!p || p != port)
00087 {
00088 info.printf("%s %-20s -> ", !p ? "learned " : "replaced",
00089 port->get_name());
00090 src.print(info);
00091 info.cprintf("\n");
00092 }
00093 }
00094
00095 auto status = _mac_table.emplace(std::tuple(src, vlan_id),
00096 &_entries[_rr_index]);
00097 if (L4_UNLIKELY(status.second))
00098 {
00099 if (_entries[_rr_index].port)
00100 {
00101 // remove old entry
00102 _mac_table.erase(std::tuple(_entries[_rr_index].addr,
00103 _entries[_rr_index].vlan_id));
00104 }
00105 // Set/Replace port and mac address
00106 _entries[_rr_index].port = port;
00107 _entries[_rr_index].addr = src;
00108 _entries[_rr_index].vlan_id = vlan_id;
00109 _rr_index = (_rr_index + 1U) % Size;
00110 }
00111 else
00112 {
00113 // Update port to allow for movement of client between ports
00114 status.first->second->port = port;
00115 }
00116 }
00117
00129 void flush(Port_iface *port)
00130 {
00131 typedef std::pair<std::tuple<const Mac_addr, l4_uint16_t>, Entry*> TableEntry;
00132
00133 auto iter = _mac_table.begin();
00134 while ((iter = std::find_if(iter, _mac_table.end(),
00135 [port](TableEntry const &p)
00136 { return p.second->port == port; })))
00137 {
00138 iter->second->port = nullptr;
00139 iter->second->addr = Mac_addr::Addr_unknown;
00140 iter->second->vlan_id = 0;
00141 iter = _mac_table.erase(iter);
00142 }
00143
00144 assert(std::find_if(_mac_table.begin(), _mac_table.end(),
00145 [port](TableEntry const &p)
00146 { return p.second->port == port; }) == _mac_table.end());
00147 }
00148
00150 private:
00151 struct Entry {
00152 Port_iface *port;
00153 Mac_addr addr;
00154 l4_uint16_t vlan_id;
00155
00156 Entry()
00157 : port(nullptr),
00158 addr(Mac_addr::Addr_unknown),
00159 vlan_id(0)
00160 {}
00161 };
00162
00163 std::map<std::tuple<Mac_addr, l4_uint16_t>, Entry*> _mac_table;
00164 std::array<Entry, Size> _entries;
00165 size_t _rr_index;
00166 };

```

## 17.48 main.cc

```

00001 /*
00002 * Copyright (C) 2016-2020, 2022-2024 Kernkonzept GmbH.
00003 * Author(s): Jean Wolter <jean.wolter@kernkonzept.com>
00004 * Manuel von Oltersdorff-Kalettka <manuel.kalettka@kernkonzept.com>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #include <l4/re/util/meta>
00009 #include <l4/re/util/object_registry>
00010 #include <l4/re/util/br_manager>
00011
00012 #include <l4/sys/factory>
00013 #include <l4/sys/task>
00014
00015 #include <l4/sys/cxx/ipc_epiface>
00016 #include <l4/sys/cxx/ipc_varg>
00017 #include <l4/cxx/dlist>
00018 #include <l4/cxx/string>
00019
00020 #include <stdlib.h>
00021 #include <string>
00022 #include <terminate_handler-l4>
00023 #include <vector>
00024
00025 #include "debug.h"
00026 #include "options.h"
00027 #include "switch.h"
00028 #include "vlan.h"
00029 #include <l4/virtio-net-switch/stats.h>
00030
00047
00048
00049 /*
00050 * Registry for our server, used to register
00051 * - factory capability
00052 * - irq object for capability deletion irqs
00053 * - virtio host kick irqs
00054 */
00055 static L4Re::Util::Registry_server<L4Re::Util::Br_manager_hooks> server;
00056
00057 using Ds_vector = std::vector<L4::Cap<L4Re::Dataspace>>;
00058 static std::shared_ptr<Ds_vector> trusted_dataspaces;
00059
00060 static bool
00061 parse_int_param(L4::Ipc::Varg const ¶m, char const *prefix, int *out)
00062 {
00063 l4_size_t headlen = strlen(prefix);
00064
00065 if (param.length() < headlen)
00066 return false;
00067
00068 char const *pstr = param.value<char const *>();
00069
00070 if (strncmp(pstr, prefix, headlen) != 0)
00071 return false;
00072
00073 std::string tail(pstr + headlen, param.length() - headlen);
00074
00075 if (!parse_int_optstring(tail.c_str(), out))
00076 {
00077 Err(Err::Normal).printf("Bad parameter '%s'. Invalid number specified.\n",
00078 prefix);
00079 throw L4::Runtime_error(-L4_EINVAL);
00080 }
00081
00082 return true;
00083 }
00084
00085 static void
00086 assign_random_mac(l4_uint8_t mac[6])
00087 {
00088 static bool initialized = false;
00089
00090 if (!initialized)
00091 {
00092 srandom(l4_kip_clock(l4re_kip()));
00093 initialized = true;
00094 }
00095
00096 for (int i = 0; i < 6; i++)
00097 mac[i] = static_cast<l4_uint8_t>(random());
00098
00099 mac[0] &= ~(1U << 0); // clear multicast bit
00100 mac[0] |= 1U << 1; // set "locally administered" bit

```

```

00101 }
00102
00114 class Switch_factory : public L4::Epiface_t<Switch_factory, L4::Factory>
00115 {
00119 class Port : public L4virtio_port
00120 {
00121 // Irq used to notify the guest
00122 L4::Cap<L4::Irq> _device_notify_irq;
00123
00124 L4::Cap<L4::Irq> device_notify_irq() const override
00125 { return _device_notify_irq; }
00126
00127 public:
00128 Port(unsigned vq_max, unsigned num_ds, char const *name,
00129 l4_uint8_t const *mac)
00130 : L4virtio_port(vq_max, num_ds, name, mac) {}
00131
00133 void register_end_points(L4Re::Util::Object_registry* registry,
00134 L4::Epiface *kick_irq)
00135 {
00136 // register virtio host kick irq
00137 _device_notify_irq = L4Re::chkcapi(registry->register_irq_obj(kick_irq));
00138
00139 // register virtio endpoint
00140 L4Re::chkcapi(registry->register_obj(this));
00141
00142 // decrement ref counter to get a notification when the last
00143 // external reference vanishes
00144 obj_cap()->dec_refcnt(1);
00145 }
00146
00147 virtual ~Port()
00148 { server.registry()->unregister_obj(this); }
00149 };
00150
00154 class Switch_port : public Port
00155 {
00166 class Kick_irq : public L4::Irqep_t<Kick_irq>
00167 {
00168 Virtio_switch *_switch;
00169 L4virtio_port *_port;
00170
00171 public:
00172 void handle_irq()
00173 { _switch->handle_l4virtio_port_tx(_port); }
00174
00175 Kick_irq(Virtio_switch *virtio_switch, L4virtio_port *port)
00176 : _switch{virtio_switch}, _port{port} {}
00177 };
00178
00179 Kick_irq _kick_irq;
00180 Kick_irq _reschedule_tx_irq;
00181
00182 public:
00183 Switch_port(L4Re::Util::Object_registry *registry,
00184 Virtio_switch *virtio_switch, unsigned vq_max, unsigned num_ds,
00185 char const *name, l4_uint8_t const *mac)
00186 : Port(vq_max, num_ds, name, mac),
00187 _kick_irq(virtio_switch, this),
00188 _reschedule_tx_irq(virtio_switch, this)
00189 {
00190 register_end_points(registry, &_kick_irq);
00191
00192 _pending_tx_reschedule =
00193 L4Re::chkcapi(registry->register_irq_obj(&_reschedule_tx_irq),
00194 "Register TX reschedule IRQ.");
00195 _pending_tx_reschedule->unmask();
00196 }
00197
00198 virtual ~Switch_port()
00199 {
00200 // We need to delete the IRQ object created in register_irq_obj() ourselves
00201 L4::Cap<L4::Task> (L4Re::This_task)
00202 ->unmap(_kick_irq.obj_cap().fpage(),
00203 L4_FP_ALL_SPACES | L4_FP_DELETE_OBJ);
00204 server.registry()->unregister_obj(&_kick_irq);
00205
00206 L4::Cap<L4::Task> (L4Re::This_task)
00207 ->unmap(_pending_tx_reschedule.fpage(),
00208 L4_FP_ALL_SPACES | L4_FP_DELETE_OBJ);
00209 server.registry()->unregister_obj(&_reschedule_tx_irq);
00210 }
00211 };
00212
00222 class Monitor_port : public Port
00223 {
00224 class Kick_irq : public L4::Irqep_t<Kick_irq>

```

```

00230 {
00231 L4virtio_port *_port;
00232 }
00233 public:
00241 void handle_irq()
00242 {
00243 do
00244 {
00245 _port->tx_q()->disable_notify();
00246 _port->rx_q()->disable_notify();
00247
00248 _port->drop_requests();
00249
00250 _port->tx_q()->enable_notify();
00251 _port->rx_q()->enable_notify();
00252
00253 L4virtio::wmb();
00254 L4virtio::rmb();
00255 }
00256 while (_port->tx_work_pending());
00257 }
00258
00259 Kick_irq(L4virtio_port *port) : _port{port} {}
00260 };
00261
00262 Kick_irq _kick_irq;
00263
00264 public:
00265 Monitor_port(L4Re::Util::Object_registry* registry,
00266 unsigned vq_max, unsigned num_ds, char const *name,
00267 l4_uint8_t const *mac)
00268 : Port(vq_max, num_ds, name, mac), _kick_irq(this)
00269 { register_end_points(registry, &_kick_irq); }
00270
00271 virtual ~Monitor_port()
00272 {
00273 // We need to delete the IRQ object created in register_irq_obj() ourselves
00274 L4::Cap<L4::Task>(L4Re::This_task)
00275 ->unmap(_kick_irq.obj_cap().fpage(),
00276 L4_FP_ALL_SPACES | L4_FP_DELETE_OBJ);
00277 server.registry()->unregister_obj(&_kick_irq);
00278 }
00279 };
00280
00281 class Stats_reader
00282 : public cxx::D_list_item,
00283 public L4::Epiface_t<Stats_reader, Virtio_net_switch::Statistics_if>
00284 {
00285 L4Re::Util::Unique_cap<L4Re::Dataspace> _ds;
00286 l4_addr_t _addr;
00287
00288 public:
00289 Stats_reader()
00290 {
00291 l4_size_t size = Switch_statistics::get_instance().size();
00292 _ds = L4Re::Util::make_unique_cap<L4Re::Dataspace>();
00293 L4Re::chksys(L4Re::Env::env()->mem_alloc()->alloc(size, _ds.get()),
00294 "Could not allocate shared mem ds.");
00295 L4Re::chksys(L4Re::Env::env()->rm()->attach(&_addr, _ds->size(),
00296 L4Re::Rm::F::Search_addr
00297 | L4Re::Rm::F::RW,
00298 L4::Ipc::make_cap_rw(_ds.get())));
00299
00300 memset(reinterpret_cast<void*>(_addr), 0, _ds->size());
00301 }
00302
00303 ~Stats_reader()
00304 {
00305 L4Re::Env::env()->rm()->detach(reinterpret_cast<l4_addr_t>(_addr), 0);
00306 server.registry()->unregister_obj(this);
00307 }
00308
00309 long op_get_buffer(Virtio_net_switch::Statistics_if::Rights,
00310 L4::Ipc::Cap<L4Re::Dataspace> &ds)
00311 {
00312 // We hand out the dataspace in a read only manner. Clients must not be
00313 // able to modify information as that would create an unwanted data
00314 // channel.
00315 ds = L4::Ipc::Cap<L4Re::Dataspace>(_ds.get(), L4_CAP_FPAGE_RO);
00316 return L4_EOK;
00317 }
00318
00319 long op_sync(Virtio_net_switch::Statistics_if::Rights)
00320 {
00321 memcpy(reinterpret_cast<void*>(_addr),
00322 reinterpret_cast<void*>(Switch_statistics::get_instance().stats()),
00323 Switch_statistics::get_instance().size());
00324 }

```

```

00327 return L4_EOK;
00328 }
00329
00330 bool is_valid()
00331 { return obj_cap() && obj_cap().validate().label(); }
00332 };
00333
00334 class Stats_reader_list
00335 {
00336 cxx::D_list<Stats_reader> _readers;
00337
00338 public:
00339 void check_readers()
00340 {
00341 auto it = _readers.begin();
00342 while (it != _readers.end())
00343 {
00344 auto *reader = *it;
00345 if (!reader->is_valid())
00346 {
00347 it = _readers.erase(it);
00348 delete reader;
00349 }
00350 else
00351 ++it;
00352 }
00353 }
00354
00355 void push_back(cxx::unique_ptr<Stats_reader> reader)
00356 {
00357 _readers.push_back(reader.release());
00358 }
00359 };
00360
00361 /*
00362 * Handle vanishing caps by telling the switch that a port might have gone
00363 */
00364 struct Del_cap_irq : public L4::Irqep_t<Del_cap_irq>
00365 {
00366 public:
00367 void handle_irq()
00368 {
00369 _switch->check_ports();
00370 _stats_readers->check_readers();
00371 }
00372
00373 Del_cap_irq(Virtio_switch *virtio_switch, Stats_reader_list *stats_readers)
00374 : _switch{virtio_switch},
00375 _stats_readers{stats_readers}
00376 {}
00377
00378 private:
00379 Virtio_switch *_switch;
00380 Stats_reader_list *_stats_readers;
00381 };
00382
00383 Virtio_switch *_virtio_switch;
00384
00385 unsigned _vq_max_num;
00386 Stats_reader_list _stats_readers;
00387 Del_cap_irq _del_cap_irq;
00388
00389 bool handle_opt_arg(L4::Ipc::Varg const &opt, bool &monitor,
00400 char *name, size_t size,
00401 l4_uint16_t &vlan_access,
00402 std::vector<l4_uint16_t> &vlan_trunk,
00403 bool *vlan_trunk_all,
00404 l4_uint8_t mac[6], bool &mac_set)
00405 {
00406 assert(opt.is_of<char const *>());
00407 unsigned len = opt.length();
00408 const char *opt_str = opt.data();
00409 Err err(Err::Normal);
00410
00411 if (len > 5)
00412 {
00413 if (!strncmp("type=", opt_str, 5))
00414 {
00415 if (!strncmp("type=monitor", opt_str, len))
00416 {
00417 monitor = true;
00418 return true;
00419 }
00420 else if (!strncmp("type=none", opt_str, len))
00421 return true;
00422 }
00423 }
00424 }

```

```

00430 err.printf("Unknown type '%.*s'\n", opt.length() - 5, opt.data() + 5);
00431 return false;
00432 }
00433 else if (!strcmp("name=", opt_str, 5))
00434 {
00435 snprintf(name, size, "%.*s", opt.length() - 5, opt.data() + 5);
00436 return true;
00437 }
00438 else if (!strcmp("vlan=", opt_str, 5))
00439 {
00440 cxx::String str(opt_str + 5, strnlen(opt_str + 5, len - 5));
00441 cxx::String::Index idx;
00442
00443 if ((idx = str.starts_with("access=")))
00444 {
00445 str = str.substr(idx);
00446 l4_uint16_t vid;
00447 int next = str.from_dec(&vid);
00448 if (next && next == str.len() && vlan_valid_id(vid))
00449 vlan_access = vid;
00450 else
00451 {
00452 err.printf("Invalid VLAN access port id '%.*s'\n",
00453 opt.length(), opt.data());
00454 return false;
00455 }
00456 }
00457 else if ((idx = str.starts_with("trunk=")))
00458 {
00459 int next;
00460 l4_uint16_t vid;
00461 str = str.substr(idx);
00462 if (str == cxx::String("all"))
00463 {
00464 *vlan_trunk_all = true;
00465 return true;
00466 }
00467 while ((next = str.from_dec(&vid)))
00468 {
00469 if (!vlan_valid_id(vid))
00470 break;
00471 vlan_trunk.push_back(vid);
00472 if (next < str.len() && str[next] != ',')
00473 break;
00474 str = str.substr(next+1);
00475 }
00476
00477 if (vlan_trunk.empty() || !str.empty())
00478 {
00479 err.printf("Invalid VLAN trunk port spec '%.*s'\n",
00480 opt.length(), opt.data());
00481 return false;
00482 }
00483 }
00484 else
00485 {
00486 err.printf("Invalid VLAN specification..\n");
00487 return false;
00488 }
00489 }
00490 return true;
00491 }
00492 else if (!strcmp("mac=", opt_str, 4))
00493 {
00494 size_t const OPT_LEN = 4 /* mac= */ + 6*2 /* digits */ + 5 /* : */;
00495 // expect NUL terminated string for simplicity
00496 if (len > OPT_LEN && opt_str[OPT_LEN] == '\0' &&
00497 sscanf(opt_str+4, "%hhx:%hhx:%hhx:%hhx:%hhx:%hhx", &mac[0],
00498 &mac[1], &mac[2], &mac[3], &mac[4], &mac[5]) == 6)
00499 {
00500 mac_set = true;
00501 return true;
00502 }
00503
00504 err.printf("Invalid mac address '%.*s'\n", len - 4, opt_str + 4);
00505 return false;
00506 }
00507 }
00508
00509 err.printf("Unknown option '%.*s'\n", opt.length(), opt.data());
00510 return false;
00511 }
00512
00513 public:
00514 Switch_factory(Virtio_switch *virtio_switch, unsigned vq_max_num)
00515 : _virtio_switch{virtio_switch}, _vq_max_num{vq_max_num},
00516 _del_cap_irq{virtio_switch, &_stats_readers}

```



```

00517 {
00518 auto c = L4Re::chkcapp(server.registry()->register_irq_obj(&_del_cap_irq));
00519 L4Re::chksys(L4Re::Env::env()->main_thread()->register_del_irq(c));
00520 };
00521
00522 long op_create(L4::Factory::Rights, L4::Ipc::Cap<void> &res,
00523 l4_umword_t type, L4::Ipc::Varg_list_ref va)
00524 {
00525 switch (type)
00526 {
00527 case 0:
00528 return create_port(res, va);
00529 case 1:
00530 return create_stats(res);
00531 default:
00532 Dbg(Dbg::Core, Dbg::Warn).printf("op_create: Invalid object type\n");
00533 return -L4_EINVAL;
00534 }
00535 }
00536
00537 long create_port(L4::Ipc::Cap<void> &res, L4::Ipc::Varg_list_ref va)
00538 {
00539 Dbg warn(Dbg::Port, Dbg::Warn, "Port");
00540 Dbg info(Dbg::Port, Dbg::Info, "Port");
00541
00542 info.printf("Incoming port request\n");
00543
00544 bool monitor = false;
00545 char name[20] = "";
00546 unsigned arg_n = 2;
00547 l4_uint16_t vlan_access = 0;
00548 std::vector<l4_uint16_t> vlan_trunk;
00549 bool vlan_trunk_all = false;
00550
00551 l4_uint8_t mac[6];
00552 bool mac_set = false;
00553 int num_ds = 2;
00554
00555 for (L4::Ipc::Varg opt: va)
00556 {
00557 if (!opt.is_of<char const *>())
00558 {
00559 warn.printf("Unexpected type for argument %d\n", arg_n);
00560 return -L4_EINVAL;
00561 }
00562
00563 if (parse_int_param(opt, "ds-max=", &num_ds))
00564 {
00565 if (num_ds <= 0 || num_ds > 80)
00566 {
00567 Err(Err::Normal).printf("warning: client requested invalid number"
00568 " of data spaces: 0 < %d <= 80\n", num_ds);
00569 return -L4_EINVAL;
00570 }
00571 }
00572 else if (!handle_opt_arg(opt, monitor, name, sizeof(name), vlan_access,
00573 vlan_trunk, &vlan_trunk_all, mac, mac_set))
00574 return -L4_EINVAL;
00575
00576 ++arg_n;
00577 }
00578
00579 int port_num = _virtio_switch->port_available(monitor);
00580 if (port_num < 0)
00581 {
00582 warn.printf("No port available\n");
00583 return -L4_ENOMEM;
00584 }
00585
00586 if (vlan_access && (!vlan_trunk.empty() || vlan_trunk_all))
00587 {
00588 warn.printf("VLAN port cannot be access and trunk simultaneously.\n");
00589 return -L4_EINVAL;
00590 }
00591
00592 if (!name[0])
00593 snprintf(name, sizeof(name), "%s[%d]", monitor ? "monitor" : "",
00594 port_num);
00595
00596 info.printf(" Creating port %s\n", name,
00597 monitor ? " as monitor port" : "");
00598
00599 // Assign a random MAC address if we assign one to our devices but the
00600 // user has not passed an explicit one for a port.
00601 if (!mac_set && Options::get_options()->assign_mac())
00602 assign_random_mac(mac);
00603 }

```

```

00610 l4_uint8_t *mac_ptr = (mac_set || Options::get_options()->assign_mac())
00611 ? mac : nullptr;
00612
00613 // create port
00614 Port *port;
00615 if (monitor)
00616 {
00617 port = new Monitor_port(server.registry(), _vq_max_num, num_ds, name,
00618 mac_ptr);
00619 port->set_monitor();
00620
00621 if (vlan_access)
00622 warn.printf("vlan=access=<id> ignored on monitor ports!\n");
00623 if (!vlan_trunk.empty())
00624 warn.printf("vlan=trunk=... ignored on monitor ports!\n");
00625 }
00626 else
00627 {
00628 port = new Switch_port(server.registry(), _virtio_switch, _vq_max_num,
00629 num_ds, name, mac_ptr);
00630
00631 if (vlan_access)
00632 port->set_vlan_access(vlan_access);
00633 else if (vlan_trunk_all)
00634 port->set_vlan_trunk_all();
00635 else if (!vlan_trunk.empty())
00636 port->set_vlan_trunk(vlan_trunk);
00637 }
00638
00639 port->add_trusted_dataspaces(trusted_dataspaces);
00640 if (!trusted_dataspaces->empty())
00641 port->enable_trusted_ds_validation();
00642
00643 // hand port over to the switch
00644 bool added = monitor ? _virtio_switch->add_monitor_port(port)
00645 : _virtio_switch->add_port(port);
00646 if (!added)
00647 {
00648 delete port;
00649 return -L4_ENOMEM;
00650 }
00651 res = L4::Ipc::make_cap(port->obj_cap(), L4_CAP_FPAGE_RWS);
00652
00653 info.printf(" Created port %s\n", name);
00654 return L4_EOK;
00655 }
00656
00657 long create_stats(L4::Ipc::Cap<void> &res)
00658 {
00659 // Create a stats reader and throw away our reference to get a notification
00660 // when the external reference vanishes.
00661 auto reader = cxx::make_unique<Stats_reader>();
00662 L4Re::chkcapi(server.registry()->register_obj(reader.get()));
00663 reader->obj_cap()->dec_refcnt(1);
00664 res = L4::Ipc::make_cap(reader->obj_cap(),
00665 L4_CAP_FPAGE_R | L4_CAP_FPAGE_D);
00666 _stats_readers.push_back(cxx::move(reader));
00667 return L4_EOK;
00668 }
00669 };
00670
00671 #if CONFIG_VNS_IXL
00672 class Ixl_hw_port : public Ixl_port
00673 {
00674 template<typename Derived>
00675 class Port_irq : public L4::Irqp_t<Derived>
00676 {
00677 public:
00678 Port_irq(Virtio_switch *virtio_switch, Ixl_port *port)
00679 : _switch{virtio_switch}, _port{port} {}
00680
00681 protected:
00682 Virtio_switch *_switch;
00683 Ixl_port *_port;
00684 };
00685
00686 class Receive_irq : public Port_irq<Receive_irq>
00687 {
00688 public:
00689 using Port_irq::Port_irq;
00690
00691 void handle_irq()
00692 {
00693 if (!_port->dev()->check_rcv_irq(0))
00694 return;
00695
00696 if (_switch->handle_ixl_port_tx(_port))

```

```

00706 _port->dev()->ack_recv_irq(0);
00707 }
00708 };
00709
00710 class Reschedule_tx_irq : public Port_irq<Reschedule_tx_irq>
00711 {
00712 public:
00713 using Port_irq::Port_irq;
00714
00715 void handle_irq()
00716 {
00717 if (_switch->handle_ixl_port_tx(_port))
00718 // Entire TX queue handled, re-enable the recv IRQ again.
00719 _port->dev()->ack_recv_irq(0);
00720 }
00721 };
00722
00723 Receive_irq _recv_irq;
00724 Reschedule_tx_irq _reschedule_tx_irq;
00725
00726 public:
00727 Ixl_hw_port(L4Re::Util::Object_registry *registry,
00728 Virtio_switch *virtio_switch, Ixl::Ixl_device *dev)
00729 : Ixl_port(dev),
00730 _recv_irq(virtio_switch, this),
00731 _reschedule_tx_irq(virtio_switch, this)
00732 {
00733 L4::Cap<L4::Irq> recv_irq_cap = L4Re::chkcap(dev->get_recv_irq(0), "Get receive IRQ");
00734 L4Re::chkcap(registry->register_obj(&_recv_irq, recv_irq_cap),
00735 "Register receive IRQ.");
00736 recv_irq_cap->unmask();
00737
00738 _pending_tx_reschedule =
00739 L4Re::chkcap(registry->register_irq_obj(&_reschedule_tx_irq),
00740 "Register TX reschedule IRQ.");
00741 _pending_tx_reschedule->unmask();
00742 }
00743
00744 ~Ixl_hw_port() override
00745 {
00746 server.registry()->unregister_obj(&_recv_irq);
00747 }
00748 };
00749
00750 static void
00751 discover_ixl_devices(L4::Cap<L4vbus::Vbus> vbus, Virtio_switch *virtio_switch)
00752 {
00753 struct Ixl::Dev_cfg cfg;
00754 // Configure the device in asynchronous notify mode.
00755 cfg.irq_timeout_ms = -1;
00756
00757 // TODO: Support detecting multiple devices on a Vbus.
00758 // Setup the driver (also resets and initializes the NIC).
00759 Ixl::Ixl_device *dev = Ixl::Ixl_device::ixl_init(vbus, 0, cfg);
00760 if (!dev)
00761 // No Ixl supported device found, Ixl already printed an error message.
00762 return;
00763
00764 Ixl_hw_port *hw_port = new Ixl_hw_port(server.registry(), virtio_switch, dev);
00765 if (!virtio_switch->add_port(hw_port))
00766 {
00767 Err().printf("error adding ixl port\n");
00768 delete hw_port;
00769 }
00770 }
00771 #endif
00772
00773 int main(int argc, char *argv[])
00774 {
00775 trusted_dataspaces = std::make_shared<Ds_vector>();
00776 auto *opts = Options::parse_options(argc, argv, trusted_dataspaces);
00777 if (!opts)
00778 {
00779 Err().printf("Error during command line parsing.\n");
00780 return 1;
00781 }
00782
00783 // Show welcome message if debug level is not set to quiet
00784 if (Dbg(Dbg::Core, Dbg::Warn).is_active())
00785 printf("Hello from l4virtio switch\n");
00786
00787 Virtio_switch *virtio_switch = new Virtio_switch(opts->get_max_ports());
00788
00789 #ifdef CONFIG_VNS_STATS
00790 Switch_statistics::get_instance().initialize(opts->get_max_ports());
00791 #endif
00792 }

```

```

00793 #if CONFIG_VNS_IXL
00794 auto vbus = L4Re::Env::env()->get_cap<L4vbus::Vbus>("vbus");
00795 if (vbus.is_valid())
00796 discover_ixl_devices(vbus, virtio_switch);
00797 #endif
00798
00799 Switch_factory *factory = new Switch_factory(virtio_switch,
00800 opts->get_virtq_max_num());
00801
00802 L4::Cap<void> cap = server.registry()->register_obj(factory, "svr");
00803 if (!cap.is_valid())
00804 {
00805 Err().printf("error registering switch\n");
00806 return 2;
00807 }
00808
00809 /*
00810 * server loop will handle 4 types of events
00811 * - Switch_factory
00812 * - factory protocol
00813 * - capability deletion
00814 * - delegated to Virtio_switch::check_ports()
00815 * - Switch_factory::Switch_port
00816 * - irqs triggered by clients
00817 * - delegated to Virtio_switch::handle_l4virtio_port_tx()
00818 * - Virtio_net_transfer
00819 * - timeouts for pending transfer requests added by
00820 * Port_iface::handle_request() via registered via
00821 * L4::Epiface::server_iface()->add_timeout()
00822 */
00823 server.loop();
00824 return 0;
00825 }
00826

```

## 17.49 Makefile

```

00001 PKGDIR = ..
00002 L4DIR ?= $(PKGDIR)/../..
00003
00004 PKGNAME = drivers
00005 PC_FILENAME = drivers-frst
00006 EXTRA_TARGET += hw_mmio_register_block hw_register_block
00007
00008 include $(L4DIR)/mk/include.mk

```

## 17.50 Makefile

```

00001 PKGDIR = ../..
00002 L4DIR ?= $(PKGDIR)/../..
00003
00004 PKGNAME = drivers
00005
00006 include $(L4DIR)/mk/include.mk

```

## 17.51 Makefile

```

00001 PKGDIR = ../..
00002 L4DIR ?= $(PKGDIR)/../..
00003
00004 EXTRA_TARGET += cmd_control
00005
00006 include $(L4DIR)/mk/include.mk

```

## 17.52 Makefile

```

00001 PKGDIR ?= ../..
00002 L4DIR ?= $(PKGDIR)/../..
00003

```

```

00004 TARGET = l4vio_switch
00005
00006 REQUIRES_LIBS = libstdc++ l4virtio
00007 REQUIRES_LIBS-$(CONFIG_VNS_IXL) += ixl
00008
00009 SRC_CC-$(CONFIG_VNS_PORT_FILTER) += filter.cc
00010
00011 SRC_CC = main.cc switch.cc options.cc
00012
00013 include $(L4DIR)/mk/prog.mk

```

## 17.53 options.cc

```

00001 /*
00002 * Copyright (C) 2016-2017, 2019, 2022-2024 Kernkonzept GmbH.
00003 * Author(s): Jean Wolter <jean.wolter@kernkonzept.com>
00004 * Manuel von Oltersdorff-Kalettka <manuel.kalettka@kernkonzept.de>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00009 #include <getopt.h>
00010 #include <stdlib.h>
00011 #include <cstring>
00012 #include <type_traits>
00013
00014 #include <l4/cxx/exceptions>
00015 #include <l4/re/error_helper>
00016 #include <l4/re/env>
00017
00018 #include "debug.h"
00019 #include "options.h"
00020
00021 bool
00022 parse_int_optstring(char const *optstring, int *out)
00023 {
00024 char *endp;
00025
00026 errno = 0;
00027 long num = strtol(optstring, &endp, 10);
00028
00029 // check that long can be converted to int
00030 if (errno || *endp != '\0' || num < INT_MIN || num > INT_MAX)
00031 return false;
00032
00033 *out = num;
00034
00035 return true;
00036 }
00037
00038 static int
00039 verbosity_mask_from_string(char const *str, unsigned *mask)
00040 {
00041 if (strcmp("quiet", str) == 0)
00042 {
00043 *mask = Dbg::Quiet;
00044 return 0;
00045 }
00046 if (strcmp("warn", str) == 0)
00047 {
00048 *mask = Dbg::Warn;
00049 return 0;
00050 }
00051 if (strcmp("info", str) == 0)
00052 {
00053 *mask = Dbg::Warn | Dbg::Info;
00054 return 0;
00055 }
00056 if (strcmp("debug", str) == 0)
00057 {
00058 *mask = Dbg::Warn | Dbg::Info | Dbg::Debug;
00059 return 0;
00060 }
00061 if (strcmp("trace", str) == 0)
00062 {
00063 *mask = Dbg::Warn | Dbg::Info | Dbg::Debug | Dbg::Trace;
00064 return 0;
00065 }
00066
00067 return -L4_ENOENT;
00068 }
00069
00099 static void

```

```

00100 set_verbosity(char const *str)
00101 {
00102 unsigned mask;
00103 if (verbosity_mask_from_string(str, &mask) == 0)
00104 {
00105 Dbg::set_verbosity(mask);
00106 return;
00107 }
00108
00109 static char const *const components[] =
00110 { "core", "virtio", "port", "request", "queue", "packet" };
00111
00112 static_assert(std::extent<decltype(components)>::value == Dbg::Max_component,
00113 "Component names must match 'enum Component'.");
00114
00115 for (unsigned i = 0; i < Dbg::Max_component; ++i)
00116 {
00117 auto len = strlen(components[i]);
00118 if (strncmp(components[i], str, len) == 0 && str[len] == '='
00119 && verbosity_mask_from_string(str + len + 1, &mask) == 0)
00120 {
00121 Dbg::set_verbosity(i, mask);
00122 return;
00123 }
00124 }
00125 }
00126
00127 int
00128 Options::parse_cmd_line(int argc, char **argv,
00129 std::shared_ptr<Ds_vector> trusted_dataspaces)
00130 {
00131 int opt, index;
00132
00133 struct option options[] =
00134 {
00135 {"size", 1, 0, 's' }, // size of in/out queue == #buffers in queue
00136 {"ports", 1, 0, 'p' }, // number of ports
00137 {"mac", 0, 0, 'm' }, // switch sets MAC address for each client
00138 {"debug", 1, 0, 'D' }, // configure debug levels
00139 {"verbose", 0, 0, 'v' },
00140 {"quiet", 0, 0, 'q' },
00141 {"register-ds", 1, 0, 'd' }, // register a trusted dataspace
00142 {0, 0, 0, 0}
00143 };
00144
00145 unsigned long verbosity = Dbg::Warn;
00146
00147 Dbg info(Dbg::Core, Dbg::Info);
00148
00149 Dbg::set_verbosity(Dbg::Core, Dbg::Info);
00150 info.printf("Arguments:\n");
00151 for (int i = 0; i < argc; ++i)
00152 info.printf("\t%s\n", argv[i]);
00153
00154 Dbg::set_verbosity(verbosity);
00155 while ((opt = getopt_long(argc, argv, "s:p:mMqyD:d:", options, &index)) != -1)
00156 {
00157 switch (opt)
00158 {
00159 case 's':
00160
00161 // QueueNumMax must be power of 2 between 1 and 0x8000
00162 if (!parse_int_optstring(optarg, &_virtq_max_num)
00163 || _virtq_max_num < 1 || _virtq_max_num > 32768
00164 || (_virtq_max_num & (_virtq_max_num - 1)))
00165 {
00166 Err().printf("Max number of virtqueue buffers must be power of 2"
00167 " between 1 and 32768. Invalid value %i or argument "
00168 "%s\n",
00169 _virtq_max_num, optarg);
00169 return -1;
00170 }
00171 info.printf("Max number of buffers in virtqueue: %i\n",
00172 _virtq_max_num);
00173 break;
00174 case 'p':
00175 if (parse_int_optstring(optarg, &_max_ports))
00176 info.printf("Max number of ports: %u\n", _max_ports);
00177 else
00178 {
00179 Err().printf("Invalid number of ports argument: %s\n", optarg);
00180 return -1;
00181 }
00182 break;
00183 case 'q':
00184 verbosity = Dbg::Quiet;
00185 Dbg::set_verbosity(verbosity);
00186

```

```

00187 break;
00188 case 'v':
00189 verbosity = (verbosity < 1) | 1;
00190 Dbg::set_verbosity(verbosity);
00191 break;
00192 case 'D':
00193 set_verbosity(optarg);
00194 break;
00195 case 'm':
00196 info.printf("Option -m ignored to compatibility.\n");
00197 break;
00198 case 'M':
00199 _assign_mac = false;
00200 break;
00201 case 'd':
00202 {
00203 L4Re::Cap<L4Re::Dataspace> ds =
00204 L4Re::chkcap(L4Re::Env::env()->get_cap<L4Re::Dataspace>(optarg),
00205 "Find a dataspace capability.\n");
00206 trusted_dataspaces->push_back(ds);
00207 break;
00208 }
00209 default:
00210 Err().printf("Unknown command line option '%c' (%d)\n", opt, opt);
00211 return -1;
00212 }
00213 }
00214 return 0;
00215 }
00216
00217 static Options options;
00218
00219 Options const *
00220 Options::get_options()
00221 { return &options; }
00222
00223 Options const *
00224 Options::parse_options(int argc, char **argv,
00225 std::shared_ptr<Ds_vector> trusted_dataspaces)
00226 {
00227 if (options.parse_cmd_line(argc, argv, trusted_dataspaces) < 0)
00228 return nullptr;
00229
00230 return &options;
00231 }

```

## 17.54 options.h

```

00001 /*
00002 * Copyright (C) 2016-2017, 2022, 2024 Kernkonzept GmbH.
00003 * Author(s): Jean Wolter <jean.wolter@kernkonzept.com>
00004 * Manuel von Oltersdorff-Kalettka <manuel.kalettka@kernkonzept.com>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include <memory>
00011 #include <vector>
00012 #include <cerrno>
00013 #include <climits>
00014
00015 #include <l4/re/dataspace>
00016
00017 bool
00018 parse_int_optstring(char const *optstring, int *out);
00019
00020 class Options
00021 {
00022 using Ds_vector = std::vector<L4Re::Cap<L4Re::Dataspace>;
00023 public:
00024 int get_max_ports() const
00025 { return _max_ports; }
00026
00027 int get_virtq_max_num() const
00028 { return _virtq_max_num; }
00029
00030 int get_portq_max_num() const
00031 { return _portq_max_num; }
00032
00033 int get_request_timeout() const
00034 { return _request_timeout; }
00035 }

```

```

00036 int assign_mac() const
00037 { return _assign_mac; }
00038
00039 static Options const *
00040 parse_options(int argc, char **argv,
00041 std::shared_ptr<Ds_vector> trusted_dataspaces);
00042 static Options const *get_options();
00043
00044 private:
00045 int _max_ports = 5;
00046 int _virtq_max_num = 0x100; // default value for data queues
00047 int _portq_max_num = 50; // default value for port queues
00048 int _request_timeout = 1 * 1000 * 1000; // default packet timeout 1 second
00049 bool _assign_mac = true;
00050
00051 int parse_cmd_line(int argc, char **argv,
00052 std::shared_ptr<Ds_vector> trusted_dataspaces);
00053 };

```

## 17.55 port.h

```

00001 /*
00002 * Copyright (C) 2016-2018, 2020, 2022-2024 Kernkonzept GmbH.
00003 * Author(s): Jean Wolter <jean.wolter@kernkonzept.com>
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include "request.h"
00011 #include "mac_addr.h"
00012 #include "vlan.h"
00013 #include "stats.h"
00014
00015 #include <cassert>
00016 #include <set>
00017 #include <vector>
00018
00023
00024 class Port_iface
00025 {
00026 protected:
00027 Virtio_net_switch::Port_statistics *_stats;
00028
00029 public:
00030 Port_iface(char const *name)
00031 {
00032 strncpy(_name, name, sizeof(_name));
00033 _name[sizeof(_name) - 1] = '\0';
00034 #ifdef CONFIG_VNS_STATS
00035 _stats = Switch_statistics::get_instance().allocate_port_statistics(name);
00036 if (!_stats)
00037 throw L4::Runtime_error(-L4_ENOMEM,
00038 "Could not allocate port statistics.\n");
00039 #endif
00040 }
00041
00042 virtual ~Port_iface()
00043 {
00044 #ifdef CONFIG_VNS_STATS
00045 _stats->in_use = false;
00046 #endif
00047 }
00048
00049 // delete copy and assignment
00050 Port_iface(Port_iface const &) = delete;
00051 Port_iface &operator = (Port_iface const &) = delete;
00052
00053 char const *get_name() const
00054 { return _name; }
00055
00056 l4_uint16_t get_vlan() const
00057 { return _vlan_id; }
00058
00059 inline bool is_trunk() const
00060 { return _vlan_id == VLAN_ID_TRUNK; }
00061
00062 inline bool is_native() const
00063 { return _vlan_id == VLAN_ID_NATIVE; }
00064
00065 inline bool is_access() const
00066 { return !is_trunk() && !is_native(); }

```



```

00067
00075 void set_vlan_access(l4_uint16_t id)
00076 {
00077 assert(vlan_valid_id(id));
00078 _vlan_id = id;
00079 _vlan_bloom_filter = 0;
00080 _vlan_ids.clear();
00081 }
00082
00092 void set_vlan_trunk(const std::vector<l4_uint16_t> &ids)
00093 {
00094 // bloom filter to quickly reject packets that do not belong to this port
00095 l4_uint32_t filter = 0;
00096
00097 _vlan_ids.clear();
00098 for (const auto id : ids)
00099 {
00100 assert(vlan_valid_id(id));
00101 filter |= vlan_bloom_hash(id);
00102 _vlan_ids.insert(id);
00103 }
00104
00105 _vlan_id = VLAN_ID_TRUNK;
00106 _vlan_bloom_filter = filter;
00107 }
00108
00112 void set_vlan_trunk_all()
00113 {
00114 _vlan_all = true;
00115 _vlan_id = VLAN_ID_TRUNK;
00116 _vlan_bloom_filter = -1;
00117 }
00118
00125 void set_monitor()
00126 {
00127 _vlan_id = VLAN_ID_TRUNK;
00128 _vlan_bloom_filter = 0;
00129 }
00130
00139 bool match_vlan(uint16_t id)
00140 {
00141 // Regular case native/access port
00142 if (id == _vlan_id)
00143 return true;
00144
00145 // This port participates in all VLANs
00146 if (_vlan_all)
00147 return true;
00148
00149 // Quick check: does port probably accept this VLAN?
00150 if ((_vlan_bloom_filter & vlan_bloom_hash(id)) == 0)
00151 return false;
00152
00153 return _vlan_ids.find(id) != _vlan_ids.end();
00154 }
00155
00162 inline Mac_addr mac() const
00163 { return _mac; }
00164
00165 Virtio_vlan_mangle create_vlan_mangle(Port_iface *src_port) const
00166 {
00167 Virtio_vlan_mangle mangle;
00168
00169 if (is_trunk())
00170 {
00171 /*
00172 * Add a VLAN tag only if the packet does not already have one (by
00173 * coming from another trunk port) or if the packet does not belong to
00174 * any VLAN (by coming from a native port). The latter case is only
00175 * relevant if this is a monitor port. Otherwise traffic from native
00176 * ports is never forwarded to trunk ports.
00177 */
00178 if (!src_port->is_trunk() && !src_port->is_native())
00179 mangle = Virtio_vlan_mangle::add(src_port->get_vlan());
00180 }
00181 else
00182 {
00183 /*
00184 * Remove VLAN tag only if the packet actually has one (by coming from a
00185 * trunk port).
00186 */
00187 if (src_port->is_trunk())
00188 mangle = Virtio_vlan_mangle::remove();
00189 }
00190 return mangle;
00191 }
00192 virtual void rx_notify_disable_and_remember() = 0;

```

```

00193 virtual void rx_notify_emit_and_enable() = 0;
00194
00195 virtual bool is_gone() const = 0;
00196
00197 // std::optional<Net_request> get_tx_request() = 0;
00198
00199 enum class Result
00200 {
00201 Delivered, Exception, Dropped,
00202 };
00203
00204 virtual Result handle_request(Port_iface *src_port,
00205 Net_transfer &src,
00206 l4_uint64_t *bytes_transferred) = 0;
00207
00208 void reschedule_pending_tx()
00209 { _pending_tx_reschedule->trigger(); }
00210
00211 protected:
00212 /*
00213 * VLAN related management information.
00214 *
00215 * A port may either be
00216 * - a native port (_vlan_id == VLAN_ID_NATIVE), or
00217 * - an access port (_vlan_id set accordingly), or
00218 * - a trunk port (_vlan_id == VLAN_ID_TRUNK, _vlan_bloom_filter and
00219 * _vlan_ids populated accordingly, or _vlan_all == true).
00220 */
00221 l4_uint16_t _vlan_id = VLAN_ID_NATIVE; // VID for native/access port
00222 l4_uint32_t _vlan_bloom_filter = 0; // Bloom filter for trunk ports
00223 std::set<l4_uint16_t> _vlan_ids; // Authoritative list of trunk VLANs
00224 bool _vlan_all; // This port participates in all VLANs (ignoring _vlan_ids)
00225
00226 inline l4_uint32_t vlan_bloom_hash(l4_uint16_t vid)
00227 { return 1UL << (vid & 31U); }
00228
00229 L4::Cap<L4::Irq> _pending_tx_reschedule;
00230
00231 Mac_addr _mac = Mac_addr(Mac_addr::Addr_unknown);
00232 char _name[20];
00233
00234 public:
00235 #ifdef CONFIG_VNS_STATS
00236 inline void stat_inc_tx_num()
00237 { _stats->tx_num++; }
00238 inline void stat_inc_tx_dropped()
00239 { _stats->tx_dropped++; }
00240 inline void stat_inc_tx_bytes(l4_uint64_t bytes)
00241 { _stats->tx_bytes += bytes; }
00242 inline void stat_inc_rx_num()
00243 { _stats->rx_num++; }
00244 inline void stat_inc_rx_dropped()
00245 { _stats->rx_dropped++; }
00246 inline void stat_inc_rx_bytes(l4_uint64_t bytes)
00247 { _stats->rx_bytes += bytes; }
00248 #else
00249 inline void stat_inc_tx_num()
00250 {}
00251 inline void stat_inc_tx_dropped()
00252 {}
00253 inline void stat_inc_tx_bytes(l4_uint64_t /*bytes*/)
00254 {}
00255 inline void stat_inc_rx_num()
00256 {}
00257 inline void stat_inc_rx_dropped()
00258 {}
00259 inline void stat_inc_rx_bytes(l4_uint64_t /*bytes*/)
00260 {}
00261 #endif
00262 };
00263
00264
00265
00266
00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282

```

## 17.56 port\_ixl.h

```

00001 /*
00002 * Copyright (C) 2024 Kernkonzept GmbH.
00003 * Author(s): Georg Kotheimer <georg.kotheimer@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include "port.h"

```

```

00010 #include "request_ixl.h"
00011
00012 #include <l4/ixl/device.h>
00013 #include <l4/ixl/memory.h>
00014
00015 #include <optional>
00016
00021
00022 class Ixl_port : public Port_iface
00023 {
00024 public:
00025 static constexpr unsigned Tx_batch_size = 32;
00026 static constexpr unsigned Num_bufs = 1024;
00027 static constexpr unsigned Buf_size = 2048;
00028 static constexpr l4_uint64_t Max_mem_size = 1ULL << 28;
00029
00030 Ixl_port(Ixl::Ixl_device *dev)
00031 : Port_iface(dev->get_driver_name().c_str()),
00032 _dev(dev),
00033 _mempool(*_dev, Num_bufs, Buf_size, Max_mem_size)
00034 {
00035 Ixl::mac_address mac_addr = _dev->get_mac_addr();
00036 _mac = Mac_addr(reinterpret_cast<char const *>(mac_addr.addr));
00037 #if CONFIG_VNS_STATS
00038 _mac.to_array(_stats->mac);
00039 #endif
00040 }
00041
00042 // OPTIMIZE: Could use this information for rx batching, i.e. collect while
00043 // rx_notify is disabled, then flush the collected buffers when
00044 // rx_notify is enabled again.
00045 void rx_notify_disable_and_remember() override {}
00046 void rx_notify_emit_and_enable() override {}
00047 bool is_gone() const override { return false; }
00048
00050 bool tx_work_pending()
00051 {
00052 fetch_tx_requests();
00053 return _tx_batch_idx < _tx_batch_len;
00054 }
00055
00057 std::optional<Ixl_net_request> get_tx_request()
00058 {
00059 fetch_tx_requests();
00060 if (_tx_batch_idx < _tx_batch_len)
00061 return std::make_optional<Ixl_net_request>(_tx_batch[_tx_batch_idx++]);
00062 else
00063 return std::nullopt;
00064 }
00065
00066 Result handle_request(Port_iface *src_port, Net_transfer &src,
00067 l4_uint64_t *bytes_transferred) override
00068 {
00069 Virtio_vlan_mangle mangle = create_vlan_mangle(src_port);
00070
00071 Dbg trace(Dbg::Request, Dbg::Trace, "REQ-IXL");
00072 trace.printf("%s: Transfer request %p.\n", _name, src.req_id());
00073
00074 struct Ixl::pkt_buf *buf = _mempool.pkt_buf_alloc();
00075 if (!buf)
00076 {
00077 trace.printf("\tTransfer failed, out-of-memory, dropping.\n");
00078 return Result::Dropped;
00079 }
00080
00081 // NOTE: Currently, the switch does not offer checksum or segmentation
00082 // offloading to its l4virtio clients, so it is fine to simply ignore
00083 // the Virtio_net::Hdr of the request here.
00084
00085 // Copy the request to the pkt_buf.
00086 Buffer dst_buf(reinterpret_cast<char *>(buf->data),
00087 Buf_size - offsetof(Ixl::pkt_buf, data));
00088 unsigned max_size = Buf_size - offsetof(Ixl::pkt_buf, data);
00089 for (;;)
00090 {
00091 try
00092 {
00093 if (src.done())
00094 // Request completely copied to destination.
00095 break;
00096 }
00097 catch (L4virtio::Svr::Bad_descriptor &e)
00098 {
00099 trace.printf("\tTransfer failed, bad descriptor exception, dropping.\n");
00100
00101 // Handle partial transfers to destination port.
00102 Ixl::pkt_buf_free(buf);

```

```

00103 throw;
00104 }
00105
00106 if (dst_buf.done())
00107 {
00108 trace.printf(
00109 "\tTransfer failed, exceeds max packet-size, dropping.\n");
00110 Ixl::pkt_buf_free(buf);
00111 return Result::Dropped;
00112 }
00113
00114 auto &src_buf = src.cur_buf();
00115 trace.printf("\tCopying %p#%p:%u (%x) -> %p#%p:%u (%x)\n",
00116 src_port, src_buf.pos, src_buf.left, src_buf.left,
00117 static_cast<Port_iface *>(this),
00118 dst_buf.pos, dst_buf.left, dst_buf.left);
00119
00120 mangle.copy_pkt(dst_buf, src_buf);
00121 }
00122 buf->size = max_size - dst_buf.left;
00123 *bytes_transferred = buf->size;
00124
00125 // Enqueue the pkt_buf at the device.
00126 if (_dev->tx_batch(0, &buf, 1) == 1)
00127 {
00128 trace.printf("\tTransfer queued at device.\n");
00129 return Result::Delivered;
00130 }
00131 else
00132 {
00133 trace.printf("\tTransfer failed, dropping.\n");
00134 Ixl::pkt_buf_free(buf);
00135 return Result::Dropped;
00136 }
00137 }
00138
00139 Ixl::Ixl_device *dev() { return _dev; }
00140
00141 private:
00142 void fetch_tx_requests()
00143 {
00144 if (_tx_batch_idx < _tx_batch_len)
00145 // Previous batch not yet fully processed.
00146 return;
00147
00148 // Batch receive, then cache in member array, to avoid frequent interactions
00149 // with the hardware.
00150 _tx_batch_len = _dev->rx_batch(0, _tx_batch, Tx_batch_size);
00151 _tx_batch_idx = 0;
00152 }
00153
00154 Ixl::Ixl_device *_dev;
00155 Ixl::Mempool _mempool;
00156 Ixl::pkt_buf *_tx_batch[Tx_batch_size];
00157 unsigned _tx_batch_idx = 0;
00158 unsigned _tx_batch_len = 0;
00159 };
00160

```

## 17.57 port\_l4virtio.h

```

00001 /*
00002 * Copyright (C) 2016-2017, 2020, 2022-2024 Kernkonzept GmbH.
00003 * Author(s): Jean Wolter <jean.wolter@kernkonzept.com>
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * Georg Kotheimer <georg.kotheimer@kernkonzept.com>
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 #pragma once
00010
00011 #include "port.h"
00012 #include "request_l4virtio.h"
00013 #include "virtio_net.h"
00014
00015 #include <l4/cxx/pair>
00016
00017 #include <vector>
00018
00023
00036 class L4virtio_port : public Port_iface, public Virtio_net
00037 {
00038 public:

```

```

00042 explicit L4virtio_port(unsigned vq_max, unsigned num_ds, char const *name,
00043 l4_uint8_t const *mac)
00044 : Port_iface(name), Virtio_net(vq_max)
00045 {
00046 init_mem_info(num_ds);
00047
00048 Features hf = _dev_config.host_features(0);
00049 if (mac)
00050 {
00051 _mac = Mac_addr((char const *)mac);
00052 memcpy((void *)_dev_config.priv_config()->mac, mac,
00053 sizeof(_dev_config.priv_config()->mac));
00054
00055 hf.mac() = true;
00056 Dbg d(Dbg::Port, Dbg::Info);
00057 d.cprintf("%s: Adding Mac '", _name);
00058 _mac.print(d);
00059 d.cprintf("' to host features to %x\n", hf.raw);
00060 }
00061 _dev_config.host_features(0) = hf.raw;
00062 _dev_config.reset_hdr();
00063 Dbg(Dbg::Port, Dbg::Info)
00064 .printf("%s: Set host features to %x\n", _name,
00065 _dev_config.host_features(0));
00066 #if CONFIG_VNS_STATS
00067 _mac.to_array(_stats->mac);
00068 #endif
00069 }
00070
00071 void rx_notify_disable_and_remember() override
00072 {
00073 kick_disable_and_remember();
00074 }
00075
00076 void rx_notify_emit_and_enable() override
00077 {
00078 kick_emit_and_enable();
00079 }
00080
00081 bool is_gone() const override
00082 {
00083 return obj_cap() && !obj_cap().validate().label();
00084 }
00085
00086 bool tx_work_pending() const
00087 {
00088 return L4_LIKELY(tx_q()->ready()) && tx_q()->desc_avail();
00089 }
00090
00091
00092 std::optional<Virtio_net_request> get_tx_request()
00093 {
00094 return Virtio_net_request::get_request(this, tx_q());
00095 }
00096
00097
00103 void drop_requests()
00104 { Virtio_net_request::drop_requests(this, tx_q()); }
00105
00106 Result handle_request(Port_iface *src_port, Net_transfer &src,
00107 l4_uint64_t *bytes_transferred) override
00108 {
00109 Virtio_vlan_mangle mangle = create_vlan_mangle(src_port);
00110
00111 Dbg trace(Dbg::Request, Dbg::Trace, "REQ-VIO");
00112 trace.printf("%s: Transfer request %p.\n", _name, src.req_id());
00113
00114 Buffer dst;
00115 int total = 0;
00116 l4_uint16_t num_merged = 0;
00117 l4_uint64_t total_merged = 0;
00118 typedef cxx::Pair<L4virtio::Svr::Virtqueue::Head_desc, l4_uint32_t> Consumed_entry;
00119 std::vector<Consumed_entry> consumed;
00120
00121 Virtio_net *dst_dev = this;
00122 Virtqueue *dst_queue = rx_q();
00123 L4virtio::Svr::Virtqueue::Head_desc dst_head;
00124 L4virtio::Svr::Request_processor dst_req_proc;
00125 Virtio_net::Hdr *dst_header = nullptr;
00126
00127 for (;;)
00128 {
00129 try
00130 {
00131 if (src.done())
00132 // Request completely copied to destination.
00133 break;
00134 }
00135 catch (L4virtio::Svr::Bad_descriptor &e)

```

```

00136 {
00137 trace.printf("\tTransfer failed, bad descriptor exception, dropping.\n");
00138
00139 // Handle partial transfers to destination port.
00140 if (!consumed.empty())
00141 // Partial transfer, rewind to before first descriptor of transfer.
00142 dst_queue->rewind_avail(consumed.at(0).first);
00143 else if (dst_head)
00144 // Partial transfer, still at first _dst_head.
00145 dst_queue->rewind_avail(dst_head);
00146 throw;
00147 }
00148
00149 /* The source data structures are already initialized, the header
00150 is consumed and src stands at the very first real buffer.
00151 Initialize the target data structures if necessary and fill the
00152 header. */
00153 if (!dst_head)
00154 {
00155 if (!dst_queue->ready())
00156 return Result::Dropped;
00157
00158 auto r = dst_queue->next_avail();
00159
00160 if (L4_UNLIKELY(!r))
00161 {
00162 trace.printf("\tTransfer failed, destination queue depleted, dropping.\n");
00163 // Abort incomplete transfer.
00164 if (!consumed.empty())
00165 dst_queue->rewind_avail(consumed.front().first);
00166 return Result::Dropped;
00167 }
00168
00169 try
00170 {
00171 dst_head = dst_req_proc.start(dst_dev->mem_info(), r, &dst);
00172 }
00173 catch (L4virtio::Svr::Bad_descriptor &e)
00174 {
00175 Dbg(Dbg::Request, Dbg::Warn, "REQ")
00176 .printf("%s: bad descriptor exception: %s - %i"
00177 " -- signal device error in destination device %p.\n",
00178 __PRETTY_FUNCTION__, e.message(), e.error, dst_dev);
00179
00180 dst_dev->device_error();
00181 return Result::Exception; // Must not touch the dst queues anymore.
00182 }
00183
00184 if (!dst_header)
00185 {
00186 if (dst.left < sizeof(Virtio_net::Hdr))
00187 throw L4::Runtime_error(-L4_EINVAL,
00188 "Target buffer too small for header");
00189 dst_header = reinterpret_cast<Virtio_net::Hdr *>(dst.pos);
00190 trace.printf("\tCopying header to %p (size: %u)\n",
00191 dst.pos, dst.left);
00192
00193 /*
00194 * Header and csum offloading/general segmentation offloading
00195 *
00196 * We just copy the original header from source to
00197 * destination and have to consider three different
00198 * cases:
00199 * - no flags are set
00200 * - we got a packet that is completely checksummed
00201 * and correctly fragmented, there is nothing to
00202 * do other then copying.
00203 * - virtio_net_hdr_f_needs_csum set
00204 * - the packet is partially checksummed; if we would
00205 * send the packet out on the wire we would have
00206 * to calculate checksums now. But here we rely on
00207 * the ability of our guest to handle partially
00208 * checksummed packets and simply delegate the
00209 * checksum calculation to them.
00210 * - gso_type != gso_none
00211 * - the packet needs to be segmented; if we would
00212 * send it out on the wire we would have to
00213 * segment it now. But again we rely on the
00214 * ability of our guest to handle gso
00215 *
00216 * We currently assume that our guests negotiated
00217 * virtio_net_f_guest_*, this needs to be checked in
00218 * the future.
00219 *
00220 * We also discussed the usage of
00221 * virtio_net_hdr_f_data_valid to remove the need to
00222 * checksum packets at all. But since our clients send
00223 * partially checksummed packets anyway the only

```

```

00223 * interesting case would be a packet without
00224 * net_hdr_f_needs_checksum set. In that case we would
00225 * signal that we checked the checksum and the
00226 * checksum is actually correct. Since we do not know
00227 * the origin of the packet (it could have been send
00228 * by an external node and could have been routed to
00229 * u) we can not signal this without actually
00230 * verifying the checksum. Otherwise a packet with an
00231 * invalid checksum could be successfully delivered.
00232 */
00233 total = sizeof(Virtio_net::Hdr);
00234 src.copy_header(dst_header);
00235 mangle.rewrite_hdr(dst_header);
00236 dst.skip(total);
00237 }
00238 ++num_merged;
00239 }
00240
00241 bool has_dst_buffer = !dst.done();
00242 if (!has_dst_buffer)
00243 try
00244 {
00245 // The current dst buffer is full, try to get next chained buffer.
00246 has_dst_buffer = dst_req_proc.next(dst_dev->mem_info(), &dst);
00247 }
00248 catch (L4virtio::Svr::Bad_descriptor &e)
00249 {
00250 Dbg(Dbg::Request, Dbg::Warn, "REQ")
00251 .printf("%s: bad descriptor exception: %s - %i"
00252 " -- signal device error in destination device %p.\n",
00253 __PRETTY_FUNCTION__, e.message(), e.error, dst_dev);
00254 dst_dev->device_error();
00255 return Result::Exception; // Must not touch the dst queues anymore.
00256 }
00257
00258 if (has_dst_buffer)
00259 {
00260 auto &src_buf = src.cur_buf();
00261 trace.printf("\tCopying %p#%p:%u (%x) -> %p#%p:%u (%x)\n",
00262 src_port, src_buf.pos, src_buf.left, src_buf.left,
00263 static_cast<Port_iface*>(this),
00264 dst.pos, dst.left, dst.left);
00265
00266 total += mangle.copy_pkt(dst, src_buf);
00267 }
00268 else if (negotiated_features().mrg_rxbuf())
00269 {
00270 // save descriptor information for later
00271 trace.printf("\tSaving descriptor for later\n");
00272 consumed.push_back(Consumed_entry(dst_head, total));
00273 total_merged += total;
00274 total = 0;
00275 dst_head = L4virtio::Svr::Virtqueue::Head_desc();
00276 }
00277 else
00278 {
00279 trace.printf("\tTransfer failed, destination buffer too small, dropping.\n");
00280 // Abort incomplete transfer.
00281 dst_queue->rewind_avail(dst_head);
00282 return Result::Dropped;
00283 }
00284 }
00285
00286 /*
00287 * Finalize the Request delivery. Call `finish()` on the destination
00288 * port's receive queue, which will result in triggering the destination
00289 * client IRQ.
00290 */
00291
00292 if (!dst_header)
00293 {
00294 if (!total)
00295 trace.printf("\tTransfer - not started yet, dropping\n");
00296 return Result::Dropped;
00297 }
00298
00299 if (consumed.empty())
00300 {
00301 assert(dst_head);
00302 assert(num_merged == 1);
00303 trace.printf("\tTransfer - Invoke dst_queue->finish()\n");
00304 dst_header->num_buffers = 1;
00305 dst_queue->finish(dst_head, dst_dev, total);
00306 *bytes_transferred = total;
00307 }
00308 else
00309 {

```

```

00310 assert(dst_head);
00311 dst_header->num_buffers = num_merged;
00312 consumed.push_back(Consumed_entry(dst_head, total));
00313 trace.printf("\tTransfer - Invoke dst_queue->finish(iter)\n");
00314 *bytes_transferred = total + total_merged;
00315 dst_queue->finish(consumed.begin(), consumed.end(), dst_dev);
00316 }
00317 return Result::Delivered;
00318 }
00319 };
00320

```

## 17.58 request.h

```

00001 /*
00002 * Copyright (C) 2016-2017, 2020, 2022, 2024 Kernkonzept GmbH.
00003 * Author(s): Jean Wolter <jean.wolter@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include "mac_addr.h"
00010 #include "virtio_net.h"
00011 #include "virtio_net_buffer.h"
00012 #include "vlan.h"
00013
00014 #include <l4/l4virtio/server/virtio>
00015
00016
00021
00033 class Net_transfer
00034 {
00035 public:
00036 virtual ~Net_transfer() = default;
00037
00041 void const *req_id() const { return _req_id; }
00042
00046 virtual void copy_header(Virtio_net::Hdr *dst_header) const = 0;
00047
00054 Buffer &cur_buf() { return _cur_buf; }
00055
00065 virtual bool done() = 0;
00066
00067 protected:
00068 Buffer _cur_buf;
00069 void const *_req_id;
00070 };
00071
00072 class Net_request
00073 {
00074 public:
00075
00076 bool has_vlan() const
00077 {
00078 if (!_pkt.pos || _pkt.left < 14)
00079 return false;
00080
00081 uint8_t *p = reinterpret_cast<uint8_t *>(_pkt.pos);
00082 return p[12] == 0x81U && p[13] == 0x00U;
00083 }
00084
00085 uint16_t vlan_id() const
00086 {
00087 if (!has_vlan() || _pkt.left < 16)
00088 return VLAN_ID_NATIVE;
00089
00090 uint8_t *p = reinterpret_cast<uint8_t *>(_pkt.pos);
00091 return (uint16_t){p[14]} << 8 | p[15] & 0xffffU;
00092 }
00093
00105 uint8_t const *buffer(size_t *size) const
00106 {
00107 *size = _pkt.left;
00108 return reinterpret_cast<uint8_t const *>(_pkt.pos);
00109 }
00110
00111 void dump_pkt() const
00112 {
00113 Dbg pkt_debug(Dbg::Packet, Dbg::Debug, "PKT");
00114 if (pkt_debug.is_active())
00115 {
00116 //pkt_debug.cprintf("\t");

```



```

00117 //src_mac().print(pkt_debug);
00118 //pkt_debug.cprintf(" -> ");
00119 //dst_mac().print(pkt_debug);
00120 //pkt_debug.cprintf("\n");
00121
00122 Dbg pkt_trace(Dbg::Packet, Dbg::Trace, "PKT");
00123 if (pkt_trace.is_active() && _pkt.left >= 14)
00124 {
00125 uint8_t const *packet = reinterpret_cast<uint8_t const *>(_pkt.pos);
00126 pkt_trace.cprintf("\n\tEthertype: ");
00127 uint16_t ether_type = uint16_t(packet[12]) << 8 | packet[13];
00128 char const *protocol;
00129 switch (ether_type)
00130 {
00131 case 0x0800: protocol = "IPv4"; break;
00132 case 0x0806: protocol = "ARP"; break;
00133 case 0x8100: protocol = "Vlan"; break;
00134 case 0x86dd: protocol = "IPv6"; break;
00135 case 0x8863: protocol = "PPPoE Discovery"; break;
00136 case 0x8864: protocol = "PPPoE Session"; break;
00137 default: protocol = nullptr;
00138 }
00139 if (protocol)
00140 pkt_trace.cprintf("%s\n", protocol);
00141 else
00142 pkt_trace.cprintf("%04x\n", ether_type);
00143 }
00144 }
00145 }
00146
00147 protected:
00148 Buffer _pkt;
00149 };
00150

```

## 17.59 request.h

```

00001 /*
00002 * Copyright (C) 2019-2020, 2024 Kernkonzept GmbH.
00003 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 namespace Block_device {
00010
00014 struct Pending_request
00015 {
00016 virtual ~Pending_request() = 0;
00017
00028 virtual int handle_request() = 0;
00029
00036 virtual void fail_request() = 0;
00037 };
00038
00039 inline Pending_request::~Pending_request() = default;
00040
00041 } // namespace

```

## 17.60 request\_ixl.h

```

00001 /*
00002 * Copyright (C) 2024 Kernkonzept GmbH.
00003 * Author(s): Georg Kotheimer <georg.kotheimer@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include "port.h"
00010 #include "request.h"
00011
00012 #include <i4/ixl/memory.h>
00013
00014 #include <utility>
00015
00020

```

```

00021 class Ixl_net_request final : public Net_request
00022 {
00023 public:
00024 class Ixl_net_transfer final : public Net_transfer
00025 {
00026 public:
00027 explicit Ixl_net_transfer(Ixl_net_request const &request)
00028 : _request(request)
00029 {
00030 _cur_buf = Buffer(reinterpret_cast<char *>(request.buf()->data),
00031 request.buf()->size);
00032 _req_id = _request.buf();
00033 }
00034
00035 // delete copy constructor and copy assignment operator
00036 Ixl_net_transfer(Ixl_net_transfer const &) = delete;
00037 Ixl_net_transfer &operator = (Ixl_net_transfer const &) = delete;
00038
00039 void copy_header(Virtio_net::Hdr *dst_header) const override
00040 {
00041 dst_header->flags.data_valid() = 0;
00042 dst_header->flags.need_csum() = 0;
00043 dst_header->gso_type = 0; // GSO_NONE
00044 dst_header->hdr_len = sizeof(Virtio_net::Hdr);
00045 dst_header->gso_size = 0;
00046 dst_header->csum_start = 0;
00047 dst_header->csum_offset = 0;
00048 dst_header->num_buffers = 1;
00049 }
00050
00051 bool done() override { return _cur_buf.done(); }
00052
00053 private:
00054 Ixl_net_request const &_request;
00055 };
00056
00057 void dump_request(Port_iface *port) const
00058 {
00059 Dbg debug(Dbg::Request, Dbg::Debug, "REQ-IXL");
00060 if (debug.is_active())
00061 {
00062 debug.printf("%s: Next packet: %p - %x bytes\n",
00063 port->get_name(), _pkt.pos, _pkt.left);
00064 }
00065 dump_pkt();
00066 }
00067
00068 explicit Ixl_net_request(Ixl::pkt_buf *buf) : _buf(buf)
00069 {
00070 _pkt = Buffer(reinterpret_cast<char *>(buf->data), buf->size);
00071 }
00072
00073 // delete copy constructor and copy assignment operator
00074 Ixl_net_request(Ixl_net_request const &) = delete;
00075 Ixl_net_request &operator=(Ixl_net_request const &) = delete;
00076
00077 // define move constructor and copy assignment operator
00078 Ixl_net_request(Ixl_net_request &&other)
00079 : _buf(other._buf)
00080 {
00081 _pkt = std::move(other._pkt);
00082
00083 // Invalidate other.
00084 other._buf = nullptr;
00085 }
00086
00087 Ixl_net_request &operator=(Ixl_net_request &&other)
00088 {
00089 // Invalidate self.
00090 if (_buf != nullptr)
00091 Ixl::pkt_buf_free(_buf);
00092
00093 _buf = other._buf;
00094 _pkt = std::move(other._pkt);
00095
00096 // Invalidate other.
00097 other._buf = nullptr;
00098
00099 return *this;
00100 }
00101
00102 ~Ixl_net_request()
00103 {
00104 if (_buf != nullptr)
00105 {
00106 Ixl::pkt_buf_free(_buf);
00107 _buf = nullptr;
00108 }
00109 }

```

```

00108 }
00109 }
00110
00112 Mac_addr dst_mac() const
00113 {
00114 return (_pkt.pos && _pkt.left >= Mac_addr::Addr_length)
00115 ? Mac_addr::from_uncached(_pkt.pos)
00116 : Mac_addr(Mac_addr::Addr_unknown);
00117 }
00118
00120 Mac_addr src_mac() const
00121 {
00122 return (_pkt.pos && _pkt.left >= Mac_addr::Addr_length * 2)
00123 ? Mac_addr::from_uncached(_pkt.pos + Mac_addr::Addr_length)
00124 : Mac_addr(Mac_addr::Addr_unknown);
00125 }
00126
00127 Ixl::pkt_buf *buf() const { return _buf; }
00128
00129 Ixl_net_transfer transfer_src() const
00130 { return Ixl_net_transfer(*this); }
00131
00132 private:
00133 Ixl::pkt_buf *_buf;
00134 };
00135

```

## 17.61 request\_l4virtio.h

```

00001 /*
00002 * Copyright (C) 2016-2017, 2020, 2022, 2024 Kernkonzept GmbH.
00003 * Author(s): Jean Wolter <jean.wolter@kernkonzept.com>
00004 * Georg Kotheimer <georg.kotheimer@kernkonzept.com>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include "debug.h"
00011 #include "port.h"
00012 #include "request.h"
00013 #include "virtio_net.h"
00014
00015 #include <l4/l4virtio/server/virtio>
00016 #include <l4/util/assert.h>
00017
00018 #include <optional>
00019 #include <utility>
00020
00025
00035 class Virtio_net_request final : public Net_request
00036 {
00037 public:
00038 class Virtio_net_transfer final : public Net_transfer
00039 {
00040 public:
00041 explicit Virtio_net_transfer(Virtio_net_request const &request)
00042 : _request(request),
00043 // We already looked at the very first buffer to find the target of the
00044 // packet. The request processor of the "parent request" contains the
00045 // current state of the transaction up to this point. Since there might be
00046 // more than one target for the request we have to keep track of our own
00047 // state and need our own request processor instance, which will be
00048 // initialized using the current state of the "parent request".
00049 _req_proc(_request.get_request_processor())
00050 {
00051 // The buffer descriptors used for this transaction and the amount of bytes
00052 // copied to the current target descriptor.
00053 _cur_buf = request.first_buffer();
00054 _req_id = _request.header();
00055 }
00056
00057 // delete copy constructor and copy assignment operator
00058 Virtio_net_transfer(Virtio_net_transfer const &) = delete;
00059 Virtio_net_transfer &operator = (Virtio_net_transfer const &) = delete;
00060
00061 void copy_header(Virtio_net::Hdr *dst_header) const override
00062 {
00063 memcpy(dst_header, _request.header(), sizeof(Virtio_net::Hdr));
00064 }
00065
00066 bool done() override
00067 {

```

```

00068 return _cur_buf.done() && !_req_proc.next(_request.dev()->mem_info(), &_cur_buf);
00069 }
00070
00071 private:
00072 Virtio_net_request const &_request;
00073 L4virtio::Svr::Request_processor _req_proc;
00074 };
00075
00076 void dump_request(Port_iface *port) const
00077 {
00078 Dbg debug(Dbg::Request, Dbg::Debug, "REQ-VIO");
00079 if (debug.is_active())
00080 {
00081 debug.printf("%s: Next packet: %p:%p - %x bytes\n",
00082 port->get_name(), _header, _pkt.pos, _pkt.left);
00083 if (_header->flags.raw || _header->gso_type)
00084 {
00085 debug.cprintf("flags:\t%x\n\t"
00086 "gso_type:\t%x\n\t"
00087 "header len:\t%x\n\t"
00088 "gso size:\t%x\n\t"
00089 "csum start:\t%x\n\t"
00090 "csum offset:\t%x\n\t"
00091 "\tnum buffer:\t%x\n\t",
00092 _header->flags.raw,
00093 _header->gso_type, _header->hdr_len,
00094 _header->gso_size,
00095 _header->csum_start, _header->csum_offset,
00096 _header->num_buffers);
00097 }
00098 }
00099 dump_pkt();
00100 }
00101
00102 // delete copy constructor and copy assignment operator
00103 Virtio_net_request(Virtio_net_request const &) = delete;
00104 Virtio_net_request &operator = (Virtio_net_request const &) = delete;
00105
00106 // define move constructor and copy assignment operator
00107 Virtio_net_request(Virtio_net_request &&other)
00108 : _dev(other._dev),
00109 _queue(other._queue),
00110 _head(std::move(other._head)),
00111 _req_proc(std::move(other._req_proc)),
00112 _header(other._header)
00113 {
00114 _pkt = std::move(other._pkt);
00115
00116 // Invalidate other.
00117 other._queue = nullptr;
00118 }
00119
00120 Virtio_net_request &operator = (Virtio_net_request &&other)
00121 {
00122 // Invalidate self.
00123 finish();
00124
00125 _dev = other._dev;
00126 _queue = other._queue;
00127 _head = std::move(other._head);
00128 _req_proc = std::move(other._req_proc);
00129 _header = other._header;
00130 _pkt = std::move(other._pkt);
00131
00132 // Invalidate other.
00133 other._queue = nullptr;
00134
00135 return *this;
00136 }
00137
00138 Virtio_net_request(Virtio_net *dev, L4virtio::Svr::Virtqueue *queue,
00139 L4virtio::Svr::Virtqueue::Request const &req)
00140 : _dev(dev), _queue(queue)
00141 {
00142 _head = _req_proc.start(_dev->mem_info(), req, &_pkt);
00143
00144 _header = (Virtio_net::Hdr *)_pkt.pos;
00145 L4_uint32_t skipped = _pkt.skip(sizeof(Virtio_net::Hdr));
00146
00147 if (L4_UNLIKELY((skipped != sizeof(Virtio_net::Hdr))
00148 || (_pkt.done() && !_next_buffer(&_pkt))))
00149 {
00150 _header = 0;
00151 Dbg(Dbg::Queue, Dbg::Warn).printf("Invalid request\n");
00152 return;
00153 }
00154 }

```

```

00155
00156 ~Virtio_net_request()
00157 { finish(); }
00158
00159 bool valid() const
00160 { return _header != 0; }
00161
00172 static void drop_requests(Virtio_net *dev,
00173 L4virtio::Svr::Virtqueue *queue)
00174 {
00175 if (L4_UNLIKELY(!queue->ready()))
00176 return;
00177
00178 if (queue->desc_avail())
00179 Dbg(Dbg::Request, Dbg::Debug)
00180 .printf("Dropping incoming packets on monitor port\n");
00181
00182 L4virtio::Svr::Request_processor req_proc;
00183 Buffer pkt;
00184
00185 while (auto req = queue->next_avail())
00186 {
00187 auto head = req_proc.start(dev->mem_info(), req, &pkt);
00188 queue->finish(head, dev, 0);
00189 }
00190 }
00191
00198 static std::optional<Virtio_net_request>
00199 get_request(Virtio_net *dev, L4virtio::Svr::Virtqueue *queue)
00200 {
00201 if (L4_UNLIKELY(!queue->ready()))
00202 return std::nullopt;
00203
00204 if (auto r = queue->next_avail())
00205 {
00206 // Virtio_net_request keeps "a lot of internal state",
00207 // therefore we create the object before creating the
00208 // state.
00209 // We might check later on whether it is possible to
00210 // save the state when we actually have to because a
00211 // transfer is blocking on a port.
00212 auto request = Virtio_net_request(dev, queue, r);
00213 if (request.valid())
00214 return request;
00215 }
00216 return std::nullopt;
00217 }
00218
00219 Buffer const &first_buffer() const
00220 { return _pkt; }
00221
00222 Virtio_net::Hdr const *header() const
00223 { return _header; }
00224
00225 L4virtio::Svr::Request_processor const &get_request_processor() const
00226 { return _req_proc; }
00227
00228 Virtio_net const *dev() const
00229 { return _dev; }
00230
00231 Virtio_net_transfer transfer_src() const
00232 { return Virtio_net_transfer(*this); }
00233
00235 Mac_addr dst_mac() const
00236 {
00237 return (_pkt.pos && _pkt.left >= Mac_addr::Addr_length)
00238 ? Mac_addr(_pkt.pos)
00239 : Mac_addr(Mac_addr::Addr_unknown);
00240 }
00241
00243 Mac_addr src_mac() const
00244 {
00245 return (_pkt.pos && _pkt.left >= Mac_addr::Addr_length * 2)
00246 ? Mac_addr(_pkt.pos + Mac_addr::Addr_length)
00247 : Mac_addr(Mac_addr::Addr_unknown);
00248 }
00249
00250 private:
00252 /* needed for Virtqueue::finish() */
00254 Virtio_net *_dev;
00256 L4virtio::Svr::Virtqueue *_queue;
00257 L4virtio::Svr::Virtqueue::Head_desc _head;
00258
00259 /* the actual request processor, encapsulates the decoding of the request */
00260 L4virtio::Svr::Request_processor _req_proc;
00261

```

```

00262 /* A request to the virtio net layer consists of one or more buffers
00263 containing the Virtio_net::Hdr and the actual packet. To make a
00264 switching decision we need to be able to look at the packet while
00265 still being able access the Virtio_net::Hdr for the actual copy
00266 operation. Therefore we keep track of two locations, the header
00267 location and the start of the packet (which might be in a
00268 different buffer) */
00269 Virtio_net::Hdr *_header;
00270
00271 bool _next_buffer(Buffer *buf)
00272 { return _req_proc.next(_dev->mem_info(), buf); }
00273
00280 void finish()
00281 {
00282 if (_queue == nullptr || !_queue->ready())
00283 return;
00284
00285 Dbg(Dbg::Virtio, Dbg::Trace).printf("%s(%p)\n", __PRETTY_FUNCTION__, this);
00286 _queue->finish(_head, _dev, 0);
00287 _queue = nullptr;
00288 }
00289 };
00290

```

## 17.62 stats.h

```

00001 #include <l4/re/env>
00002 #include <l4/re/dataspace>
00003 #include <l4/re/error_helper>
00004 #include <l4/re/util/cap_alloc>
00005 #include <l4/virtio-net-switch/stats.h>
00006
00007 class Switch_statistics
00008 {
00009 private:
00010 L4Re::Util::Ref_cap<L4Re::Dataspace>::Cap_ds;
00011 Virtio_net_switch::Statistics *_stats;
00012 bool _initialized = false;
00013
00014 Switch_statistics() {}
00015
00016 ~Switch_statistics()
00017 {
00018 if (_initialized)
00019 L4Re::Env::env()->rm()->detach(reinterpret_cast<l4_addr_t>(_stats), 0);
00020 }
00021
00022 l4_size_t _size;
00023
00024 public:
00025 Virtio_net_switch::Statistics *stats()
00026 {
00027 if (_initialized)
00028 return _stats;
00029 else
00030 throw L4::Runtime_error(-L4_EAGAIN, "Statistics not set up.");
00031 }
00032
00033 static Switch_statistics& get_instance()
00034 {
00035 static Switch_statistics instance;
00036 return instance;
00037 }
00038
00039 void initialize(l4_uint64_t num_max_ports)
00040 {
00041 _size = l4_round_page(sizeof(Virtio_net_switch::Statistics)
00042 + sizeof(Virtio_net_switch::Port_statistics) * num_max_ports);
00043 void *addr = malloc(_size);
00044 if (!addr)
00045 throw L4::Runtime_error(-L4_ENOMEM,
00046 "Could not allocate statistics memory.");
00047
00048 memset(addr, 0, _size);
00049 _stats = reinterpret_cast<Virtio_net_switch::Statistics *>(addr);
00050 _initialized = true;
00051 _stats->max_ports = num_max_ports;
00052 }
00053
00054 Virtio_net_switch::Port_statistics *
00055 allocate_port_statistics(char const* name)
00056 {
00057 for (unsigned i = 0; i < _stats->max_ports; ++i)

```

```

00058 {
00059 if (!_stats->port_stats[i].in_use)
00060 {
00061 memset(reinterpret_cast<void*>(&_stats->port_stats[i]), 0,
00062 sizeof(Virtio_net_switch::Port_statistics));
00063 _stats->port_stats[i].in_use = 1;
00064 size_t len = std::min(strlen(name), sizeof(_stats->port_stats[i].name) - 1);
00065 memcpy(_stats->port_stats[i].name, name, len);
00066 _stats->port_stats[i].name[len] = '\\0';
00067 _stats->age++;
00068 return &_stats->port_stats[i];
00069 }
00070 }
00071 return nullptr;
00072 }
00073
00074 inline l4_size_t size()
00075 { return _size; }
00076
00077 Switch_statistics(Switch_statistics const&) = delete;
00078 void operator=(Switch_statistics const &) = delete;
00079 };

```

## 17.63 switch.cc

```

00001 /*
00002 * Copyright (C) 2016-2018, 2020, 2023-2024 Kernkonzept GmbH.
00003 * Author(s): Jean Wolter <jean.wolter@kernkonzept.com>
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #include "debug.h"
00009 #include "switch.h"
00010 #include "filter.h"
00011
00012 Virtio_switch::Virtio_switch(unsigned max_ports)
00013 : _ports(new Port_iface *[max_ports]()),
00014 _max_ports(max_ports)
00015 {
00016 }
00017
00018 int
00019 Virtio_switch::lookup_free_slot()
00020 {
00021 for (unsigned idx = 0; idx < _max_ports; ++idx)
00022 if (!_ports[idx])
00023 return idx;
00024 return -1;
00025 }
00026
00027 bool
00028 Virtio_switch::add_port(Port_iface *port)
00029 {
00030 if (!port->mac().is_unknown())
00031 for (unsigned idx = 0; idx < _max_ports; ++idx)
00032 if (_ports[idx] && _ports[idx]->mac() == port->mac())
00033 {
00034 Dbg(Dbg::Port, Dbg::Warn)
00035 .printf("Rejecting port '%s'. MAC address already in use.\n",
00036 port->get_name());
00037 return false;
00038 }
00039 int idx = lookup_free_slot();
00040 if (idx < 0)
00041 return false;
00042 unsigned uidx = static_cast<unsigned>(idx);
00043 _ports[uidx] = port;
00044 if (_max_used == uidx)
00045 ++_max_used;
00046 return true;
00047 }
00048
00049 bool
00050 Virtio_switch::add_monitor_port(Port_iface *port)
00051 {
00052 if (!_monitor)
00053 _monitor = port;
00054 }

```

```

00059 return true;
00060 }
00061
00062 Dbg(Dbg::Port, Dbg::Warn).printf("'%' already defined as monitor port,"
00063 " rejecting monitor port '%s'\n",
00064 _monitor->get_name(), port->get_name());
00065 return false;
00066 }
00067
00068 void
00069 Virtio_switch::check_ports()
00070 {
00071 for (unsigned idx = 0; idx < _max_used; ++idx)
00072 {
00073 Port_iface *port = _ports[idx];
00074 if (port && port->is_gone())
00075 {
00076 Dbg(Dbg::Port, Dbg::Info)
00077 .printf("Client on port %p has gone. Deleting...\n", port);
00078
00079 _ports[idx] = nullptr;
00080 if (idx == _max_used-1)
00081 --_max_used;
00082
00083 _mac_table.flush(port);
00084 delete(port);
00085 }
00086 }
00087
00088 if (_monitor && _monitor->is_gone())
00089 {
00090 delete(_monitor);
00091 _monitor = nullptr;
00092 }
00093 }
00094
00095 template<typename REQ>
00096 void
00097 Virtio_switch::handle_tx_request(Port_iface *port, REQ const &request)
00098 {
00099 // Trunk ports are required to have a VLAN tag and only accept packets that
00100 // belong to a configured VLAN.
00101 if (port->is_trunk() && !port->match_vlan(request.vlan_id()))
00102 {
00103 // Drop packet.
00104 port->stat_inc_tx_dropped();
00105 return;
00106 }
00107
00108 // Access ports must not be VLAN tagged to prevent double tagging attacks.
00109 if (port->is_access() && request.has_vlan())
00110 {
00111 // Drop packet.
00112 port->stat_inc_tx_dropped();
00113 return;
00114 }
00115
00116 auto handle_request = [](Port_iface *dst_port, Port_iface *src_port,
00117 REQ const &req)
00118 {
00119 auto transfer_src = req.transfer_src();
00120 l4_uint64_t bytes;
00121 auto res = dst_port->handle_request(src_port, transfer_src, &bytes);
00122 switch (res)
00123 {
00124 case Port_iface::Result::Delivered:
00125 dst_port->stat_inc_tx_num();
00126 dst_port->stat_inc_tx_bytes(bytes);
00127 src_port->stat_inc_rx_num();
00128 src_port->stat_inc_rx_bytes(bytes);
00129 break;
00130 case Port_iface::Result::Dropped:
00131 [[fallthrough]];
00132 case Port_iface::Result::Exception:
00133 [[fallthrough]];
00134 default:
00135 dst_port->stat_inc_tx_dropped();
00136 break;
00137 }
00138 };
00139
00140 Mac_addr src = request.src_mac();
00141
00142 auto dst = request.dst_mac();
00143 bool is_broadcast = dst.is_broadcast();
00144 uint16_t vlan = request.has_vlan() ? request.vlan_id() : port->get_vlan();
00145 _mac_table.learn(src, port, vlan);

```



```

00146 if (L4_LIKELY(!is_broadcast))
00147 {
00148 auto *target = _mac_table.lookup(dst, vlan);
00149 if (target)
00150 {
00151 // Do not send packets to the port they came in; they might
00152 // be sent to us by another switch which does not know how
00153 // to reach the target.
00154 if (target != port)
00155 {
00156 handle_request(target, port, request);
00157 if (_monitor && !filter_request(request))
00158 handle_request(_monitor, port, request);
00159 }
00160 return;
00161 }
00162 }
00163
00164 // It is either a broadcast or an unknown destination - send to all
00165 // known ports except the source port
00166 for (unsigned idx = 0; idx < _max_used && _ports[idx]; ++idx)
00167 {
00168 auto *target = _ports[idx];
00169 if (target != port && target->match_vlan(vlan))
00170 handle_request(target, port, request);
00171 }
00172
00173 // Send a copy to the monitor port
00174 if (_monitor && !filter_request(request))
00175 handle_request(_monitor, port, request);
00176 }
00177
00178 template<typename PORT>
00179 void
00180 Virtio_switch::handle_tx_requests(PORT *port, unsigned &num_reqs_handled)
00181 {
00182 while (auto req = port->get_tx_request())
00183 {
00184 req->dump_request(port);
00185 handle_tx_request(port, *req);
00186
00187 if (++num_reqs_handled >= Tx_burst)
00188 // Port has hit its TX burst limit.
00189 break;
00190 }
00191 }
00192
00193 bool
00194 Virtio_switch::handle_l4virtio_port_tx(L4virtio_port *port)
00195 {
00196 /* handle IRQ on one port for the time being */
00197 if (!port->tx_work_pending())
00198 Dbg(Dbg::Port, Dbg::Debug)
00199 .printf("%s: Irq without pending work\n", port->get_name());
00200
00201 unsigned num_reqs_handled = 0;
00202 do
00203 {
00204 port->tx_q()->disable_notify();
00205 port->rx_q()->disable_notify();
00206
00207 if (num_reqs_handled >= Tx_burst)
00208 {
00209 Dbg(Dbg::Port, Dbg::Debug)
00210 .printf(
00211 "%s: Tx burst limit hit, reschedule remaining Tx work.\n",
00212 port->get_name());
00213
00214 // Port has hit its TX burst limit, so for fairness reasons, stop
00215 // processing TX work from this port, and instead reschedule the
00216 // pending work for later.
00217 port->reschedule_pending_tx();
00218 // NOTE: Notifications for this port remain disabled, until eventually
00219 // the reschedule handler calls `handle_l4virtio_port_tx` again.
00220 return false;
00221 }
00222
00223 // Within the loop, to trigger before enabling notifications again.
00224 all_rx_notify_disable_and_remember();
00225
00226 try
00227 {
00228 // throws Bad_descriptor exceptions raised on SRC port
00229 handle_tx_requests(port, num_reqs_handled);
00230 }
00231 catch (L4virtio::Svr::Bad_descriptor &e)
00232 {

```

```

00233 Dbg(Dbg::Port, Dbg::Warn, "REQ")
00234 .printf("%s: caught bad descriptor exception: %s - %i"
00235 " -- Signal device error on device %p.\n",
00236 __PRETTY_FUNCTION__, e.message(), e.error, port);
00237 port->device_error();
00238 all_rx_notify_emit_and_enable();
00239 return false;
00240 }
00241
00242 all_rx_notify_emit_and_enable();
00243
00244 port->tx_q()->enable_notify();
00245 port->rx_q()->enable_notify();
00246
00247 L4virtio::wmb();
00248 L4virtio::rmb();
00249 }
00250 while (port->tx_work_pending());
00251
00252 return true;
00253 }
00254
00255 #if CONFIG_VNS_IXL
00256 bool
00257 Virtio_switch::handle_ixl_port_tx(Ixl_port *port)
00258 {
00259 unsigned num_reqs_handled = 0;
00260
00261 all_rx_notify_disable_and_remember();
00262 handle_tx_requests(port, num_reqs_handled);
00263 all_rx_notify_emit_and_enable();
00264
00265 if (num_reqs_handled >= Tx_burst && port->tx_work_pending())
00266 {
00267 Dbg(Dbg::Port, Dbg::Info)
00268 .printf("%s: Tx burst limit hit, reschedule remaining Tx work.\n",
00269 port->get_name());
00270
00271 // Port has hit its TX burst limit, so for fairness reasons, stop
00272 // processing TX work from this port, and instead reschedule the
00273 // pending work for later.
00274 port->reschedule_pending_tx();
00275 return false;
00276 }
00277
00278 return true;
00279 }
00280 #endif
00281

```

## 17.64 switch.h

```

00001 /*
00002 * Copyright (C) 2016-2017, 2020, 2022-2024 Kernkonzept GmbH.
00003 * Author(s): Jean Wolter <jean.wolter@kernkonzept.com>
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include "port.h"
00011 #include "port_l4virtio.h"
00012 #include "mac_table.h"
00013
00014 #if CONFIG_VNS_IXL
00015 #include "port_ixl.h"
00016 #endif
00017
00033 class Virtio_switch
00034 {
00035 private:
00036 Port_iface **_ports;
00037 Port_iface *_monitor = nullptr;
00038
00039 unsigned _max_ports;
00040 unsigned _max_used = 0;
00041 Mac_table<> _mac_table;
00042
00043 // Limits the number of consecutive TX requests a port can process before
00044 // being interrupted to ensure fairness to other ports.
00045 static constexpr unsigned Tx_burst = 128;
00046

```

```

00047 int lookup_free_slot();
00048
00058 template<typename REQ>
00059 void handle_tx_request(Port_iface *port, REQ const &request);
00060
00061 template<typename PORT>
00062 void handle_tx_requests(PORT *port, unsigned &num_reqs_handled);
00063
00064
00065 void all_rx_notify_emit_and_enable()
00066 {
00067 for (unsigned idx = 0; idx < _max_ports; ++idx)
00068 if (_ports[idx])
00069 _ports[idx]->rx_notify_emit_and_enable();
00070 }
00071
00072 void all_rx_notify_disable_and_remember()
00073 {
00074 for (unsigned idx = 0; idx < _max_ports; ++idx)
00075 if (_ports[idx])
00076 _ports[idx]->rx_notify_disable_and_remember();
00077 }
00078
00079 public:
00085 explicit Virtio_switch(unsigned max_ports);
00086
00095 bool add_port(Port_iface *port);
00096
00105 bool add_monitor_port(Port_iface *port);
00106
00114 void check_ports();
00115
00125 bool handle_l4virtio_port_tx(L4virtio_port *port);
00126
00127 #if CONFIG_VNS_IXL
00137 bool handle_ixl_port_tx(Ixl_port *port);
00138 #endif
00139
00148 int port_available(bool monitor)
00149 {
00150 if (monitor)
00151 return !_monitor ? 0 : -1;
00152
00153 return lookup_free_slot();
00154 }
00155 };

```

## 17.65 virtio\_net.h

```

00001 /*
00002 * Copyright (C) 2016-2017, 2019, 2022-2024 Kernkonzept GmbH.
00003 * Author(s): Jean Wolter <jean.wolter@kernkonzept.com>
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include <l4/re/dataspace>
00011 #include <l4/re/util/unique_cap>
00012
00013 #include <l4/sys/cxx/ipc_epiface>
00014
00015 #include <l4/l4virtio/server/virtio>
00016 #include <l4/l4virtio/server/l4virtio>
00017 #include <l4/l4virtio/l4virtio>
00018
00019 #include "debug.h"
00024 class Virtqueue : public L4virtio::Svr::Virtqueue
00025 {
00026 public:
00027 bool kick_queue()
00028 {
00029 if (no_notify_guest())
00030 return false;
00031
00032 if (_do_kick)
00033 return true;
00034
00035 _kick_pending = true;
00036 return false;
00037 }
00038

```

```

00039 bool kick_enable_get_pending()
00040 {
00041 _do_kick = true;
00042 return _kick_pending;
00043 }
00044
00045 void kick_disable_and_remember()
00046 {
00047 _do_kick = false;
00048 _kick_pending = false;
00049 }
00050
00051 private:
00052 bool _do_kick = true;
00053 bool _kick_pending = false;
00054 };
00055
00071 class Virtio_net :
00072 public L4virtio::Svr::Device,
00073 public L4::Epiface_t<Virtio_net, L4virtio::Device>
00074 {
00075 public:
00076 struct Hdr_flags
00077 {
00078 l4_uint8_t raw;
00079 CXX_BITFIELD_MEMBER(0, 0, need_csum, raw);
00080 CXX_BITFIELD_MEMBER(1, 1, data_valid, raw);
00081 };
00082
00083 struct Hdr
00084 {
00085 Hdr_flags flags;
00086 l4_uint8_t gso_type;
00087 l4_uint16_t hdr_len;
00088 l4_uint16_t gso_size;
00089 l4_uint16_t csum_start;
00090 l4_uint16_t csum_offset;
00091 l4_uint16_t num_buffers;
00092 };
00093
00094 struct Features : L4virtio::Svr::Dev_config::Features
00095 {
00096 Features() = default;
00097 Features(l4_uint32_t raw) : L4virtio::Svr::Dev_config::Features(raw) {}
00098
00099 CXX_BITFIELD_MEMBER(0, 0, csum, raw); // host handles partial csum
00100 CXX_BITFIELD_MEMBER(1, 1, guest_csum, raw); // guest handles partial csum
00101 CXX_BITFIELD_MEMBER(5, 5, mac, raw); // host has given mac
00102 CXX_BITFIELD_MEMBER(6, 6, gso, raw); // host handles packets /w any GSO
00103 CXX_BITFIELD_MEMBER(7, 7, guest_tso4, raw); // guest handles TSOv4 in
00104 CXX_BITFIELD_MEMBER(8, 8, guest_tso6, raw); // guest handles TSOv6 in
00105 CXX_BITFIELD_MEMBER(9, 9, guest_ecn, raw); // guest handles TSO[6] with ECN in
00106 CXX_BITFIELD_MEMBER(10, 10, guest_ufo, raw); // guest handles UFO in
00107 CXX_BITFIELD_MEMBER(11, 11, host_tso4, raw); // host handles TSOv4 in
00108 CXX_BITFIELD_MEMBER(12, 12, host_tso6, raw); // host handles TSOv6 in
00109 CXX_BITFIELD_MEMBER(13, 13, host_ecn, raw); // host handles TSO[6] with ECN in
00110 CXX_BITFIELD_MEMBER(14, 14, host_ufo, raw); // host handles UFO
00111 CXX_BITFIELD_MEMBER(15, 15, mrg_rxbuf, raw); // host can merge receive buffers
00112 CXX_BITFIELD_MEMBER(16, 16, status, raw); // virtio_net_config.status available
00113 CXX_BITFIELD_MEMBER(17, 17, ctrl_vq, raw); // Control channel available
00114 CXX_BITFIELD_MEMBER(18, 18, ctrl_rx, raw); // Control channel RX mode support
00115 CXX_BITFIELD_MEMBER(19, 19, ctrl_vlan, raw); // Control channel VLAN filtering
00116 CXX_BITFIELD_MEMBER(20, 20, ctrl_rx_extra, raw); // Extra RX mode control support
00117 CXX_BITFIELD_MEMBER(21, 21, guest_announce, raw); // Guest can announce device on the network
00118 CXX_BITFIELD_MEMBER(22, 22, mq, raw); // Device supports Receive Flow Steering
00119 CXX_BITFIELD_MEMBER(23, 23, ctrl_mac_addr, raw); // Set MAC address
00120 };
00121
00122 enum
00123 {
00124 Rx = 0,
00125 Tx = 1,
00126 };
00127
00128 struct Net_config_space
00129 {
00130 // The config defining mac address (if VIRTIO_NET_F_MAC aka Features::mac)
00131 l4_uint8_t mac[6];
00132 // currently not used ...
00133 l4_uint16_t status;
00134 l4_uint16_t max_virtqueue_pairs;
00135 };
00136
00137 L4virtio::Svr::Dev_config_t<Net_config_space> _dev_config;
00138
00139 explicit Virtio_net(unsigned vq_max)
00140 : L4virtio::Svr::Device(&_dev_config),

```

```

00141 _dev_config(L4VIRTIO_VENDOR_KK, L4VIRTIO_ID_NET, 2),
00142 _vq_max(vq_max)
00143 {
00144 Features hf(0);
00145 hf.ring_indirect_desc() = true;
00146 hf.mrg_rxbuf() = true;
00147 #if 0
00148 // disable currently unsupported options, but leave them in for
00149 // documentation purposes
00150 hf.csum() = true;
00151 hf.host_tso4() = true;
00152 hf.host_tso6() = true;
00153 hf.host_ufo() = true;
00154 hf.host_ecn() = true;
00155
00156 hf.guest_csum() = true;
00157 hf.guest_tso4() = true;
00158 hf.guest_tso6() = true;
00159 hf.guest_ufo() = true;
00160 hf.guest_ecn() = true;
00161 #endif
00162
00163 _dev_config.host_features(0) = hf.raw;
00164 _dev_config.set_host_feature(L4VIRTIO_FEATURE_VERSION_1);
00165 _dev_config.reset_hdr();
00166
00167 reset_queue_config(Rx, vq_max);
00168 reset_queue_config(Tx, vq_max);
00169 }
00170
00171 void reset() override
00172 {
00173 for (L4virtio::Svr::Virtqueue &q: _q)
00174 q.disable();
00175
00176 reset_queue_config(Rx, _vq_max);
00177 reset_queue_config(Tx, _vq_max);
00178 _dev_config.reset_hdr();
00179 }
00180
00181 template<typename T, unsigned N >
00182 static unsigned array_length(T (&)[N]) { return N; }
00183
00184 int reconfig_queue(unsigned index) override
00185 {
00186 Dbg(Dbg::Virtio, Dbg::Info, "Virtio")
00187 .printf("(%p): Reconfigure queue %d (%p): Status: %02x\n",
00188 this, index, _q + index, _dev_config.status().raw);
00189
00190 if (index >= array_length(_q))
00191 return -L4_ERANGE;
00192
00193 if (setup_queue(_q + index, index, _vq_max))
00194 return 0;
00195
00196 return -L4_EINVAL;
00197 }
00198
00199 void dump_features(Dbg const &dbg, const volatile l4_uint32_t *p)
00200 {
00201 dbg.cprintf("%08x:%08x:%08x:%08x:%08x:%08x:%08x:%08x\n",
00202 p[0], p[1], p[2], p[3], p[4], p[5], p[6], p[17]);
00203 }
00204
00205 void dump_features()
00206 {
00207 Dbg info(Dbg::Virtio, Dbg::Info, "Virtio");
00208 if (!info.is_active())
00209 return;
00210
00211 auto *hdr = _dev_config.hdr();
00212
00213 info.printf("Device %p running (%02x)\n\thost features: ",
00214 this, _dev_config.status().raw);
00215 dump_features(info, hdr->dev_features_map);
00216 info.printf("\tguest features: ");
00217 dump_features(info, hdr->driver_features_map);
00218 }
00219
00220 bool check_features() override
00221 {
00222 _negotiated_features = _dev_config.negotiated_features(0);
00223 return true;
00224 }
00225
00226 bool device_needs_reset() const
00227 { return _dev_config.status().device_needs_reset(); }

```

```

00228
00230 bool check_queues() override
00231 {
00232 for (L4virtio::Svr::Virtqueue &q: _q)
00233 if (!q.ready())
00234 {
00235 reset();
00236 Err().printf("failed to start queues\n");
00237 return false;
00238 }
00239 dump_features();
00240 return true;
00241 }
00242
00243 Server_iface *server_iface() const override
00244 { return L4::Epiface::server_iface(); }
00245
00250 void register_single_driver_irq() override
00251 {
00252 _kick_guest_irq = L4Re::Util::Unique_cap<L4::Irq>(
00253 L4Re::chkcap(server_iface()->template rcv_cap<L4::Irq>(0)));
00254 L4Re::chksys(server_iface()->realloc_rcv_cap(0));
00255 }
00256
00257 void trigger_driver_config_irq() override
00258 {
00259 _dev_config.add_irq_status(L4VIRTIO_IRQ_STATUS_CONFIG);
00260 _kick_guest_irq->trigger();
00261 }
00262
00269 void notify_queue(L4virtio::Svr::Virtqueue *queue)
00270 {
00271 // Downcast to Virtqueue to access kick_queue() - we know that our
00272 // queues have the type Virtqueue.
00273 Virtqueue *q = static_cast<Virtqueue*>(queue);
00274 if (q->kick_queue())
00275 {
00276 _dev_config.add_irq_status(L4VIRTIO_IRQ_STATUS_VRING);
00277 _kick_guest_irq->trigger();
00278 }
00279 }
00280
00281 void kick_emit_and_enable()
00282 {
00283 bool kick_pending = false;
00284
00285 for (auto &q : _q)
00286 kick_pending |= q.kick_enable_get_pending();
00287
00288 if (kick_pending)
00289 {
00290 _dev_config.add_irq_status(L4VIRTIO_IRQ_STATUS_VRING);
00291 _kick_guest_irq->trigger();
00292 }
00293 }
00294
00295 void kick_disable_and_remember()
00296 {
00297 for (auto &q : _q)
00298 q.kick_disable_and_remember();
00299 }
00300
00301 Features negotiated_features() const
00302 { return _negotiated_features; }
00303
00305 Virtqueue *tx_q() { return &_q[Tx]; }
00307 Virtqueue *rx_q() { return &_q[Rx]; }
00309 Virtqueue const *tx_q() const { return &_q[Tx]; }
00311 Virtqueue const *rx_q() const { return &_q[Rx]; }
00312
00313 private:
00314 Features _negotiated_features;
00316 unsigned _vq_max;
00318 Virtqueue _q[2];
00323 L4Re::Util::Unique_cap<L4::Irq> _kick_guest_irq;
00324 };

```

## 17.66 virtio\_net.h

```

00001 /* SPDX-License-Identifier: MIT */
00002 /*
00003 * Copyright (C) 2022, 2024 Kernkonzept GmbH.
00004 * Author(s): Stephan Gerhold <stephan.gerhold@kernkonzept.com>

```

```

00005 */
00006
00007 #pragma once
00008
00014
00015 #include <l4/sys/types.h>
00016
00020 typedef struct l4virtio_net_header_t
00021 {
00022 l4_uint8_t flags;
00023 l4_uint8_t gso_type;
00024 l4_uint16_t hdr_len;
00025 l4_uint16_t gso_size;
00026 l4_uint16_t csum_start;
00027 l4_uint16_t csum_offset;
00028 l4_uint16_t num_buffers;
00029 } l4virtio_net_header_t;
00030
00034 typedef struct l4virtio_net_config_t
00035 {
00036 l4_uint8_t mac[6];
00037 l4_uint16_t status;
00038 l4_uint16_t max_virtqueue_pairs;
00039 l4_uint16_t mtu;
00040 l4_uint32_t speed;
00041 l4_uint8_t duplex;
00042 } l4virtio_net_config_t;
00043
00045 enum L4virtio_net_feature_bits
00046 {
00047 L4VIRTIO_NET_F_CSUM = 0,
00048 L4VIRTIO_NET_F_GUEST_CSUM = 1,
00049 L4VIRTIO_NET_F_MTU = 3,
00050 L4VIRTIO_NET_F_MAC = 5,
00051 L4VIRTIO_NET_F_GUEST_TSO4 = 7,
00052 L4VIRTIO_NET_F_GUEST_TSO6 = 8,
00053 L4VIRTIO_NET_F_GUEST_ECN = 9,
00054 L4VIRTIO_NET_F_GUEST_UFO = 10,
00055 L4VIRTIO_NET_F_HOST_TSO4 = 11,
00056 L4VIRTIO_NET_F_HOST_TSO6 = 12,
00057 L4VIRTIO_NET_F_HOST_ECN = 13,
00058 L4VIRTIO_NET_F_HOST_UFO = 14,
00059 L4VIRTIO_NET_F_MRG_RXBUF = 15,
00060 L4VIRTIO_NET_F_STATUS = 16,
00061 L4VIRTIO_NET_F_CTRL_VQ = 17,
00062 L4VIRTIO_NET_F_CTRL_RX = 18,
00063 L4VIRTIO_NET_F_CTRL_VLAN = 19,
00064 L4VIRTIO_NET_F_GUEST_ANNOUNCE = 21,
00065 L4VIRTIO_NET_F_MQ = 22,
00066 L4VIRTIO_NET_F_CTRL_MAC_ADDR = 23,
00067 };
00068

```

## 17.67 virtio\_net\_buffer.h

```

00001 /*
00002 * Copyright (C) 2016-2017, 2022, 2024 Kernkonzept GmbH.
00003 * Author(s): Jean Wolter <jean.wolter@kernkonzept.com>
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include <l4/l4virtio/server/l4virtio>
00015
00019 struct Buffer : L4virtio::Svr::Data_buffer
00020 {
00021 Buffer() = default;
00022 Buffer(L4virtio::Svr::Driver_mem_region const *r,
00023 L4virtio::Svr::Virtqueue::Desc const &d,
00024 L4virtio::Svr::Request_processor const *)
00025 {
00026 pos = static_cast<char *>(r->local(d.addr));
00027 left = d.len;
00028 }
00029
00030 Buffer(char *data, l4_uint32_t size)
00031 {
00032 pos = data;
00033 left = size;
00034 }
00035

```

```

00036 template<typename T>
00037 explicit Buffer(T *p) : Data_buffer(p) {};
00038 };
00039

```

## 17.68 vlan.h

```

00001 /*
00002 * Copyright (C) 2020, 2022-2024 Kernkonzept GmbH.
00003 * Author(s): Jan Klötzke <jan.kloetzke@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/cxx/minmax>
00010 #include <l4/l4virtio/server/virtio>
00011 #include <l4/sys/types.h>
00012 #include <string.h>
00013
00014 #include "virtio_net.h"
00015 #include "virtio_net_buffer.h"
00016
00017 namespace {
00018
00019 const l4_uint16_t VLAN_ID_NATIVE = 0xffffU;
00020 const l4_uint16_t VLAN_ID_TRUNK = 0xfffeU;
00021
00022 inline bool vlan_valid_id(l4_uint16_t id)
00023 {
00024 return id > 0U && id < 0xffffU;
00025 }
00026
00027 }
00028
00029
00030
00031
00032
00033
00034
00035
00036 class Virtio_vlan_mangle
00037 {
00038 l4_uint16_t _tci;
00039 l4_uint8_t _mac_remaining;
00040 l4_int8_t _tag_remaining;
00041
00042 constexpr Virtio_vlan_mangle(l4_uint16_t tci, l4_int8_t tag_remaining)
00043 : _tci{tci}, _mac_remaining{12}, _tag_remaining{tag_remaining}
00044 {}
00045
00046 public:
00047 Virtio_vlan_mangle()
00048 : _tci{0}, _mac_remaining{0}, _tag_remaining{0}
00049 {}
00050
00051
00052 static constexpr Virtio_vlan_mangle add(l4_uint16_t tci)
00053 {
00054 return Virtio_vlan_mangle(tci, 4);
00055 }
00056
00057
00058 static constexpr Virtio_vlan_mangle remove()
00059 {
00060 return Virtio_vlan_mangle(0xffffU, -4);
00061 }
00062
00063
00064
00065
00066
00067
00068
00069
00070
00071
00072
00073
00074
00075
00076
00077
00078
00079
00080
00081
00082
00083
00084
00085
00086
00087
00088
00089
00090
00091
00092
00093 l4_uint32_t copy_pkt(Buffer &dst, Buffer &src)
00094 {
00095 l4_uint32_t ret;
00096
00097 if (L4_LIKELY(_tci == 0))
00098 {
00099 // pass through (no tag or keep tag)
00100 ret = src.copy_to(&dst);
00101 }
00102 else if (_mac_remaining)
00103 {
00104 // copy initial MAC addresses
00105 ret = src.copy_to(&dst, _mac_remaining);
00106 _mac_remaining -= ret;
00107 }
00108 else if (_tag_remaining > 0)
00109 {
00110 // add VLAN tag
00111 l4_uint8_t tag[4] = {
00112 0x81, 0x00,
00113 static_cast<l4_uint8_t>(_tci >> 8),
00114 static_cast<l4_uint8_t>(_tci & 0xffU)
00115 };
00116 }
00117 }

```



```

00116
00117 ret = cxx::min(static_cast<l4_uint32_t>(_tag_remaining), dst.left);
00118 memcpy(dst.pos, &tag[4 - _tag_remaining], ret);
00119 dst.skip(ret);
00120 _tag_remaining -= (int)ret;
00121 }
00122 else if (_tag_remaining < 0)
00123 {
00124 // remove VLAN tag
00125 _tag_remaining += static_cast<int>(src.skip(-_tag_remaining));
00126 ret = 0;
00127 }
00128 else
00129 ret = src.copy_to(&dst);
00130
00131 return ret;
00132 }
00133
00142 void rewrite_hdr(Virtio_net::Hdr *hdr)
00143 {
00144 if (L4_UNLIKELY(!_tci != 0 && hdr->flags.need_csum()))
00145 {
00146 if (_tci == 0xffffU)
00147 hdr->csum_start -= 4U;
00148 else
00149 hdr->csum_start += 4U;
00150 }
00151 }
00152 };
00153

```

## 17.69 amd64/l4/util/perform.h File Reference

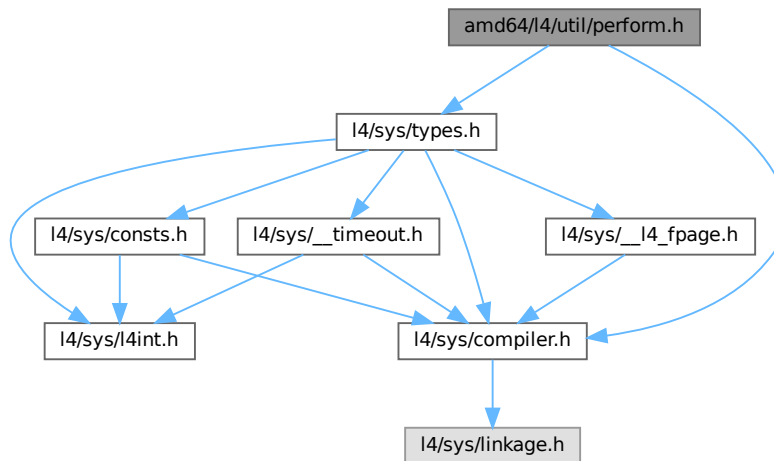
Performance Monitoring using P5/P6 Measurement Counters.

```

#include <l4/sys/types.h>
#include <l4/sys/compiler.h>

```

Include dependency graph for perform.h:



### 17.69.1 Detailed Description

Performance Monitoring using P5/P6 Measurement Counters.

Define either `CPU_PENTIUM` or `CPU_P6`

Definition in file [perform.h](#).

## 17.70 perform.h

[Go to the documentation of this file.](#)

```

00001
00007 /*
00008 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00009 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #ifndef __L4UTIL_PERFORM_H
00014 #define __L4UTIL_PERFORM_H
00015
00016 #include <l4/sys/types.h>
00017 #include <l4/sys/compiler.h>
00018
00019 L4_BEGIN_DECLS
00020
00021 extern const char*strp6pmc_event(l4_uint32_t event);
00022
00023 #ifndef CONFIG_PERFORM_ONLY_PROTOTYPES
00024
00025 #if ! (defined CPU_PENTIUM ^ defined CPU_P6 ^ defined CPU_K7)
00026
00027 #error You must define your target architecture.
00028 #error Define EITHER CPU_PENTIUM for Intel Pentium or CPU_P6 for Intel PPro/PII/PIII.
00029
00030 #else
00031
00032 /* P5/P6/K7 section */
00033
00034 /* Makros for access to model specific registers (MSR) */
00035
00036 /* Write the 64-Bit Model Specific Register. First argument is the register,
00037 second the 64-Bit value. This can only be called at privilege level 0.
00038 With L4, the kernel emulates the WRMSR when calling in PL 3.
00039 */
00040 static inline void l4_i586_wrmsr(unsigned reg,unsigned long long*val){
00041 unsigned long dummyeax, dummyecx, dummyedx;
00042
00043 asm volatile(
00044 ".byte 0xf; .byte 0x30\n" /* wrmsr */
00045 : "=a" (dummyeax), "=d" (dummyedx), "=c" (dummyecx)
00046 : "2" (reg), "0" (*(unsigned *)val), "1" (*(unsigned *)val+1))
00047);
00048 }
00049
00050 /* Read the 64-Bit Model Specific Register. First argument is the register,
00051 second the address to a 64-Bit value. This can only be called at
00052 privilege level 0. With L4, the kernel emulates the RDMSR when calling
00053 in PL 3.
00054 */
00055 static inline void l4_i586_rdmsr(unsigned reg,unsigned long long*val){
00056 unsigned dummy;
00057
00058 asm volatile(
00059 ".byte 0xf; .byte 0x32\n" /* rdmsr */
00060 : "=a" (*(unsigned *)val), "=d" (*(unsigned *)val+1), "=c" (dummy)
00061 : "2" (reg)
00062);
00063 }
00064
00065
00066 #ifndef CPU_PENTIUM
00067 /* Pentium section */
00068
00069 /* functions and events defined here are only usable at Pentium
00070 Processors. P6 architecture does NOT support this kind of measuring and
00071 these events. P6 architecture has its own counters and its own events.
00072 See P6-section for details. */
00073
00074 /* from l4linux/arch/l4-i386/include/perform.h */
00075
00076 static inline void
00077 l4_i586_reset_event_counter(void){
00078 asm volatile("xor %%rax, %%rax\n"
00079 "xor %%rdx, %%rdx\n"
00080 "mov $0x12, %%rcx\n"
00081 ".byte 0x0f, 0x30\n"
00082 "movl $0x13, %%rcx\n"
00083 ".byte 0x0f, 0x30\n"
00084 : : : "cx", "ax", "dx"
00085);
00086 };
00087

```

```

00088 static inline void
00089 l4_i586_read_event_counter_long(long long *counter0, long long *counter1)
00090 {
00091 asm volatile(
00092 /* "movl $0, %%eax\n"
00093 "movl $0x11, %%ecx\n"
00094 ".byte 0x0f, 0x30\n" */ /* stop event counting */
00095 "mov $0x12, %%rcx\n"
00096 ".byte 0x0f, 0x32\n"
00097 "mov %%rax, (%%rbx)\n"
00098 "mov %%rdx, 4(%%rbx)\n"
00099 "mov $0x13, %%ecx\n"
00100 ".byte 0x0f, 0x32\n"
00101 "mov %%rax, (%%rsi)\n"
00102 "mov %%rdx, 4(%%rsi)\n"
00103 : /* no output */
00104 : "b" (counter0), "S" (counter1)
00105 : "ax", "cx", "dx"
00106);
00107 }
00108
00109 static inline void
00110 l4_i586_read_event_counter(int *counter0, int *counter1)
00111 {
00112 asm volatile("push %%rdx\n"
00113 ".byte 0x0f, 0x30\n"
00114 "mov $0x12, %%rcx\n"
00115 ".byte 0x0f, 0x32\n"
00116 "mov %%rax, %%rbx\n"
00117 "movl $0x13, %%rcx\n"
00118 ".byte 0x0f, 0x32\n"
00119 "popl %%edx\n"
00120 : "b" (*counter0), "=a" (*counter1)
00121 : "1" (0), "c" (0x11)
00122);
00123 }
00124
00125 static inline void
00126 l4_i586_select_event(int event0, int event1)
00127 {
00128 asm volatile(".byte 0x0f, 0x30\n"
00129 :
00130 :
00131 "a" (event0 + (event1 << 16)),
00132 "d" (0),
00133 "c" (0x11)
00134);
00135 };
00136
00137 #define P5_RD_MISS 0x003 /* 000011B */
00138 #define P5_WR_MISS 0x008 /* 000100B */
00139 #define P5_RW_MISS 0x029 /* 101001B */
00140 #define P5_EX_MISS 0x00e /* 001110B */
00141
00142 #define P5_D_WBACK 0x006 /* 000110B */
00143
00144 #define P5_RW_TLB 0x002 /* 00010B */
00145 #define P5_EX_TLB 0x00d /* 01101B */
00146
00147 #define P5_A_STALL 0x01f /* 11111B */
00148 #define P5_W_STALL 0x019 /* 11001B */
00149 #define P5_R_STALL 0x01a /* 11010B */
00150 #define P5_X_STALL 0x01b /* 11011B */
00151
00152 #define P5_AGI_STALL 0x01f /* 11111B */
00153
00154 #define P5_PIPELINE_FLUSH 0x015 /* 10101B */
00155
00156 #define P5_NON_CACHE_RD 0x01e /* 11110B */
00157 #define P5_NCACHE_REFS 0x01e /* 11110B */
00158 #define P5_LOCKED_BUS 0x01c /* 11100B */
00159
00160 #define P5_MEM2PIPE 0x009 /* 01001B */
00161 #define P5_BANK_CONF 0x00a /* 01010B */
00162
00163
00164 #define P5_INSTRS_EX 0x016 /* 10110B */
00165 #define P5_INSTRS_EX_V 0x017 /* 10111B */
00166
00167
00168 #define P5_CNT_NOTHING (0x00 << 6) /* 00B << 6 */
00169 #define P5_CNT_EVENT_PL0 (0x01 << 6) /* 01B << 6 */
00170 #define P5_CNT_EVENT_PL3 (0x02 << 6) /* 10B << 6 */
00171 #define P5_CNT_EVENT (0x03 << 6) /* 11B << 6 */
00172 #define P5_CNT_CLOCKS_PL0 (0x05 << 6) /* 101B << 6 */
00173 #define P5_CNT_CLOCKS_PL3 (0x06 << 6) /* 110B << 6 */
00174 #define P5_CNT_CLOCKS (0x07 << 6) /* 111B << 6 */

```

```

00175
00176
00177 #else
00178 #if defined CPU_P6
00179 /* PPro/PII/PIII section */
00180
00181 /*-
00182 * Copyright (c) 1997 The President and Fellows of Harvard College.
00183 * All rights reserved.
00184 * Copyright (c) 1997 Aaron B. Brown.
00185 *
00186 * Redistribution and use in source and binary forms, with or without
00187 * modification, are permitted provided that the following conditions
00188 * are met:
00189 * 1. Redistributions of source code must retain the above copyright
00190 * notice, this list of conditions and the following disclaimer.
00191 * 2. Redistributions in binary form must reproduce the above copyright
00192 * notice, this list of conditions and the following disclaimer in the
00193 * documentation and/or other materials provided with the distribution.
00194 * 3. All advertising materials mentioning features or use of this software
00195 * must display the following acknowledgement:
00196 * This product includes software developed by Harvard University
00197 * and its contributors.
00198 * 4. Neither the name of the University nor the names of its contributors
00199 * may be used to endorse or promote products derived from this software
00200 * without specific prior written permission.
00201 *
00202 * THIS SOFTWARE IS PROVIDED BY HARVARD AND CONTRIBUTORS ``AS IS'' AND
00203 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00204 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00205 * ARE DISCLAIMED. IN NO EVENT SHALL HARVARD UNIVERSITY OR CONTRIBUTORS BE
00206 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00207 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00208 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00209 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00210 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00211 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00212 * POSSIBILITY OF SUCH DAMAGE.
00213 */
00214
00215 /*****
00216 ** Symbolic names for counter numbers (used in select_p6counter()) **
00217 *****/
00218 *
00219 * These correspond in order to the Pentium Pro counters. Add new counters at
00220 * the end. These agree with the mnemonics in the Pentium Pro Family
00221 * Developer's Manual, vol 3.
00222 *
00223 * Those events marked with a $ require a MESI unit field; those marked with
00224 * a @ require a self/any unit field. Those marked with a 0 are only supported
00225 * in counter 0; those marked with 1 are only supported in counter 1.
00226 */
00227
00228 /* Data cache unit */
00229 #define P6_DATA_MEM_REFS 0x43 /* total memory refs */
00230 #define P6_DCU_LINES_IN 0x45 /* all lines allocated in cache unit */
00231 #define P6_DCU_M_LINES_IN 0x46 /* M lines allocated in cache unit */
00232 #define P6_DCU_M_LINES_OUT 0x47 /* M lines evicted from cache */
00233 #define P6_DCU_MISS_OUTSTANDING 0x48 /* #cycles a miss is outstanding */
00234
00235 /* Instruction fetch unit */
00236 #define P6_IFU_IFETCH 0x80 /* instruction fetches */
00237 #define P6_IFU_IFETCH_MISS 0x81 /* instruction fetch misses */
00238 #define P6_ITLB_MISS 0x85 /* ITLB misses */
00239 #define P6_IFU_MEM_STALL 0x86 /* number of cycles IFU is stalled */
00240 #define P6_ILD_STALL 0x87 /* #stalls in instr length decode */
00241
00242 /* L2 Cache */
00243 #define P6_L2_IFETCH 0x28 /* ($) 12 ifetches */
00244 #define P6_L2_LD 0x29 /* ($) 12 data loads */
00245 #define P6_L2_ST 0x2a /* ($) 12 data stores */
00246 #define P6_L2_LINES_IN 0x24 /* lines allocated in L2 */
00247 #define P6_L2_LINES_OUT 0x26 /* lines removed from L2 */
00248 #define P6_L2_M_LINES_INM 0x25 /* modified lines allocated in L2 */
00249 #define P6_L2_M_LINES_OUTM 0x27 /* modified lines removed from L2 */
00250 #define P6_L2_RQSTS 0x2e /* ($) number of 12 requests */
00251 #define P6_L2_ADS 0x21 /* number of 12 addr strobes */
00252 #define P6_L2_DBUS_BUSY 0x22 /* number of data bus busy cycles */
00253 #define P6_L2_DBUS_BUSY_RD 0x23 /* #bus cycles xferring 12->cpu */
00254
00255 /* External bus logic */
00256 #define P6_BUS_DRDY_CLOCKS 0x62 /* (@) #clocks DRDY is asserted */
00257 #define P6_BUS_LOCK_CLOCKS 0x63 /* (@) #clocks LOCK is asserted */
00258 #define P6_BUS_REQ_OUTSTANDING 0x60 /* #bus requests outstanding */
00259 #define P6_BUS_TRAN_BRD 0x65 /* (@) bus burst read txns */
00260 #define P6_BUS_TRAN_RFO 0x66 /* (@) bus read for ownership txns */
00261 #define P6_BUS_TRAN_WB 0x67 /* (@) bus writeback txns */

```

```

00262 #define P6_BUS_TRAN_IFETCH 0x68 /* (@) bus instr fetch txns */
00263 #define P6_BUS_TRAN_INVALID 0x69 /* (@) bus invalidate txns */
00264 #define P6_BUS_TRAN_PWR 0x6a /* (@) bus partial write txns */
00265 #define P6_BUS_TRANS_P 0x6b /* (@) bus partial txns */
00266 #define P6_BUS_TRANS_IO 0x6c /* (@) bus I/O txns */
00267 #define P6_BUS_TRAN_DEF 0x6d /* (@) bus deferred txns */
00268 #define P6_BUS_TRAN_BURST 0x6e /* (@) bus burst txns */
00269 #define P6_BUS_TRAN_ANY 0x70 /* (@) total bus txns */
00270 #define P6_BUS_TRAN_MEM 0x6f /* (@) total memory txns */
00271 #define P6_BUS_DATA_RCV 0x64 /* #busclocks CPU is receiving data */
00272 #define P6_BUS_BNR_DRV 0x61 /* #busclocks CPU is driving BNR pin */
00273 #define P6_BUS_HIT_DRV 0x7a /* #busclocks CPU is driving HIT pin */
00274 #define P6_BUS_HITM_DRV 0x7b /* #busclocks CPU is driving HITM pin */
00275 #define P6_BUS_SNOOP_STALL 0x7e /* #clkcycles bus is snoop-stalled */
00276
00277 /* FPU */
00278 #define P6_FLOPS 0xc1 /* (0) number of FP ops retired */
00279 #define P6_FP_COMP_OPS 0x10 /* (0) computational FPOPS exec'd */
00280 #define P6_FP_ASSIST 0x11 /* (1) FP excep's handled in ucode */
00281 #define P6_MUL 0x12 /* (1) number of FP multiplies */
00282 #define P6_DIV 0x13 /* (1) number of FP divides */
00283 #define P6_CYCLES_DIV_BUSY 0x14 /* (0) number of cycles divider busy */
00284
00285 /* Memory ordering */
00286 #define P6_LD_BLOCKS 0x03 /* number of store buffer blocks */
00287 #define P6_SB_DRAINS 0x04 /* # of store buffer drain cycles */
00288 #define P6_MISALING_MEM_REF 0x05 /* # misaligned data memory refs */
00289
00290 /* Instruction decoding and retirement */
00291 #define P6_INST_RETIRED 0xc0 /* number of instrs retired */
00292 #define P6_UOPS_RETIRED 0xc2 /* number of micro-ops retired */
00293 #define P6_INST_DECODER 0xd0 /* number of instructions decoded */
00294
00295 /* Interrupts */
00296 #define P6_HW_INT_RX 0xc8 /* number of hardware interrupts */
00297 #define P6_CYCLES_INT_MASKED 0xc6 /* number of cycles hardints masked */
00298 #define P6_CYCLES_INT_PENDING_AND_MASKED 0xc7 /* #cycles masked but pending */
00299
00300 /* Branches */
00301 #define P6_BR_INST_RETIRED 0xc4 /* number of branch instrs retired */
00302 #define P6_BR_MISS_PRED_RETIRED 0xc5 /* number of mispred'd brs retired */
00303 #define P6_BR_TAKEN_RETIRED 0xc9 /* number of taken branches retired */
00304 #define P6_BR_MISS_PRED_TAKEN_RET 0xca /* #taken mispredictions br's retired */
00305 #define P6_BR_INST_DECODED 0xe0 /* number of branch instrs decoded */
00306 #define P6_BTБ_MISSES 0xe2 /* # of branches that missed in BTB */
00307 #define P6_BR_BOGUS 0xe4 /* number of bogus branches */
00308 #define P6_BACLEAR 0xe6 /* # times BACLEAR is asserted */
00309
00310 /* Stalls */
00311 #define P6_RESOURCE_STALLS 0xa2 /* # resource-related stall cycles */
00312 #define P6_PARTIAL_RAT_STALLS 0xd2 /* # cycles/events for partial stalls */
00313
00314 /* Segment register loads */
00315 #define P6_SEGMENT_REG_LOADS 0x06 /* number of segment register loads */
00316
00317 /* Clocks */
00318 #define P6_CPU_CLK_UNHALTED 0x79 /* #clocks CPU is not halted */
00319
00320 /* Unit field tags */
00321 #define P6_UNIT_M 0x0800
00322 #define P6_UNIT_E 0x0400
00323 #define P6_UNIT_S 0x0200
00324 #define P6_UNIT_I 0x0100
00325 #define P6_UNIT_MESI 0x0f00
00326
00327 #define P6_UNIT_SELF 0x0000
00328 #define P6_UNIT_ANY 0x2000
00329
00330 /*****
00331 ** Flag bit definitions (used for the 'flag' field in select_p6counter()) **
00332 *****/
00333 *
00334 * The driver accepts fully-formed counter specifications from user-level.
00335 * The following flags are mnemonics for the bits that get set in the
00336 * PerfEvtSel0 and PerfEvtSel1 MSR's
00337 *
00338 */
00339 #define P6CNT_U 0x010000 /* Monitor user-level events */
00340 #define P6CNT_K 0x020000 /* Monitor kernel-level events */
00341 #define P6CNT_E 0x040000 /* Edge detect: count state transitions */
00342 #define P6CNT_PC 0x080000 /* Pin control: ?? */
00343 #define P6CNT_IE 0x100000 /* Int enable: enable interrupt on overflow */
00344 #define P6CNT_F 0x200000 /* Freeze counter (handled in software) */
00345 #define P6CNT_EN 0x400000 /* enable counters (in PerfEvtSel0) */
00346 #define P6CNT_IV 0x800000 /* Invert counter mask comparison result */
00347
00348 /*****

```



```

00436 #endif
00437
00438 #endif
00439
00440 /* end of P5/P6/K7 section*/
00441 #endif
00442
00443 /* end of not only lib-prototypes section */
00444 #endif
00445
00446 L4_END_DECLS
00447
00448 #endif

```

## 17.71 x86/l4/util/perform.h File Reference

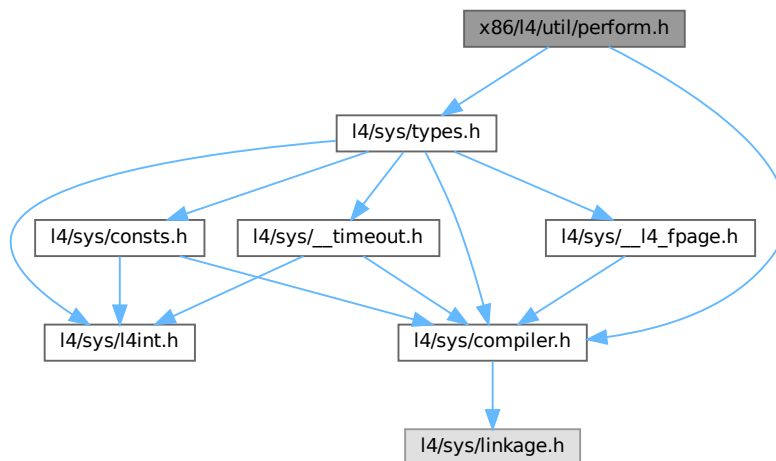
Performance Monitoring using P5/P6 Measurement Counters.

```

#include <l4/sys/types.h>
#include <l4/sys/compiler.h>

```

Include dependency graph for perform.h:



### 17.71.1 Detailed Description

Performance Monitoring using P5/P6 Measurement Counters.

Define either `CPU_PENTIUM` or `CPU_P6`

Definition in file [perform.h](#).

## 17.72 perform.h

[Go to the documentation of this file.](#)

```

00001
00007 /*
00008 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00009 * Frank Mehnert <fm3@os.inf.tu-dresden.de>,
00010 * Lars Reuther <reuther@os.inf.tu-dresden.de>
00011 * economic rights: Technische Universität Dresden (Germany)
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014
00015 #ifndef __L4UTIL_PERFORM_H
00016 #define __L4UTIL_PERFORM_H
00017
00018 #include <l4/sys/types.h>
00019 #include <l4/sys/compiler.h>
00020
00021 L4_BEGIN_DECLS
00022
00023 extern const char*strp6pmc_event(l4_uint32_t event);
00024
00025 #ifndef CONFIG_PERFORM_ONLY_PROTOTYPES
00026
00027 #if ! (defined CPU_PENTIUM ^ defined CPU_P6 ^ defined CPU_K7)
00028
00029 #error You must define your target architecture.
00030 #error Define EITHER CPU_PENTIUM for Intel Pentium or CPU_P6 for Intel PPro/PII/PIII.
00031
00032 #else
00033
00034 /* P5/P6/K7 section */
00035
00036 /* Makros for access to model specific registers (MSR) */
00037
00038 /* Write the 64-Bit Model Specific Register. First argument is the register,
00039 second the 64-Bit value. This can only be called at privilege level 0.
00040 With L4, the kernel emulates the WRMSR when calling in PL 3.
00041 */
00042 static inline void l4_i586_wrmsr(unsigned reg,unsigned long long*val){
00043 unsigned long dummyeax, dummyecx, dummyedx;
00044
00045 asm volatile(
00046 ".byte 0xf; .byte 0x30\n" /* wrmsr */
00047 : "a" (dummyeax), "d" (dummyedx), "c" (dummyecx)
00048 : "2" (reg), "0" (*(unsigned *)val), "1" (*(unsigned *)val+1))
00049);
00050 }
00051
00052 /* Read the 64-Bit Model Specific Register. First argument is the register,
00053 second the address to a 64-Bit value. This can only be called at
00054 privilege level 0. With L4, the kernel emulates the RDMSR when calling
00055 in PL 3.
00056 */
00057 static inline void l4_i586_rdmsr(unsigned reg,unsigned long long*val){
00058 unsigned dummy;
00059
00060 asm volatile(
00061 ".byte 0xf; .byte 0x32\n" /* rdmsr */
00062 : "a" (*(unsigned *)val), "d" (*(unsigned *)val+1), "c" (dummy)
00063 : "2" (reg)
00064);
00065 }
00066
00067
00068 #ifdef CPU_PENTIUM
00069 /* Pentium section */
00070
00071 /* functions and events defined here are only usable at Pentium
00072 Processors. P6 architecture does NOT support this kind of measuring and
00073 these events. P6 architecture has its own counters and its own events.
00074 See P6-section for details. */
00075
00076 /* from l4linux/arch/l4-i386/include/perform.h */
00077
00078 static inline void
00079 l4_i586_reset_event_counter(void){
00080 asm volatile("xor %%eax, %%eax\n"
00081 "xor %%edx, %%edx\n"
00082 "movl $0x12, %%ecx\n"
00083 ".byte 0x0f, 0x30\n"
00084 "movl $0x13, %%ecx\n"
00085 ".byte 0x0f, 0x30\n"
00086 : : "cx", "ax", "dx"
00087);

```



```

00088 };
00089
00090 static inline void
00091 l4_i586_read_event_counter_long(long long *counter0, long long *counter1)
00092 {
00093 asm volatile(
00094 /* "movl $0, %%eax\n"
00095 "movl $0x11, %%ecx\n"
00096 ".byte 0x0f, 0x30\n" */ /* stop event counting */
00097 "movl $0x12, %%ecx\n"
00098 ".byte 0x0f, 0x32\n"
00099 "movl %%eax, (%%ebx)\n"
00100 "movl %%edx, 4(%%ebx)\n"
00101 "movl $0x13, %%ecx\n"
00102 ".byte 0x0f, 0x32\n"
00103 "movl %%eax, (%%esi)\n"
00104 "movl %%edx, 4(%%esi)\n"
00105 : /* no output */
00106 : "b" (counter0), "S" (counter1)
00107 : "ax", "cx", "dx"
00108);
00109 }
00110
00111 static inline void
00112 l4_i586_read_event_counter(int *counter0, int *counter1)
00113 {
00114 asm volatile("pushl %%edx\n"
00115 ".byte 0x0f, 0x30\n"
00116 "movl $0x12, %%ecx\n"
00117 ".byte 0x0f, 0x32\n"
00118 "movl %%eax, %%ebx\n"
00119 "movl $0x13, %%ecx\n"
00120 ".byte 0x0f, 0x32\n"
00121 "popl %%edx\n"
00122 : "=b" (*counter0), "=a" (*counter1)
00123 : "1" (0), "c" (0x11)
00124);
00125 }
00126
00127 static inline void
00128 l4_i586_select_event(int event0, int event1)
00129 {
00130 asm volatile(".byte 0x0f, 0x30\n"
00131 :
00132 :
00133 "a" (event0 + (event1 << 16)),
00134 "d" (0),
00135 "c" (0x11)
00136);
00137 };
00138
00139 #define P5_RD_MISS 0x003 /* 000011B */
00140 #define P5_WR_MISS 0x008 /* 000100B */
00141 #define P5_RW_MISS 0x029 /* 101001B */
00142 #define P5_EX_MISS 0x00e /* 001110B */
00143
00144 #define P5_D_WBACK 0x006 /* 000110B */
00145
00146 #define P5_RW_TLB 0x002 /* 00010B */
00147 #define P5_EX_TLB 0x00d /* 01101B */
00148
00149 #define P5_A_STALL 0x01f /* 11111B */
00150 #define P5_W_STALL 0x019 /* 11001B */
00151 #define P5_R_STALL 0x01a /* 11010B */
00152 #define P5_X_STALL 0x01b /* 11011B */
00153
00154 #define P5_AGI_STALL 0x01f /* 11111B */
00155
00156 #define P5_PIPELINE_FLUSH 0x015 /* 10101B */
00157
00158 #define P5_NON_CACHE_RD 0x01e /* 11110B */
00159 #define P5_NCACHE_REFS 0x01e /* 11110B */
00160 #define P5_LOCKED_BUS 0x01c /* 11100B */
00161
00162 #define P5_MEM2PIPE 0x009 /* 01001B */
00163 #define P5_BANK_CONF 0x00a /* 01010B */
00164
00165
00166 #define P5_INSTRS_EX 0x016 /* 10110B */
00167 #define P5_INSTRS_EX_V 0x017 /* 10111B */
00168
00169
00170 #define P5_CNT_NOTHING (0x00 << 6) /* 00B << 6 */
00171 #define P5_CNT_EVENT_PL0 (0x01 << 6) /* 01B << 6 */
00172 #define P5_CNT_EVENT_PL3 (0x02 << 6) /* 10B << 6 */
00173 #define P5_CNT_EVENT (0x03 << 6) /* 11B << 6 */
00174 #define P5_CNT_CLOCKS_PL0 (0x05 << 6) /* 101B << 6 */

```

```

00175 #define P5_CNT_CLOCKS_PL3 (0x06 << 6) /* 110B << 6 */
00176 #define P5_CNT_CLOCKS (0x07 << 6) /* 111B << 6 */
00177
00178
00179 #else
00180 #if defined CPU_P6
00181 /* PPro/PII/PIII section */
00182
00183 /*-
00184 * Copyright (c) 1997 The President and Fellows of Harvard College.
00185 * All rights reserved.
00186 * Copyright (c) 1997 Aaron B. Brown.
00187 *
00188 * Redistribution and use in source and binary forms, with or without
00189 * modification, are permitted provided that the following conditions
00190 * are met:
00191 * 1. Redistributions of source code must retain the above copyright
00192 * notice, this list of conditions and the following disclaimer.
00193 * 2. Redistributions in binary form must reproduce the above copyright
00194 * notice, this list of conditions and the following disclaimer in the
00195 * documentation and/or other materials provided with the distribution.
00196 * 3. All advertising materials mentioning features or use of this software
00197 * must display the following acknowledgement:
00198 * This product includes software developed by Harvard University
00199 * and its contributors.
00200 * 4. Neither the name of the University nor the names of its contributors
00201 * may be used to endorse or promote products derived from this software
00202 * without specific prior written permission.
00203 *
00204 * THIS SOFTWARE IS PROVIDED BY HARVARD AND CONTRIBUTORS ``AS IS" AND
00205 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
00206 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
00207 * ARE DISCLAIMED. IN NO EVENT SHALL HARVARD UNIVERSITY OR CONTRIBUTORS BE
00208 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
00209 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
00210 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
00211 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00212 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
00213 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
00214 * POSSIBILITY OF SUCH DAMAGE.
00215 */
00216
00217 /*****
00218 ** Symbolic names for counter numbers (used in select_p6counter()) **
00219 *****/
00220 *
00221 * These correspond in order to the Pentium Pro counters. Add new counters at
00222 * the end. These agree with the mnemonics in the Pentium Pro Family
00223 * Developer's Manual, vol 3.
00224 *
00225 * Those events marked with a $ require a MESI unit field; those marked with
00226 * a @ require a self/any unit field. Those marked with a 0 are only supported
00227 * in counter 0; those marked with 1 are only supported in counter 1.
00228 */
00229
00230 /* Data cache unit */
00231 #define P6_DATA_MEM_REFS 0x43 /* total memory refs */
00232 #define P6_DCU_LINES_IN 0x45 /* all lines allocated in cache unit */
00233 #define P6_DCU_M_LINES_IN 0x46 /* M lines allocated in cache unit */
00234 #define P6_DCU_M_LINES_OUT 0x47 /* M lines evicted from cache */
00235 #define P6_DCU_MISS_OUTSTANDING 0x48 /* #cycles a miss is outstanding */
00236
00237 /* Instruction fetch unit */
00238 #define P6_IFU_IFETCH 0x80 /* instruction fetches */
00239 #define P6_IFU_IFETCH_MISS 0x81 /* instruction fetch misses */
00240 #define P6_ITLB_MISS 0x85 /* ITLB misses */
00241 #define P6_IFU_MEM_STALL 0x86 /* number of cycles IFU is stalled */
00242 #define P6_ILD_STALL 0x87 /* #stalls in instr length decode */
00243
00244 /* L2 Cache */
00245 #define P6_L2_IFETCH 0x28 /* ($) 12 ifetches */
00246 #define P6_L2_LD 0x29 /* ($) 12 data loads */
00247 #define P6_L2_ST 0x2a /* ($) 12 data stores */
00248 #define P6_L2_LINES_IN 0x24 /* lines allocated in L2 */
00249 #define P6_L2_LINES_OUT 0x26 /* lines removed from L2 */
00250 #define P6_L2_M_LINES_INM 0x25 /* modified lines allocated in L2 */
00251 #define P6_L2_M_LINES_OUTM 0x27 /* modified lines removed from L2 */
00252 #define P6_L2_RQSTS 0x2e /* ($) number of l2 requests */
00253 #define P6_L2_ADS 0x21 /* number of l2 addr strobes */
00254 #define P6_L2_DBUS_BUSY 0x22 /* number of data bus busy cycles */
00255 #define P6_L2_DBUS_BUSY_RD 0x23 /* #bus cycles xferring l2->cpu */
00256
00257 /* External bus logic */
00258 #define P6_BUS_DRDY_CLOCKS 0x62 /* (@) #clocks DRDY is asserted */
00259 #define P6_BUS_LOCK_CLOCKS 0x63 /* (@) #clocks LOCK is asserted */
00260 #define P6_BUS_REQ_OUTSTANDING 0x60 /* #bus requests outstanding */
00261 #define P6_BUS_TRAN_BRD 0x65 /* (@) bus burst read txns */

```

```

00262 #define P6_BUS_TRAN_RFO 0x66 /* (0) bus read for ownership txns */
00263 #define P6_BUS_TRAN_WB 0x67 /* (0) bus writeback txns */
00264 #define P6_BUS_TRAN_IFETCH 0x68 /* (0) bus instr fetch txns */
00265 #define P6_BUS_TRAN_INVALID 0x69 /* (0) bus invalidate txns */
00266 #define P6_BUS_TRAN_PWR 0x6a /* (0) bus partial write txns */
00267 #define P6_BUS_TRANS_P 0x6b /* (0) bus partial txns */
00268 #define P6_BUS_TRANS_IO 0x6c /* (0) bus I/O txns */
00269 #define P6_BUS_TRAN_DEF 0x6d /* (0) bus deferred txns */
00270 #define P6_BUS_TRAN_BURST 0x6e /* (0) bus burst txns */
00271 #define P6_BUS_TRAN_ANY 0x70 /* (0) total bus txns */
00272 #define P6_BUS_TRAN_MEM 0x6f /* (0) total memory txns */
00273 #define P6_BUS_DATA_RCV 0x64 /* #busclocks CPU is receiving data */
00274 #define P6_BUS_BNR_DRV 0x61 /* #busclocks CPU is driving BNR pin */
00275 #define P6_BUS_HIT_DRV 0x7a /* #busclocks CPU is driving HIT pin */
00276 #define P6_BUS_HITM_DRV 0x7b /* #busclocks CPU is driving HITM pin */
00277 #define P6_BUS_SNOOP_STALL 0x7e /* #clkcycles bus is snoop-stalled */
00278
00279 /* FPU */
00280 #define P6_FLOPS 0xc1 /* (0) number of FP ops retired */
00281 #define P6_FP_COMP_OPS 0x10 /* (0) computational FPOPS exec'd */
00282 #define P6_FP_ASSIST 0x11 /* (1) FP excep's handled in ucode */
00283 #define P6_MUL 0x12 /* (1) number of FP multiplies */
00284 #define P6_DIV 0x13 /* (1) number of FP divides */
00285 #define P6_CYCLES_DIV_BUSY 0x14 /* (0) number of cycles divider busy */
00286
00287 /* Memory ordering */
00288 #define P6_LD_BLOCKS 0x03 /* number of store buffer blocks */
00289 #define P6_SB_DRAINS 0x04 /* # of store buffer drain cycles */
00290 #define P6_MISALIGN_MEM_REF 0x05 /* # misaligned data memory refs */
00291
00292 /* Instruction decoding and retirement */
00293 #define P6_INST_RETIRED 0xc0 /* number of instrs retired */
00294 #define P6_UOPS_RETIRED 0xc2 /* number of micro-ops retired */
00295 #define P6_INST_DECODER 0xd0 /* number of instructions decoded */
00296
00297 /* Interrupts */
00298 #define P6_HW_INT_RX 0xc8 /* number of hardware interrupts */
00299 #define P6_CYCLES_INT_MASKED 0xc6 /* number of cycles hardints masked */
00300 #define P6_CYCLES_INT_PENDING_AND_MASKED 0xc7 /* #cycles masked but pending */
00301
00302 /* Branches */
00303 #define P6_BR_INST_RETIRED 0xc4 /* number of branch instrs retired */
00304 #define P6_BR_MISS_PRED_RETIRED 0xc5 /* number of mispred'd brs retired */
00305 #define P6_BR_TAKEN_RETIRED 0xc9 /* number of taken branches retired */
00306 #define P6_BR_MISS_PRED_TAKEN_RET 0xca /* #taken mispredictions br's retired */
00307 #define P6_BR_INST_DECODED 0xe0 /* number of branch instrs decoded */
00308 #define P6_BTБ_MISSES 0xe2 /* # of branches that missed in BTБ */
00309 #define P6_BR_BOGUS 0xe4 /* number of bogus branches */
00310 #define P6_BACLEAR 0xe6 /* # times BACLEAR is asserted */
00311
00312 /* Stalls */
00313 #define P6_RESOURCE_STALLS 0xa2 /* # resource-related stall cycles */
00314 #define P6_PARTIAL_RAT_STALLS 0xd2 /* # cycles/events for partial stalls */
00315
00316 /* Segment register loads */
00317 #define P6_SEGMENT_REG_LOADS 0x06 /* number of segment register loads */
00318
00319 /* Clocks */
00320 #define P6_CPU_CLK_UNHALTED 0x79 /* #clocks CPU is not halted */
00321
00322 /* Unit field tags */
00323 #define P6_UNIT_M 0x0800
00324 #define P6_UNIT_E 0x0400
00325 #define P6_UNIT_S 0x0200
00326 #define P6_UNIT_I 0x0100
00327 #define P6_UNIT_MESI 0x0f00
00328
00329 #define P6_UNIT_SELF 0x0000
00330 #define P6_UNIT_ANY 0x2000
00331
00332 /*****
00333 ** Flag bit definitions (used for the 'flag' field in select_p6counter()) **
00334 *****/
00335 *
00336 * The driver accepts fully-formed counter specifications from user-level.
00337 * The following flags are mnemonics for the bits that get set in the
00338 * PerfEvtSel0 and PerfEvtSel1 MSR's
00339 *
00340 */
00341 #define P6CNT_U 0x010000 /* Monitor user-level events */
00342 #define P6CNT_K 0x020000 /* Monitor kernel-level events */
00343 #define P6CNT_E 0x040000 /* Edge detect: count state transitions */
00344 #define P6CNT_PC 0x080000 /* Pin control: ?? */
00345 #define P6CNT_IE 0x100000 /* Int enable: enable interrupt on overflow */
00346 #define P6CNT_F 0x200000 /* Freeze counter (handled in software) */
00347 #define P6CNT_EN 0x400000 /* enable counters (in PerfEvtSel0) */
00348 #define P6CNT_IV 0x800000 /* Invert counter mask comparison result */

```

```

00349
00350 /*****
00351 ** Miscellaneous constants **
00352 *****/
00353 *
00354 * Number of Pentium Pro programable hardware counters.
00355 */
00356 #define NUM_P6HWC 2
00357
00358 /*****
00359 *
00360 * End of Copyright by Harvard College
00361 *
00362 *****/
00363
00364
00365 #define MSR_P6_EVTSEL0 0x186
00366 #define MSR_P6_EVTSEL1 0x187
00367 #define MSR_P6_PERFCTR0 0xc1
00368 #define MSR_P6_PERFCTR1 0xc2
00369
00370 /* P6-specific Makros to manipulate and read counters */
00371
00372 /* Read the 40 bit performance monitoring counter. This requires
00373 the PCE-flag in CR4 to be set. Otherwise GP0 is raised. Works only
00374 at P6.
00375 */
00376 #define l4_i686_rdpmc(cntnr, res_p) \
00377 __asm__ __volatile__(\
00378 "movl %2, %%ecx # put counter number in \n\
00379 .byte 0xf; .byte 0x33 # RDPMC instruction \n\
00380 movl %%edx, %1 # High order 32 bits \n\
00381 movl %%eax, %0 # Low order 32 bits" \
00382 : "=g" (*(int *) (res_p)), "=g" (((int *) res_p)+1)) \
00383 : "g" (cntnr) \
00384 : "ecx", "eax", "edx")
00385
00386 static inline l4_uint32_t l4_i686_rdpmc_32(int cntnr){
00387 l4_uint32_t x;
00388
00389 __asm__ __volatile__(
00390 ".byte 0xf; .byte 0x33 # RDPMC instruction"
00391 : "=a" (x)
00392 : "c" (cntnr)
00393 : "ecx", "eax", "edx");
00394 return x;
00395 }
00396
00397 static inline void l4_i686_select_perfctr_event(int counter,
00398 unsigned long long val){
00399 l4_i586_wrmsr(MSR_P6_EVTSEL0+counter, &val);
00400 }
00401
00402 static inline void l4_i686_select_perfctr0_event(long long *val){
00403 __asm__ volatile(
00404 "movl $MSR_P6_EVTSEL0, %%ecx\n"
00405 "movl (%ebx), %%eax\n"
00406 "movl 4(%ebx), %%edx\n"
00407 /* ".byte 0xcc, 0xeb, 0x01, 0x21\n"
00408 ".byte 0x0f, 0x30\n" // wrmsr
00409 /* ".byte 0xcc, 0xeb, 0x01, 0x21\n"
00410 : /* no output */
00411 : "b" (val)
00412 : "ax", "cx", "dx", "bx"
00413);
00414
00415 }
00416
00417 /* end of P6 section */
00418 #else
00419
00420 #define K7CNT_U 0x010000 /* Monitor user-level events */
00421 #define K7CNT_K 0x020000 /* Monitor kernel-level events */
00422 #define K7CNT_E 0x040000 /* Edge detect: count state transitions */
00423 #define K7CNT_PC 0x080000 /* Pin control: ?? */
00424 #define K7CNT_IE 0x100000 /* Int enable: enable interrupt on overflow */
00425 #define K7CNT_F 0x200000 /* Freeze counter (handled in software) */
00426 #define K7CNT_EN 0x400000 /* enable counters (in PerfEvtSel0) */
00427 #define K7CNT_IV 0x800000 /* Invert counter mask comparison result */
00428
00429 #define MSR_K7_EVTSEL0 0xC0010000
00430 #define MSR_K7_EVTSEL1 0xC0010001
00431 #define MSR_K7_EVTSEL2 0xC0010002
00432 #define MSR_K7_EVTSEL3 0xC0010003
00433 #define MSR_K7_PERFCTR0 0xC0010004
00434 #define MSR_K7_PERFCTR1 0xC0010005
00435 #define MSR_K7_PERFCTR2 0xC0010006

```

```

00436 #define MSR_K7_PERFCTR3 0xC0010007
00437
00438 #endif
00439
00440 #endif
00441
00442 /* end of P5/P6/K7 section*/
00443 #endif
00444
00445 /* end of not only lib-prototypes section */
00446 #endif
00447
00448 L4_END_DECLS
00449
00450 #endif

```

## 17.73 amd64/l4/util/rdtsc.h File Reference

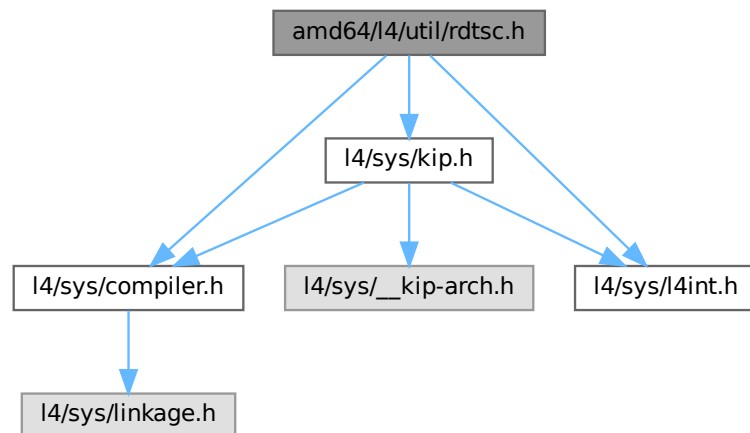
Timestamp counter related functions.

```

#include <l4/sys/compiler.h>
#include <l4/sys/l4int.h>
#include <l4/sys/kip.h>

```

Include dependency graph for rdtsc.h:



### Functions

- `l4_cpu_time_t l4_rdtsc` (void)  
Read current value of CPU-internal timestamp counter.
- `l4_uint32_t l4_rdtsc_32` (void)  
Read the lest significant 32 bit of the TSC.
- `l4_uint64_t l4_rdpmc` (int ecx)  
Return current value of CPU-internal performance measurement counter.
- `l4_uint32_t l4_rdpmc_32` (int ecx)  
Return the least significant 32 bit of a performance counter.
- `l4_uint64_t l4_tsc_to_ns` (`l4_cpu_time_t` tsc)

- Convert timestamp to ns value.*
- [l4\\_uint64\\_t l4\\_tsc\\_to\\_us \(l4\\_cpu\\_time\\_t tsc\)](#)
- Convert timestamp into micro seconds value.*
- [void l4\\_tsc\\_to\\_s\\_and\\_ns \(l4\\_cpu\\_time\\_t tsc, l4\\_uint32\\_t \\*s, l4\\_uint32\\_t \\*ns\)](#)
- Convert timestamp to s.ns value.*
- [l4\\_cpu\\_time\\_t l4\\_ns\\_to\\_tsc \(l4\\_uint64\\_t ns\)](#)
- Convert nano seconds into CPU ticks.*
- [void l4\\_busy\\_wait\\_ns \(l4\\_uint64\\_t ns\)](#)
- Wait busy for a small amount of time.*
- [void l4\\_busy\\_wait\\_us \(l4\\_uint64\\_t us\)](#)
- Wait busy for a small amount of time.*
- [l4\\_uint32\\_t l4\\_calibrate\\_tsc \(l4\\_kernel\\_info\\_t const \\*kip\)](#)
- Determine scalers for timestamp calculations.*
- [l4\\_uint32\\_t l4\\_tsc\\_init \(l4\\_kernel\\_info\\_t const \\*kip\)](#)
- Initialize scaler for TSC calibrations from the kernel.*
- [l4\\_uint32\\_t l4\\_get\\_hz \(void\)](#)
- Get CPU frequency in Hz.*

### 17.73.1 Detailed Description

Timestamp counter related functions.

#### Author

Frank Mehnert [fm3@os.inf.tu-dresden.de](mailto:fm3@os.inf.tu-dresden.de)

Definition in file [rdtsc.h](#).

## 17.74 rdtsc.h

[Go to the documentation of this file.](#)

```

00001
00008
00009 /*
00010 * (c) 2003-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00011 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00012 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00013 * economic rights: Technische Universität Dresden (Germany)
00014 * License: see LICENSE.spdx (in this directory or the directories above)
00015 */
00016
00017 #ifndef __l4_rdtsc_h
00018 #define __l4_rdtsc_h
00019
00024
00025 #include <l4/sys/compiler.h>
00026 #include <l4/sys/l4int.h>
00027 #include <l4/sys/kip.h>
00028
00029 L4_BEGIN_DECLS
00030
00031 /* interface */
00036
00037 extern l4_uint32_t l4_scaler_tsc_to_ns;
00038 extern l4_uint32_t l4_scaler_tsc_to_us;
00039 extern l4_uint32_t l4_scaler_ns_to_tsc;
00040 extern l4_uint32_t l4_scaler_tsc_linux;
00041
00046 L4_INLINE l4_cpu_time_t
00047 l4_rdtsc (void);

```

```

00048
00054 L4_INLINE
00055 l4_uint32_t l4_rdtsc_32(void);
00056
00063 L4_INLINE l4_uint64_t
00064 l4_rdpmc (int ecx);
00065
00071 L4_INLINE
00072 l4_uint32_t l4_rdpmc_32(int ecx);
00073
00078 L4_INLINE l4_uint64_t
00079 l4_tsc_to_ns (l4_cpu_time_t tsc);
00080
00085 L4_INLINE l4_uint64_t
00086 l4_tsc_to_us (l4_cpu_time_t tsc);
00087
00093 L4_INLINE void
00094 l4_tsc_to_s_and_ns (l4_cpu_time_t tsc, l4_uint32_t *s, l4_uint32_t *ns);
00095
00101 L4_INLINE l4_cpu_time_t
00102 l4_ns_to_tsc (l4_uint64_t ns);
00103
00109 L4_INLINE void
00110 l4_busy_wait_ns (l4_uint64_t ns);
00111
00117 L4_INLINE void
00118 l4_busy_wait_us (l4_uint64_t us);
00119
00126 L4_INLINE l4_uint32_t
00127 l4_calibrate_tsc(l4_kernel_info_t const *kip);
00128
00142 L4_CV l4_uint32_t
00143 l4_tsc_init(l4_kernel_info_t const *kip);
00144
00149 L4_CV l4_uint32_t
00150 l4_get_hz (void);
00151
00153
00154 L4_END_DECLS
00155
00156 /* implementation */
00157
00158 L4_INLINE l4_uint32_t
00159 l4_calibrate_tsc(l4_kernel_info_t const *kip)
00160 {
00161 return l4_tsc_init(kip);
00162 }
00163
00164 L4_INLINE l4_cpu_time_t
00165 l4_rdtsc (void)
00166 {
00167 l4_umword_t lo, hi;
00168
00169 __asm__ __volatile__ ("rdtsc" : "=a"(lo), "=d"(hi));
00170
00171 return ((l4_cpu_time_t)hi « 32) | lo;
00172 }
00173
00174 L4_INLINE l4_uint64_t
00175 l4_rdpmc (int ecx)
00176 {
00177 l4_umword_t lo, hi;
00178
00179 __asm__ __volatile__ ("rdpmc" : "=a"(lo), "=d"(hi) : "c"(ecx));
00180
00181 return ((l4_cpu_time_t)hi « 32) | lo;
00182 }
00183
00184 L4_INLINE
00185 l4_uint32_t l4_rdtsc_32(void)
00186 {
00187 l4_umword_t lo, hi;
00188
00189 __asm__ __volatile__ ("rdtsc" : "=a"(lo), "=d"(hi));
00190
00191 return lo;
00192 }
00193
00194 L4_INLINE
00195 l4_uint32_t l4_rdpmc_32(int ecx)
00196 {
00197 l4_umword_t lo, hi;
00198
00199 __asm__ __volatile__ ("rdpmc" : "=a"(lo), "=d"(hi) : "c"(ecx));
00200
00201 return lo;
00202 }

```

```

00203
00204 L4_INLINE l4_uint64_t
00205 l4_tsc_to_ns (l4_cpu_time_t tsc)
00206 {
00207 l4_uint64_t ns, dummy;
00208 __asm__
00209 ("
00210 \"mulq %3 \n\t\"
00211 \"shrd $27, %%rdx, %%rax \n\t\"
00212 :\"=a\" (ns), \"=d\" (dummy)
00213 :\"a\" (tsc), \"r\" ((l4_uint64_t)l4_scaler_tsc_to_ns)
00214);
00215 return ns;
00216 }
00217
00218 L4_INLINE l4_uint64_t
00219 l4_tsc_to_us (l4_cpu_time_t tsc)
00220 {
00221 l4_uint64_t ns, dummy;
00222 __asm__
00223 ("
00224 \"mulq %3 \n\t\"
00225 \"shrd $32, %%rdx, %%rax \n\t\"
00226 :\"=a\" (ns), \"=d\" (dummy)
00227 :\"a\" (tsc), \"r\" ((l4_uint64_t)l4_scaler_tsc_to_us)
00228);
00229 return ns;
00230 }
00231
00232 L4_INLINE void
00233 l4_tsc_to_s_and_ns (l4_cpu_time_t tsc, l4_uint32_t *s, l4_uint32_t *ns)
00234 {
00235 __asm__
00236 ("
00237 \"mulq %3 \n\t\"
00238 \"shrd $27, %%rdx, %%rax \n\t\"
00239 \"xorq %%rdx, %%rdx \n\t\"
00240 \"divq %4 \n\t\"
00241 :\"=a\" (*s), \"=d\" (*ns)
00242 :\"a\" (tsc), \"r\" ((l4_uint64_t)l4_scaler_tsc_to_ns),
00243 \"rm\"(1000000000ULL)
00244);
00245 }
00246
00247 L4_INLINE l4_cpu_time_t
00248 l4_ns_to_tsc (l4_uint64_t ns)
00249 {
00250 l4_uint64_t tsc, dummy;
00251 __asm__
00252 ("
00253 \"mulq %3 \n\t\"
00254 \"shrd $27, %%rdx, %%rax \n\t\"
00255 :\"=a\" (tsc), \"=d\" (dummy)
00256 :\"a\" (ns), \"r\" ((l4_uint64_t)l4_scaler_ns_to_tsc)
00257);
00258 return tsc;
00259 }
00260
00261 L4_INLINE void
00262 l4_busy_wait_ns (l4_uint64_t ns)
00263 {
00264 l4_cpu_time_t stop = l4_rdtsc();
00265 stop += l4_ns_to_tsc(ns);
00266
00267 while (l4_rdtsc() < stop)
00268 ;
00269 }
00270
00271 L4_INLINE void
00272 l4_busy_wait_us (l4_uint64_t us)
00273 {
00274 l4_cpu_time_t stop = l4_rdtsc ();
00275 stop += l4_ns_to_tsc(us*1000ULL);
00276
00277 while (l4_rdtsc() < stop)
00278 ;
00279 }
00280
00281 #endif /* __l4_rdtsc_h */
00282

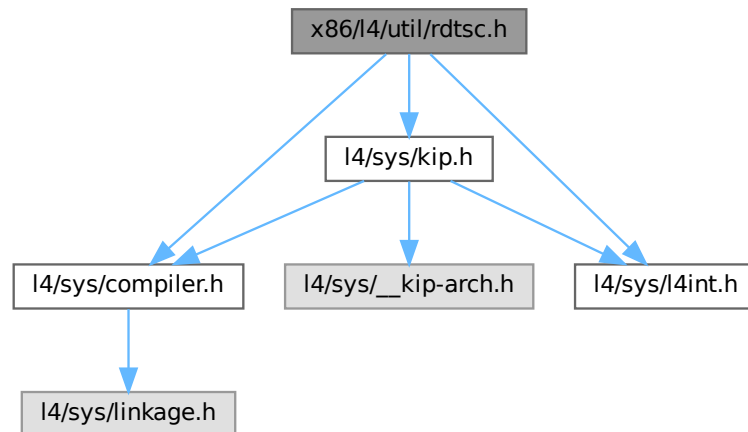
```



## 17.75 x86/l4/util/rdtsc.h File Reference

Timestamp counter related functions.

```
#include <l4/sys/compiler.h>
#include <l4/sys/l4int.h>
#include <l4/sys/kip.h>
Include dependency graph for rdtsc.h:
```



### Functions

- `l4_cpu_time_t l4_rdtsc` (void)  
*Read current value of CPU-internal timestamp counter.*
- `l4_uint32_t l4_rdtsc_32` (void)  
*Read the lest significant 32 bit of the TSC.*
- `l4_uint64_t l4_rdpmc` (int ecx)  
*Return current value of CPU-internal performance measurement counter.*
- `l4_uint32_t l4_rdpmc_32` (int ecx)  
*Return the least significant 32 bit of a performance counter.*
- `l4_uint64_t l4_tsc_to_ns` (`l4_cpu_time_t` tsc)  
*Convert timestamp to ns value.*
- `l4_uint64_t l4_tsc_to_us` (`l4_cpu_time_t` tsc)  
*Convert timestamp into micro seconds value.*
- `void l4_tsc_to_s_and_ns` (`l4_cpu_time_t` tsc, `l4_uint32_t` \*s, `l4_uint32_t` \*ns)  
*Convert timestamp to s.ns value.*
- `l4_cpu_time_t l4_ns_to_tsc` (`l4_uint64_t` ns)  
*Convert nano seconds into CPU ticks.*
- `void l4_busy_wait_ns` (`l4_uint64_t` ns)  
*Wait busy for a small amount of time.*
- `void l4_busy_wait_us` (`l4_uint64_t` us)  
*Wait busy for a small amount of time.*
- `l4_uint32_t l4_calibrate_tsc` (`l4_kernel_info_t` const \*kip)

*Determine scalers for timestamp calculations.*

- [l4\\_uint32\\_t l4\\_tsc\\_init](#) ([l4\\_kernel\\_info\\_t](#) const \*kip)

*Initialize scaler for TSC calibrations from the kernel.*

- [l4\\_uint32\\_t l4\\_get\\_hz](#) (void)

*Get CPU frequency in Hz.*

## 17.75.1 Detailed Description

Timestamp counter related functions.

### Author

Frank Mehnert [fm3@os.inf.tu-dresden.de](mailto:fm3@os.inf.tu-dresden.de)

Definition in file [rdtsc.h](#).

## 17.76 rdtsc.h

[Go to the documentation of this file.](#)

```

00001
00008
00009 /*
00010 * (c) 2003-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00011 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00012 * Frank Mehnert <fm3@os.inf.tu-dresden.de>,
00013 * Jork Löser <jork@os.inf.tu-dresden.de>,
00014 * Martin Pohlack <mp26@os.inf.tu-dresden.de>
00015 * economic rights: Technische Universität Dresden (Germany)
00016 * License: see LICENSE.spdx (in this directory or the directories above)
00017 */
00018
00019 #ifndef __l4_rdtsc_h
00020 #define __l4_rdtsc_h
00021
00026
00027 #include <l4/sys/compiler.h>
00028 #include <l4/sys/l4int.h>
00029 #include <l4/sys/kip.h>
00030
00031 L4_BEGIN_DECLS
00032
00033 /* interface */
00038
00039 extern l4_uint32_t l4_scaler_tsc_to_ns;
00040 extern l4_uint32_t l4_scaler_tsc_to_us;
00041 extern l4_uint32_t l4_scaler_ns_to_tsc;
00042 extern l4_uint32_t l4_scaler_tsc_linux;
00043
00048 L4_INLINE l4_cpu_time_t
00049 l4_rdtsc (void);
00050
00056 L4_INLINE
00057 l4_uint32_t l4_rdtsc_32(void);
00058
00065 L4_INLINE l4_uint64_t
00066 l4_rdpmc (int ecx);
00067
00073 L4_INLINE
00074 l4_uint32_t l4_rdpmc_32(int ecx);
00075
00080 L4_INLINE l4_uint64_t
00081 l4_tsc_to_ns (l4_cpu_time_t tsc);
00082
00087 L4_INLINE l4_uint64_t
00088 l4_tsc_to_us (l4_cpu_time_t tsc);
00089
00095 L4_INLINE void
00096 l4_tsc_to_s_and_ns (l4_cpu_time_t tsc, l4_uint32_t *s, l4_uint32_t *ns);
00097

```

```

00103 L4_INLINE l4_cpu_time_t
00104 l4_ns_to_tsc (l4_uint64_t ns);
00105
00111 L4_INLINE void
00112 l4_busy_wait_ns (l4_uint64_t ns);
00113
00119 L4_INLINE void
00120 l4_busy_wait_us (l4_uint64_t us);
00121
00128 L4_INLINE l4_uint32_t
00129 l4_calibrate_tsc(l4_kernel_info_t const *kip);
00130
00144 L4_CV l4_uint32_t
00145 l4_tsc_init(l4_kernel_info_t const *kip);
00146
00151 L4_CV l4_uint32_t
00152 l4_get_hz (void);
00153
00155
00156 L4_END_DECLS
00157
00158 /* implementation */
00159
00160 L4_INLINE l4_uint32_t
00161 l4_calibrate_tsc(l4_kernel_info_t const *kip)
00162 {
00163 return l4_tsc_init(kip);
00164 }
00165
00166 L4_INLINE l4_cpu_time_t
00167 l4_rdtsc (void)
00168 {
00169 l4_cpu_time_t v;
00170
00171 __asm__ __volatile__ (
00172 (" .byte 0x0f, 0x31 \n\t"
00173 " /* rdtsc\n\t*/"
00174 :
00175 : "=A" (v)
00176 : /* no inputs */
00177);
00178
00179 return v;
00180 }
00181
00182
00183 /* the same, but only 32 bit. Useful for smaller differences,
00184 needs less cycles. */
00185 L4_INLINE
00186 l4_uint32_t l4_rdtsc_32(void)
00187 {
00188 l4_uint32_t x;
00189
00190 __asm__ __volatile__ (
00191 (" .byte 0x0f, 0x31\n\t" // rdtsc
00192 : "=a" (x)
00193 :
00194 : "edx");
00195
00196 return x;
00197 }
00198
00199 L4_INLINE l4_uint64_t
00200 l4_rdpmc (int ecx)
00201 {
00202 l4_cpu_time_t v;
00203
00204 __asm__ __volatile__ (
00205 ("rdpmc \n\t"
00206 :
00207 : "=A" (v)
00208 : "c" (ecx)
00209);
00210
00211 return v;
00212 }
00213
00214 /* the same, but only 32 bit. Useful for smaller differences,
00215 needs less cycles. */
00216 L4_INLINE
00217 l4_uint32_t l4_rdpmc_32(int ecx)
00218 {
00219 l4_uint32_t x;
00220
00221 __asm__ __volatile__ (
00222 ("rdpmc \n\t"
00223 : "=a" (x)

```

```

00224 : "c" (ecx)
00225 : "edx");
00226
00227 return x;
00228 }
00229
00230 L4_INLINE l4_uint64_t
00231 l4_tsc_to_ns (l4_cpu_time_t tsc)
00232 {
00233 l4_uint32_t dummy;
00234 l4_uint64_t ns;
00235 __asm__
00236 ("
00237 \"movl %%edx, %%ecx \n\t\"
00238 \"mull %3 \n\t\"
00239 \"movl %%ecx, %%eax \n\t\"
00240 \"movl %%edx, %%ecx \n\t\"
00241 \"mull %3 \n\t\"
00242 \"addl %%ecx, %%eax \n\t\"
00243 \"adcl $0, %%edx \n\t\"
00244 \"shld $5, %%eax, %%edx \n\t\"
00245 \"shll $5, %%eax \n\t\"
00246 :\"=A\" (ns),
00247 \"=&c\" (dummy)
00248 :\"0\" (tsc),
00249 \"g\" (l4_scaler_tsc_to_ns)
00250);
00251 return ns;
00252 }
00253
00254 L4_INLINE l4_uint64_t
00255 l4_tsc_to_us (l4_cpu_time_t tsc)
00256 {
00257 l4_uint32_t dummy;
00258 l4_uint64_t us;
00259 __asm__
00260 ("
00261 \"movl %%edx, %%ecx \n\t\"
00262 \"mull %3 \n\t\"
00263 \"movl %%ecx, %%eax \n\t\"
00264 \"movl %%edx, %%ecx \n\t\"
00265 \"mull %3 \n\t\"
00266 \"addl %%ecx, %%eax \n\t\"
00267 \"adcl $0, %%edx \n\t\"
00268 :\"=A\" (us),
00269 \"=&c\" (dummy)
00270 :\"0\" (tsc),
00271 \"g\" (l4_scaler_tsc_to_us)
00272);
00273 return us;
00274 }
00275
00276 L4_INLINE void
00277 l4_tsc_to_s_and_ns (l4_cpu_time_t tsc, l4_uint32_t *s, l4_uint32_t *ns)
00278 {
00279 l4_uint32_t dummy;
00280 __asm__
00281 ("
00282 \"movl %%edx, %%ecx \n\t\"
00283 \"mull %4 \n\t\"
00284 \"movl %%ecx, %%eax \n\t\"
00285 \"movl %%edx, %%ecx \n\t\"
00286 \"mull %4 \n\t\"
00287 \"addl %%ecx, %%eax \n\t\"
00288 \"adcl $0, %%edx \n\t\"
00289 \"movl $1000000000, %%ecx \n\t\"
00290 \"shld $5, %%eax, %%edx \n\t\"
00291 \"shll $5, %%eax \n\t\"
00292 \"divl %%ecx \n\t\"
00293 :\"=a\" (*s), \"=d\" (*ns), \"=&c\" (dummy)
00294 : \"A\" (tsc), \"g\" (l4_scaler_tsc_to_ns)
00295);
00296 }
00297
00298 L4_INLINE l4_cpu_time_t
00299 l4_ns_to_tsc (l4_uint64_t ns)
00300 {
00301 l4_uint32_t dummy;
00302 l4_cpu_time_t tsc;
00303 __asm__
00304 ("
00305 \"movl %%edx, %%ecx \n\t\"
00306 \"mull %3 \n\t\"
00307 \"movl %%ecx, %%eax \n\t\"
00308 \"movl %%edx, %%ecx \n\t\"
00309 \"mull %3 \n\t\"
00310 \"addl %%ecx, %%eax \n\t\"

```

```

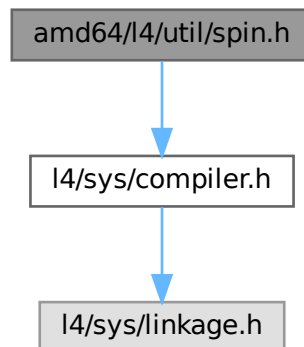
00311 "adcl $0, %%edx \n\t"
00312 "shld $5, %%eax, %%edx \n\t"
00313 "shll $5, %%eax \n\t"
00314 : "=A" (tsc),
00315 "=&c" (dummy)
00316 : "0" (ns),
00317 "g" (l4_scaler_ns_to_tsc)
00318);
00319 return tsc;
00320 }
00321
00322 L4_INLINE void
00323 l4_busy_wait_ns (l4_uint64_t ns)
00324 {
00325 l4_cpu_time_t stop = l4_rdtsc();
00326 stop += l4_ns_to_tsc(ns);
00327
00328 while (l4_rdtsc() < stop)
00329 ;
00330 }
00331
00332 L4_INLINE void
00333 l4_busy_wait_us (l4_uint64_t us)
00334 {
00335 l4_cpu_time_t stop = l4_rdtsc ();
00336 stop += l4_ns_to_tsc(us*1000ULL);
00337
00338 while (l4_rdtsc() < stop)
00339 ;
00340 }
00341
00342 #endif /* __l4_rdtsc_h */
00343

```

## 17.77 amd64/l4/util/spin.h File Reference

Spinning for amd64.

#include <l4/sys/compiler.h>  
 Include dependency graph for spin.h:



### 17.77.1 Detailed Description

Spinning for amd64.

Definition in file [spin.h](#).

## 17.78 spin.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #ifndef __l4util_spin_h
00012 #define __l4util_spin_h
00013
00014 #include <l4/sys/compiler.h>
00015
00016 L4_BEGIN_DECLS
00017
00018 L4_CV void l4_spin(int x,int y);
00019 L4_CV void l4_spin_vga(int x,int y);
00020 L4_CV void l4_spin_n_text(int x, int y, int len, const char*s);
00021 L4_CV void l4_spin_n_text_vga(int x, int y, int len, const char*s);
00022
00023 /*****
00024 *
00025 * spin_text() - spinning wheel at the hercules screen. The given text
00026 * must be a text constant, no variables or arrays. Its
00027 * size is determined with the sizeof operator, it's much
00028 * faster than the strlen function.
00029 * spin_text_vga() - same for vga.
00030 *
00031 *****/
00032 #define l4_spin_text(x, y, text) \
00033 l4_spin_n_text((x), (y), sizeof(text)-1, "" text)
00034 #define l4_spin_text_vga(x, y, text) \
00035 l4_spin_n_text_vga((x), (y), sizeof(text)-1, "" text)
00036
00037 L4_END_DECLS
00038
00039 #endif

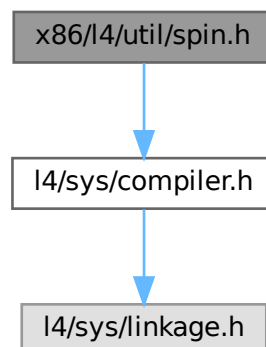
```

## 17.79 x86/l4/util/spin.h File Reference

Spinning for x86.

```
#include <l4/sys/compiler.h>
```

Include dependency graph for spin.h:



## 17.79.1 Detailed Description

Spinning for x86.

Definition in file [spin.h](#).

## 17.80 spin.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Frank Mehnert <fm3@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #ifndef __l4util_spin_h
00012 #define __l4util_spin_h
00013
00014 #include <l4/sys/compiler.h>
00015
00016 L4_BEGIN_DECLS
00017
00018 L4_CV void l4_spin(int x,int y);
00019 L4_CV void l4_spin_vga(int x,int y);
00020 L4_CV void l4_spin_n_text(int x, int y, int len, const char*s);
00021 L4_CV void l4_spin_n_text_vga(int x, int y, int len, const char*s);
00022
00023 /*****
00024 *
00025 * spin_text() - spinning wheel at the hercules screen. The given text
00026 * must be a text constant, no variables or arrays. Its
00027 * size is determined with the sizeof operator, it's much
00028 * faster than the strlen function.
00029 * spin_text_vga() - same for vga.
00030 *
00031 *****/
00032 #define l4_spin_text(x, y, text) \
00033 l4_spin_n_text((x), (y), sizeof(text)-1, "" text)
00034 #define l4_spin_text_vga(x, y, text) \
00035 l4_spin_n_text_vga((x), (y), sizeof(text)-1, "" text)
00036
00037 L4_END_DECLS
00038
00039 #endif

```

## 17.81 amd64/l4/sys/segment.h File Reference

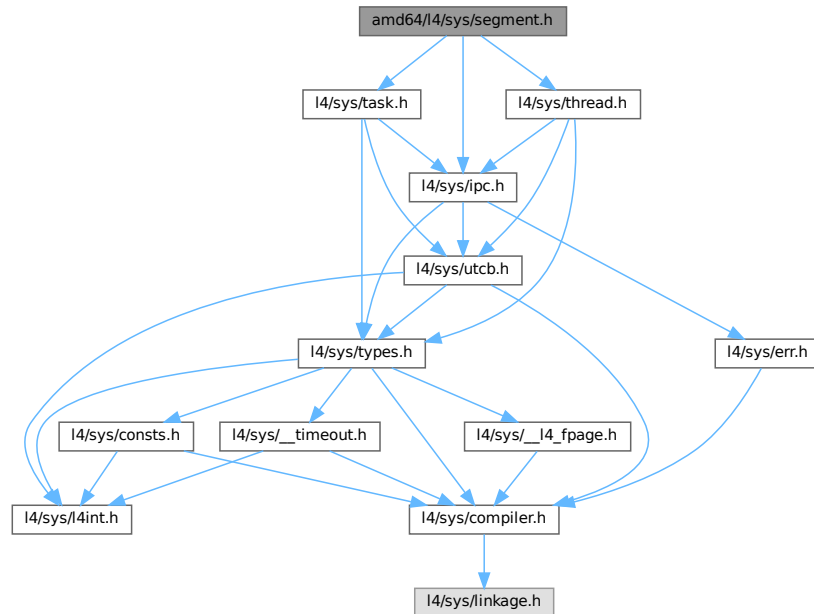
Segment handling (AMD64).

```

#include <l4/sys/ipc.h>
#include <l4/sys/task.h>
#include <l4/sys/thread.h>

```

Include dependency graph for segment.h:



## Enumerations

- enum [L4\\_task\\_ldt\\_x86\\_consts](#) { [L4\\_TASK\\_LDT\\_X86\\_ENTRY\\_SIZE](#) = 8 , [L4\\_TASK\\_LDT\\_X86\\_MAX\\_ENTRIES](#) }
- Constants for LDT handling.*
- enum [L4\\_sys\\_segment](#) { [L4\\_AMD64\\_SEGMENT\\_FS](#) = 0 , [L4\\_AMD64\\_SEGMENT\\_GS](#) = 1 }
- Constants for identifying segments.*

## Functions

- long [fiasco\\_ldt\\_set](#) ([l4\\_cap\\_idx\\_t](#) task, void \*ldt, unsigned int num\_desc, unsigned int entry\_number\_start, [l4\\_utcb\\_t](#) \*utcb)  
*Set LDT segments descriptors.*
- long [fiasco\\_gdt\\_set](#) ([l4\\_cap\\_idx\\_t](#) thread, void \*desc, unsigned int size, unsigned int entry\_number\_start, [l4\\_utcb\\_t](#) \*utcb)  
*Set GDT segment descriptors.*
- unsigned [fiasco\\_gdt\\_get\\_entry\\_offset](#) ([l4\\_cap\\_idx\\_t](#) thread, [l4\\_utcb\\_t](#) \*utcb)  
*Return the offset of the entry in the GDT.*
- long [fiasco\\_amd64\\_set\\_fs](#) ([l4\\_cap\\_idx\\_t](#) thread, [l4\\_umword\\_t](#) base, [l4\\_utcb\\_t](#) \*utcb)  
*Set the base address for the FS segment.*
- long [fiasco\\_amd64\\_set\\_segment\\_base](#) ([l4\\_cap\\_idx\\_t](#) thread, enum [L4\\_sys\\_segment](#) segr, [l4\\_umword\\_t](#) base, [l4\\_utcb\\_t](#) \*utcb)  
*Set the base address for a segment.*
- long [fiasco\\_amd64\\_segment\\_info](#) ([l4\\_cap\\_idx\\_t](#) thread, unsigned \*user\_ds, unsigned \*user\_cs, unsigned \*user32\_cs, [l4\\_utcb\\_t](#) \*utcb)  
*Get segment information.*



## 17.81.1 Detailed Description

Segment handling (AMD64).

Definition in file [segment.h](#).

## 17.81.2 Enumeration Type Documentation

### 17.81.2.1 L4\_sys\_segment

```
enum L4_sys_segment
```

Constants for identifying segments.

#### Enumerator

|                     |                                      |
|---------------------|--------------------------------------|
| L4_AMD64_SEGMENT_FS | Constant identifying the FS segment. |
| L4_AMD64_SEGMENT_GS | Constant identifying the GS segment. |

Definition at line 106 of file [segment.h](#).

### 17.81.2.2 L4\_task\_ldt\_x86\_consts

```
enum L4_task_ldt_x86_consts
```

Constants for LDT handling.

#### Enumerator

|                             |                                                                  |
|-----------------------------|------------------------------------------------------------------|
| L4_TASK_LDT_X86_ENTRY_SIZE  | Size of an LDT entry.                                            |
| L4_TASK_LDT_X86_MAX_ENTRIES | Maximum number of LDT entries that can be written with one call. |

Definition at line 75 of file [segment.h](#).

## 17.81.3 Function Documentation

### 17.81.3.1 fiasco\_amd64\_segment\_info()

```
long fiasco_amd64_segment_info (
 l4_cap_idx_t thread,
 unsigned * user_ds,
 unsigned * user_cs,
 unsigned * user32_cs,
 l4_utcb_t * utcb) [inline]
```

Get segment information.

#### Parameters

---

|     |                  |                                                                                                            |
|-----|------------------|------------------------------------------------------------------------------------------------------------|
| in  | <i>thread</i>    | Thread to get info from.                                                                                   |
| out | <i>user_ds</i>   | DS segment selector.                                                                                       |
| out | <i>user_cs</i>   | 64-bit CS segment selector.                                                                                |
| out | <i>user32_cs</i> | 32-bit CS segment selector.                                                                                |
|     | <i>utcb</i>      | UTCB to be used for this operation, shall be the UTCB of the calling thread. See <a href="#">l4_utcb</a> . |

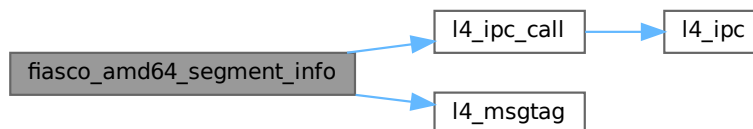
## Returns

System call error

Definition at line 175 of file [segment.h](#).

References [l4\\_ipc\\_call\(\)](#), [L4\\_IPC\\_NEVER](#), [l4\\_msgtag\(\)](#), [L4\\_PROTO\\_THREAD](#), [L4\\_THREAD\\_AMD64\\_GET\\_SEGMENT\\_INFO\\_OP](#), and [l4\\_msg\\_regs\\_t::mr](#).

Here is the call graph for this function:



### 17.81.3.2 fiasco\_amd64\_set\_fs()

```

long fiasco_amd64_set_fs (
 l4_cap_idx_t thread,
 l4_umword_t base,
 l4_utcb_t * utcb) [inline]

```

Set the base address for the FS segment.

## Parameters

|               |                                                                                                            |
|---------------|------------------------------------------------------------------------------------------------------------|
| <i>thread</i> | Thread for which the FS base address shall be modified.                                                    |
| <i>base</i>   | Base address.                                                                                              |
| <i>utcb</i>   | UTCB to be used for this operation, shall be the UTCB of the calling thread. See <a href="#">l4_utcb</a> . |

## Return values

|               |          |
|---------------|----------|
| <i>L4_EOK</i> | Success. |
|---------------|----------|

|                         |                                                            |
|-------------------------|------------------------------------------------------------|
| <code>-L4_EINVAL</code> | Invalid base address ( <i>base</i> ).                      |
| <code>-L4_ENOSYS</code> | Operation not supported with current kernel configuration. |

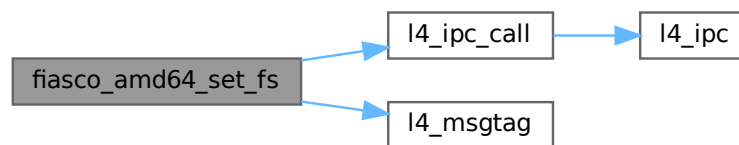
**Note**

Calling this function is equivalent to calling `fiasco_amd64_set_segment_base(thread, L4_↵AMD64_SEGMENT_FS, base, utcb)`.

Definition at line 24 of file [segment.h](#).

References [L4\\_AMD64\\_SEGMENT\\_FS](#), [l4\\_ipc\\_call\(\)](#), [L4\\_IPC\\_NEVER](#), [l4\\_msgtag\(\)](#), [L4\\_PROTO\\_THREAD](#), and [L4\\_THREAD\\_AMD64\\_SET\\_SEGMENT\\_BASE\\_OP](#).

Here is the call graph for this function:

**17.81.3.3 fiasco\_amd64\_set\_segment\_base()**

```

long fiasco_amd64_set_segment_base (
 l4_cap_idx_t thread,
 enum L4_sys_segment segr,
 l4_umword_t base,
 l4_utcb_t * utcb) [inline]

```

Set the base address for a segment.

**Parameters**

|               |                                                                                                            |
|---------------|------------------------------------------------------------------------------------------------------------|
| <i>thread</i> | Thread for which the base address of the selected segment shall be modified.                               |
| <i>segr</i>   | Segment to modify (one of <a href="#">L4_sys_segment</a> ).                                                |
| <i>base</i>   | Base address.                                                                                              |
| <i>utcb</i>   | UTCB to be used for this operation, shall be the UTCB of the calling thread. See <a href="#">l4_utcb</a> . |

**Return values**

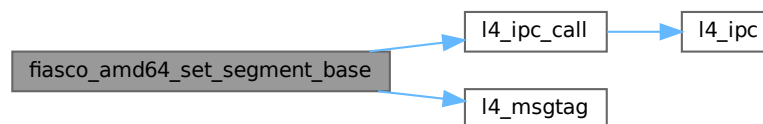
|                     |          |
|---------------------|----------|
| <code>L4_EOK</code> | Success. |
|---------------------|----------|

|                         |                                                                              |
|-------------------------|------------------------------------------------------------------------------|
| <code>-L4_EINVAL</code> | Invalid segment ( <code>segr</code> ) or base address ( <code>base</code> ). |
| <code>-L4_ENOSYS</code> | Operation not supported with current kernel configuration.                   |

Definition at line 32 of file [segment.h](#).

References [l4\\_ipc\\_call\(\)](#), [L4\\_IPC\\_NEVER](#), [l4\\_msgtag\(\)](#), [L4\\_PROTO\\_THREAD](#), and [L4\\_THREAD\\_AMD64\\_SET\\_SEGMENT\\_BASE](#).

Here is the call graph for this function:



## 17.82 segment.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 /*****
00013 #ifndef __L4_SYS_ARCH_X86__SEGMENT_H__
00014 #define __L4_SYS_ARCH_X86__SEGMENT_H__
00015
00016 #ifndef L4API_l4f
00017 #error This header file can only be used with a L4API version!
00018 #endif
00019
00020 #include <l4/sys/ipc.h>
00021
00039 L4_INLINE long
00040 fiasco_ldt_set(l4_cap_idx_t task, void *ldt, unsigned int num_desc,
00041 unsigned int entry_number_start, l4_utcb_t *utcb);
00042
00059 L4_INLINE long
00060 fiasco_gdt_set(l4_cap_idx_t thread, void *desc, unsigned int size,
00061 unsigned int entry_number_start, l4_utcb_t *utcb);
00062
00069 L4_INLINE unsigned
00070 fiasco_gdt_get_entry_offset(l4_cap_idx_t thread, l4_utcb_t *utcb);
00071
00075 enum L4_task_ldt_x86_consts
00076 {
00078 L4_TASK_LDT_X86_ENTRY_SIZE = 8,
00080 L4_TASK_LDT_X86_MAX_ENTRIES
00081 = (L4_UTCB_GENERIC_DATA_SIZE - 2)
00082 / (L4_TASK_LDT_X86_ENTRY_SIZE / (L4_MWORD_BITS / 8)),
00083 };
00084
00100 L4_INLINE long
00101 fiasco_amd64_set_fs(l4_cap_idx_t thread, l4_umword_t base, l4_utcb_t *utcb);
00102
00106 enum L4_sys_segment
00107 {
00109 L4_AMD64_SEGMENT_FS = 0,
00111 L4_AMD64_SEGMENT_GS = 1
00112 };
00113
00127 L4_INLINE long

```

```

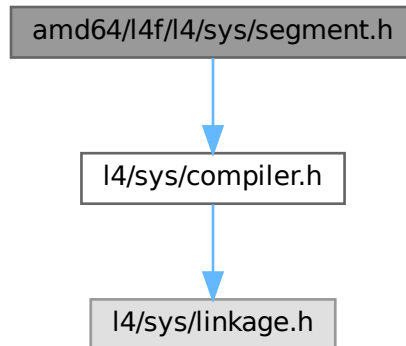
00128 fiasco_amd64_set_segment_base(l4_cap_idx_t thread, enum L4_sys_segment segr,
00129 l4_umword_t base, l4_utcb_t *utcb);
00130
00140 L4_INLINE long
00141 fiasco_amd64_segment_info(l4_cap_idx_t thread, unsigned *user_ds,
00142 unsigned *user_cs, unsigned *user32_cs,
00143 l4_utcb_t *utcb);
00144
00145 /*****
00146 *** Implementation
00147 *****/
00148
00149 #include <l4/sys/task.h>
00150 #include <l4/sys/thread.h>
00151
00152 L4_INLINE long
00153 fiasco_ldt_set(l4_cap_idx_t task, void *ldt, unsigned int num_desc,
00154 unsigned int entry_number_start, l4_utcb_t *utcb)
00155 {
00156 if (num_desc > L4_TASK_LDT_X86_MAX_ENTRIES)
00157 return -L4_EINVAL;
00158 l4_utcb_mr_u(utcb)->mr[0] = L4_TASK_LDT_SET_X86_OP;
00159 l4_utcb_mr_u(utcb)->mr[1] = entry_number_start;
00160 __builtin_memcpy(&l4_utcb_mr_u(utcb)->mr[2], ldt,
00161 num_desc * L4_TASK_LDT_X86_ENTRY_SIZE);
00162 return l4_error_u(l4_ipc_call(task, utcb, l4_msgtag(L4_PROTO_TASK, 2 + num_desc * 2, 0, 0),
00163 L4_IPC_NEVER), utcb);
00164 }
00165
00165 L4_INLINE unsigned
00166 fiasco_gdt_get_entry_offset(l4_cap_idx_t thread, l4_utcb_t *utcb)
00167 {
00168 l4_utcb_mr_u(utcb)->mr[0] = L4_THREAD_X86_GDT_OP;
00169 if (l4_error_u(l4_ipc_call(thread, utcb, l4_msgtag(L4_PROTO_THREAD, 1, 0, 0), L4_IPC_NEVER), utcb))
00170 return -1;
00171 return l4_utcb_mr_u(utcb)->mr[0];
00172 }
00173
00174 L4_INLINE long
00175 fiasco_amd64_segment_info(l4_cap_idx_t thread, unsigned *user_ds,
00176 unsigned *user_cs, unsigned *user32_cs,
00177 l4_utcb_t *utcb)
00178 {
00179 l4_msg_regs_t *m = l4_utcb_mr_u(utcb);
00180 int r;
00181
00182 m->mr[0] = L4_THREAD_AMD64_GET_SEGMENT_INFO_OP;
00183
00184 r = l4_error_u(l4_ipc_call(thread, utcb, l4_msgtag(L4_PROTO_THREAD, 1, 0, 0),
00185 L4_IPC_NEVER), utcb);
00186 if (r < 0)
00187 return r;
00188
00189 *user_ds = m->mr[0];
00190 *user_cs = m->mr[1];
00191 *user32_cs = m->mr[2];
00192
00193 return 0;
00194 }
00195
00196 #endif /* ! __L4_SYS_ARCH_X86_SEGMENT_H */

```

## 17.83 amd64/l4f/l4/sys/segment.h File Reference

l4f-specific fs/gs manipulation (AMD64).

```
#include <l4/sys/compiler.h>
Include dependency graph for segment.h:
```



## Functions

- long [fiasco\\_amd64\\_set\\_fs](#) ([l4\\_cap\\_idx\\_t](#) thread, [l4\\_umword\\_t](#) base, [l4\\_utcb\\_t](#) \*utcb)  
Set the base address for the FS segment.
- long [fiasco\\_amd64\\_set\\_segment\\_base](#) ([l4\\_cap\\_idx\\_t](#) thread, enum [L4\\_sys\\_segment](#) segr, [l4\\_umword\\_t](#) base, [l4\\_utcb\\_t](#) \*utcb)  
Set the base address for a segment.
- long [fiasco\\_gdt\\_set](#) ([l4\\_cap\\_idx\\_t](#) thread, void \*desc, unsigned int size, unsigned int entry\_number\_start, [l4\\_utcb\\_t](#) \*utcb)  
Set GDT segment descriptors.

### 17.83.1 Detailed Description

l4f-specific fs/gs manipulation (AMD64).

Definition in file [segment.h](#).

### 17.83.2 Function Documentation

#### 17.83.2.1 fiasco\_amd64\_set\_fs()

```
long fiasco_amd64_set_fs (
 l4_cap_idx_t thread,
 l4_umword_t base,
 l4_utcb_t * utcb) [inline]
```

Set the base address for the FS segment.

#### Parameters

---

|               |                                                                                                            |
|---------------|------------------------------------------------------------------------------------------------------------|
| <i>thread</i> | Thread for which the FS base address shall be modified.                                                    |
| <i>base</i>   | Base address.                                                                                              |
| <i>utcb</i>   | UTCB to be used for this operation, shall be the UTCB of the calling thread. See <a href="#">l4_utcb</a> . |

### Return values

|                   |                                                            |
|-------------------|------------------------------------------------------------|
| <i>L4_EOK</i>     | Success.                                                   |
| <i>-L4_EINVAL</i> | Invalid base address ( <i>base</i> ).                      |
| <i>-L4_ENOSYS</i> | Operation not supported with current kernel configuration. |

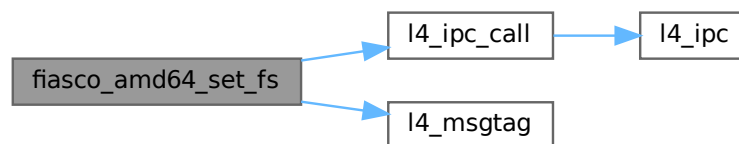
### Note

Calling this function is equivalent to calling `fiasco_amd64_set_segment_base(thread, L4_↔AMD64_SEGMENT_FS, base, utcb)`.

Definition at line 24 of file [segment.h](#).

References [L4\\_AMD64\\_SEGMENT\\_FS](#), [l4\\_ipc\\_call\(\)](#), [L4\\_IPC\\_NEVER](#), [l4\\_msgtag\(\)](#), [L4\\_PROTO\\_THREAD](#), and [L4\\_THREAD\\_AMD64\\_SET\\_SEGMENT\\_BASE\\_OP](#).

Here is the call graph for this function:



### 17.83.2.2 fiasco\_amd64\_set\_segment\_base()

```

long fiasco_amd64_set_segment_base (
 l4_cap_idx_t thread,
 enum L4_sys_segment sgr,
 l4_umword_t base,
 l4_utcb_t * utcb) [inline]

```

Set the base address for a segment.

### Parameters

|               |                                                                              |
|---------------|------------------------------------------------------------------------------|
| <i>thread</i> | Thread for which the base address of the selected segment shall be modified. |
|---------------|------------------------------------------------------------------------------|

|             |                                                                                                            |
|-------------|------------------------------------------------------------------------------------------------------------|
| <i>segr</i> | Segment to modify (one of <a href="#">L4_sys_segment</a> ).                                                |
| <i>base</i> | Base address.                                                                                              |
| <i>utcb</i> | UTCB to be used for this operation, shall be the UTCB of the calling thread. See <a href="#">l4_utcb</a> . |

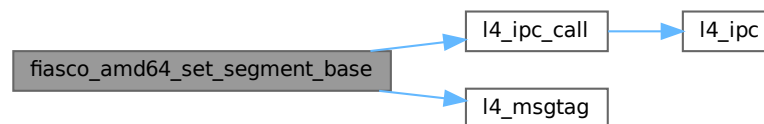
### Return values

|                   |                                                                  |
|-------------------|------------------------------------------------------------------|
| <i>L4_EOK</i>     | Success.                                                         |
| <i>-L4_EINVAL</i> | Invalid segment ( <i>segr</i> ) or base address ( <i>base</i> ). |
| <i>-L4_ENOSYS</i> | Operation not supported with current kernel configuration.       |

Definition at line 32 of file [segment.h](#).

References [l4\\_ipc\\_call\(\)](#), [L4\\_IPC\\_NEVER](#), [l4\\_msgtag\(\)](#), [L4\\_PROTO\\_THREAD](#), and [L4\\_THREAD\\_AMD64\\_SET\\_SEGMENT\\_BASE](#).

Here is the call graph for this function:



## 17.84 segment.h

[Go to the documentation of this file.](#)

```

00001 #include_next <l4/sys/segment.h>
00002
00008 /*
00009 * (c) 2011 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #ifndef __L4_SYS__ARCH_AMD64__L4API_L4F__SEGMENT_H__
00015 #define __L4_SYS__ARCH_AMD64__L4API_L4F__SEGMENT_H__
00016
00017 #include <l4/sys/compiler.h>
00018
00019 /***** Implementation *****/
00020
00021
00022
00023 L4_INLINE long
00024 fiasco_amd64_set_fs(l4_cap_idx_t thread, l4_umword_t base, l4_utcb_t *utcb)
00025 {
00026 l4_utcb_mr_u(utcb)->mr[0] = L4_THREAD_AMD64_SET_SEGMENT_BASE_OP | ((l4_umword_t)L4_AMD64_SEGMENT_FS
00027 << 16);
00027 l4_utcb_mr_u(utcb)->mr[1] = base;
00028 return l4_error_u(l4_ipc_call(thread, utcb, l4_msgtag(L4_PROTO_THREAD, 2, 0, 0), L4_IPC_NEVER),
00029 utcb);
00029 }
00030
00031 L4_INLINE long
00032 fiasco_amd64_set_segment_base(l4_cap_idx_t thread, enum L4_sys_segment segr,
00033 l4_umword_t base, l4_utcb_t *utcb)

```



```

00034 {
00035 l4_utcb_mr_u(utcb)->mr[0] = L4_THREAD_AMD64_SET_SEGMENT_BASE_OP | ((l4_umword_t)segr << 16);
00036 l4_utcb_mr_u(utcb)->mr[1] = base;
00037 return l4_error_u(l4_ipc_call(thread, utcb, l4_msgtag(L4_PROTO_THREAD, 2, 0, 0), L4_IPC_NEVER),
00038 utcb);
00039 }
00040 L4_INLINE long
00041 fiasco_gdt_set(l4_cap_idx_t thread, void *desc, unsigned int size,
00042 unsigned int entry_number_start, l4_utcb_t *utcb)
00043 {
00044 l4_utcb_mr_u(utcb)->mr[0] = L4_THREAD_X86_GDT_OP;
00045 l4_utcb_mr_u(utcb)->mr[1] = entry_number_start;
00046 __builtin_memcpy(&l4_utcb_mr_u(utcb)->mr[2], desc, size);
00047 return l4_error_u(l4_ipc_call(thread, utcb, l4_msgtag(L4_PROTO_THREAD, 2 + (size / 8), 0, 0),
00048 L4_IPC_NEVER), utcb);
00049 }
00050 #endif /* ! __L4_SYS_ARCH_X86__L4API_L4F__SEGMENT_H__ */

```

## 17.85 x86/l4/sys/segment.h File Reference

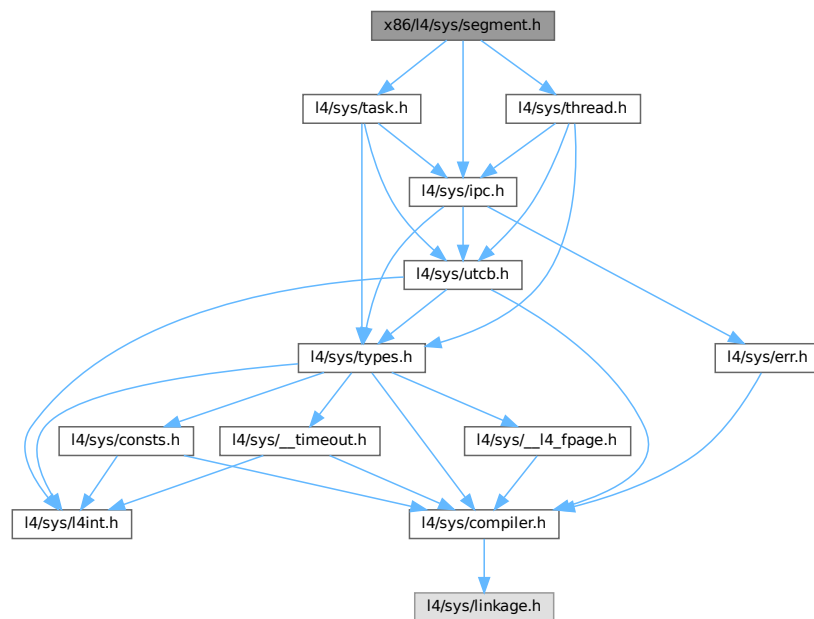
Segment handling (x86).

```

#include <l4/sys/ipc.h>
#include <l4/sys/task.h>
#include <l4/sys/thread.h>

```

Include dependency graph for segment.h:



### Enumerations

- enum [L4\\_task\\_ldt\\_x86\\_consts](#) { [L4\\_TASK\\_LDT\\_X86\\_ENTRY\\_SIZE](#) = 8, [L4\\_TASK\\_LDT\\_X86\\_MAX\\_ENTRIES](#) }

*Contants for LDT handling.*

## Functions

- long `fiasco_ldt_set` (`l4_cap_idx_t` task, void \*ldt, unsigned int num\_desc, unsigned int entry\_number\_start, `l4_utcb_t` \*utcb)  
*Set LDT segments descriptors.*
- long `fiasco_gdt_set` (`l4_cap_idx_t` thread, void \*desc, unsigned int size, unsigned int entry\_number\_start, `l4_utcb_t` \*utcb)  
*Set GDT segment descriptors.*
- unsigned `fiasco_gdt_get_entry_offset` (`l4_cap_idx_t` thread, `l4_utcb_t` \*utcb)  
*Return the offset of the entry in the GDT.*

## 17.85.1 Detailed Description

Segment handling (x86).

Definition in file [segment.h](#).

## 17.85.2 Enumeration Type Documentation

### 17.85.2.1 L4\_task\_ldt\_x86\_consts

enum `L4_task_ldt_x86_consts`

Constants for LDT handling.

#### Enumerator

|                             |                                                                  |
|-----------------------------|------------------------------------------------------------------|
| L4_TASK_LDT_X86_ENTRY_SIZE  | Size of an LDT entry.                                            |
| L4_TASK_LDT_X86_MAX_ENTRIES | Maximum number of LDT entries that can be written with one call. |

Definition at line 75 of file [segment.h](#).

## 17.86 segment.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 /*****
00013 #ifndef __L4_SYS_ARCH_X86__SEGMENT_H__
00014 #define __L4_SYS_ARCH_X86__SEGMENT_H__
00015
00016 #ifndef L4API_L4f
00017 #error This header file can only be used with a L4API version!
00018 #endif
00019
00020 #include <l4/sys/ipc.h>
00021
00039 L4_INLINE long
00040 fiasco_ldt_set(l4_cap_idx_t task, void *ldt, unsigned int num_desc,
00041 unsigned int entry_number_start, l4_utcb_t *utcb);

```

```

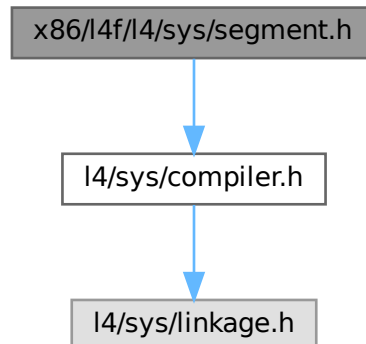
00042
00059 L4_INLINE long
00060 fiasco_gdt_set(l4_cap_idx_t thread, void *desc, unsigned int size,
00061 unsigned int entry_number_start, l4_utcb_t *utcb);
00062
00069 L4_INLINE unsigned
00070 fiasco_gdt_get_entry_offset(l4_cap_idx_t thread, l4_utcb_t *utcb);
00071
00075 enum L4_task_ldt_x86_consts
00076 {
00078 L4_TASK_LDT_X86_ENTRY_SIZE = 8,
00080 L4_TASK_LDT_X86_MAX_ENTRIES
00081 = (L4_UTCB_GENERIC_DATA_SIZE - 2)
00082 / (L4_TASK_LDT_X86_ENTRY_SIZE / (L4_MWORD_BITS / 8)),
00083 };
00084
00085 /*****
00086 *** Implementation
00087 *****/
00088
00089 #include <l4/sys/task.h>
00090 #include <l4/sys/thread.h>
00091
00092 L4_INLINE long
00093 fiasco_ldt_set(l4_cap_idx_t task, void *ldt, unsigned int num_desc,
00094 unsigned int entry_number_start, l4_utcb_t *utcb)
00095 {
00096 if (num_desc > L4_TASK_LDT_X86_MAX_ENTRIES)
00097 return -L4_EINVAL;
00098 l4_utcb_mr_u(utcb)->mr[0] = L4_TASK_LDT_SET_X86_OP;
00099 l4_utcb_mr_u(utcb)->mr[1] = entry_number_start;
00100 __builtin_memcpy(&l4_utcb_mr_u(utcb)->mr[2], ldt,
00101 num_desc * L4_TASK_LDT_X86_ENTRY_SIZE);
00102 return l4_error_u(l4_ipc_call(task, utcb, l4_msgtag(L4_PROTO_TASK, 2 + num_desc * 2, 0, 0),
00103 L4_IPC_NEVER), utcb);
00103 }
00104
00105 L4_INLINE unsigned
00106 fiasco_gdt_get_entry_offset(l4_cap_idx_t thread, l4_utcb_t *utcb)
00107 {
00108 l4_utcb_mr_u(utcb)->mr[0] = L4_THREAD_X86_GDT_OP;
00109 if (l4_error_u(l4_ipc_call(thread, utcb, l4_msgtag(L4_PROTO_THREAD, 1, 0, 0), L4_IPC_NEVER), utcb))
00110 return -1;
00111 return l4_utcb_mr_u(utcb)->mr[0];
00112 }
00113
00114 #endif /* ! __L4_SYS_ARCH_X86_SEGMENT_H__ */

```

## 17.87 x86/i4f/i4/sys/segment.h File Reference

i4f-specific segment manipulation (x86).

```
#include <l4/sys/compiler.h>
Include dependency graph for segment.h:
```



## Functions

- long [fiasco\\_gdt\\_set](#) (l4\_cap\_idx\_t thread, void \*desc, unsigned int size, unsigned int entry\_number\_start, l4\_utcb\_t \*utcb)  
Set GDT segment descriptors.

### 17.87.1 Detailed Description

l4f-specific segment manipulation (x86).

Definition in file [segment.h](#).

## 17.88 segment.h

[Go to the documentation of this file.](#)

```

00001 #include_next <l4/sys/segment.h>
00002
00008 /*
00009 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #ifndef __L4_SYS__ARCH_X86__L4API_L4F__SEGMENT_H__
00015 #define __L4_SYS__ARCH_X86__L4API_L4F__SEGMENT_H__
00016
00017 #include <l4/sys/compiler.h>
00018
00019 /***** Implementation *****/
00020
00021
00022
00023 L4_INLINE long
00024 fiasco_gdt_set(l4_cap_idx_t thread, void *desc, unsigned int size,
00025 unsigned int entry_number_start, l4_utcb_t *utcb)
00026 {
00027 l4_utcb_mr_u(utcb)->mr[0] = L4_THREAD_X86_GDT_OP;
00028 l4_utcb_mr_u(utcb)->mr[1] = entry_number_start;
00029 __builtin_memcpy(&l4_utcb_mr_u(utcb)->mr[2], desc, size);
00030 return l4_error_u(l4_ipc_call(thread, utcb, l4_msgtag(L4_PROTO_THREAD, 2 + (size >> 2), 0, 0),
00031 L4_IPC_NEVER), utcb);
00032 }
00033 #endif /* ! __L4_SYS__ARCH_X86__L4API_L4F__SEGMENT_H__ */

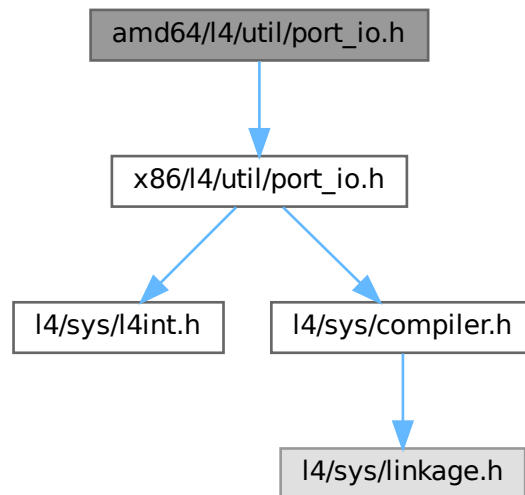
```

## 17.89 amd64/l4/util/port\_io.h File Reference

Port I/O functions.

```
#include <x86/l4/util/port_io.h>
```

Include dependency graph for port\_io.h:



### 17.89.1 Detailed Description

Port I/O functions.

Definition in file [port\\_io.h](#).

## 17.90 port\_io.h

[Go to the documentation of this file.](#)

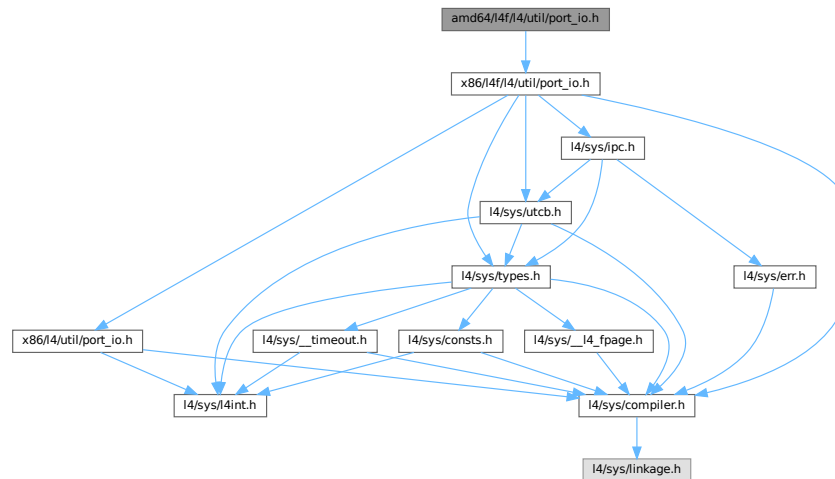
```
00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010 #include <x86/l4/util/port_io.h>
```

## 17.91 amd64/l4f/l4/util/port\_io.h File Reference

Port I/O functions.

```
#include <x86/l4f/l4/util/port_io.h>
```

Include dependency graph for port\_io.h:



### 17.91.1 Detailed Description

Port I/O functions.

Definition in file [port\\_io.h](#).

## 17.92 port\_io.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010 #include <x86/l4f/l4/util/port_io.h>

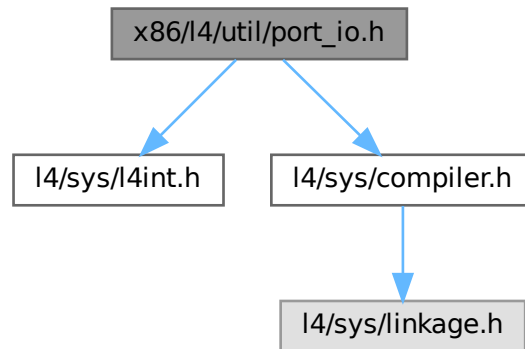
```

## 17.93 x86/l4/util/port\_io.h File Reference

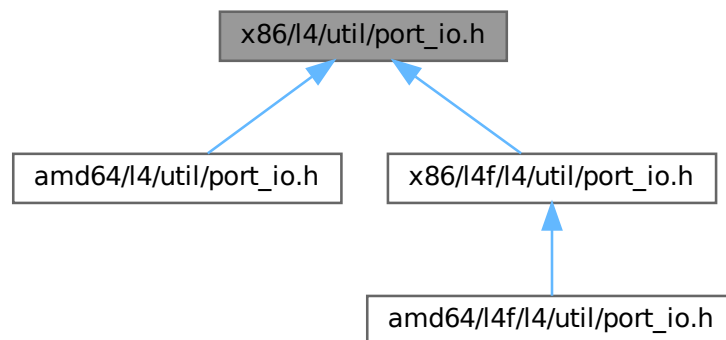
x86 port I/O

```
#include <l4/sys/l4int.h>
#include <l4/sys/compiler.h>
```

Include dependency graph for port\_io.h:



This graph shows which files directly or indirectly include this file:



## Functions

- `l4_uint8_t l4util_in8 (l4_uint16_t port)`  
*Read byte from I/O port.*
- `l4_uint16_t l4util_in16 (l4_uint16_t port)`  
*Read 16-bit-value from I/O port.*
- `l4_uint32_t l4util_in32 (l4_uint16_t port)`  
*Read 32-bit-value from I/O port.*
- `void l4util_ins8 (l4_uint16_t port, l4_umword_t addr, l4_umword_t count)`  
*Read a block of 8-bit-values from I/O ports.*

- void `l4util_ins16` (`l4_uint16_t` port, `l4_umword_t` addr, `l4_umword_t` count)  
*Read a block of 16-bit-values from I/O ports.*
- void `l4util_ins32` (`l4_uint16_t` port, `l4_umword_t` addr, `l4_umword_t` count)  
*Read a block of 32-bit-values from I/O ports.*
- void `l4util_out8` (`l4_uint8_t` value, `l4_uint16_t` port)  
*Write byte to I/O port.*
- void `l4util_out16` (`l4_uint16_t` value, `l4_uint16_t` port)  
*Write 16-bit-value to I/O port.*
- void `l4util_out32` (`l4_uint32_t` value, `l4_uint16_t` port)  
*Write 32-bit-value to I/O port.*
- void `l4util_outs8` (`l4_uint16_t` port, `l4_umword_t` addr, `l4_umword_t` count)  
*Write a block of bytes to I/O port.*
- void `l4util_outs16` (`l4_uint16_t` port, `l4_umword_t` addr, `l4_umword_t` count)  
*Write a block of 16-bit-values to I/O port.*
- void `l4util_outs32` (`l4_uint16_t` port, `l4_umword_t` addr, `l4_umword_t` count)  
*Write block of 32-bit-values to I/O port.*
- void `l4util_iodelay` (void)  
*delay I/O port access by writing to port 0x80*

### 17.93.1 Detailed Description

x86 port I/O

Date

06/2003

Author

Frank Mehnert [fm3@os.inf.tu-dresden.de](mailto:fm3@os.inf.tu-dresden.de)

Definition in file [port\\_io.h](#).

## 17.94 port\_io.h

[Go to the documentation of this file.](#)

```

00001 /*****
00009 /*****
00010
00011 */
00012 * (c) 2003-2009 Author(s)
00013 * economic rights: Technische Universität Dresden (Germany)
00014 * License: see LICENSE.spdx (in this directory or the directories above)
00015 */
00016
00017 #ifndef _L4UTIL_PORT_IO_H
00018 #define _L4UTIL_PORT_IO_H
00019
00024
00025 /* L4 includes */
00026 #include <l4/sys/l4int.h>
00027 #include <l4/sys/compiler.h>
00028
00029 /*****
00030 *** Prototypes
00031 *****/

```



```

00032
00033 L4_BEGIN_DECLS
00044 L4_INLINE l4_uint8_t
00045 l4util_in8(l4_uint16_t port);
00046
00053 L4_INLINE l4_uint16_t
00054 l4util_in16(l4_uint16_t port);
00055
00062 L4_INLINE l4_uint32_t
00063 l4util_in32(l4_uint16_t port);
00064
00072 L4_INLINE void
00073 l4util_ins8(l4_uint16_t port, l4_umword_t addr, l4_umword_t count);
00074
00082 L4_INLINE void
00083 l4util_ins16(l4_uint16_t port, l4_umword_t addr, l4_umword_t count);
00084
00092 L4_INLINE void
00093 l4util_ins32(l4_uint16_t port, l4_umword_t addr, l4_umword_t count);
00094
00101 L4_INLINE void
00102 l4util_out8(l4_uint8_t value, l4_uint16_t port);
00103
00110 L4_INLINE void
00111 l4util_out16(l4_uint16_t value, l4_uint16_t port);
00112
00119 L4_INLINE void
00120 l4util_out32(l4_uint32_t value, l4_uint16_t port);
00121
00129 L4_INLINE void
00130 l4util_outs8(l4_uint16_t port, l4_umword_t addr, l4_umword_t count);
00131
00139 L4_INLINE void
00140 l4util_outs16(l4_uint16_t port, l4_umword_t addr, l4_umword_t count);
00141
00149 L4_INLINE void
00150 l4util_outs32(l4_uint16_t port, l4_umword_t addr, l4_umword_t count);
00151
00155 L4_INLINE void
00156 l4util_iodelay(void);
00157
00159
00160 L4_END_DECLS
00161
00162
00163 /*****
00164 *** Implementation
00165 *****/
00166
00167 L4_INLINE l4_uint8_t
00168 l4util_in8(l4_uint16_t port)
00169 {
00170 l4_uint8_t value;
00171 asm volatile ("inb %w1, %b0" : "=a" (value) : "Nd" (port));
00172 return value;
00173 }
00174
00175 L4_INLINE l4_uint16_t
00176 l4util_in16(l4_uint16_t port)
00177 {
00178 l4_uint16_t value;
00179 asm volatile ("inw %w1, %w0" : "=a" (value) : "Nd" (port));
00180 return value;
00181 }
00182
00183 L4_INLINE l4_uint32_t
00184 l4util_in32(l4_uint16_t port)
00185 {
00186 l4_uint32_t value;
00187 asm volatile ("inl %w1, %0" : "=a" (value) : "Nd" (port));
00188 return value;
00189 }
00190
00191 L4_INLINE void
00192 l4util_ins8(l4_uint16_t port, l4_umword_t addr, l4_umword_t count)
00193 {
00194 l4_umword_t dummy1, dummy2;
00195 asm volatile ("rep insb" : "=D" (dummy1), "=c" (dummy2)
00196 : "d" (port), "D" (addr), "c" (count)
00197 : "memory");
00198 }
00199
00200 L4_INLINE void
00201 l4util_ins16(l4_uint16_t port, l4_umword_t addr, l4_umword_t count)
00202 {
00203 l4_umword_t dummy1, dummy2;
00204 asm volatile ("rep insw" : "=D" (dummy1), "=c" (dummy2)

```

```

00205 : "d" (port), "D" (addr), "c"(count)
00206 : "memory");
00207 }
00208
00209 L4_INLINE void
00210 l4util_ins32(l4_uint16_t port, l4_umword_t addr, l4_umword_t count)
00211 {
00212 l4_umword_t dummy1, dummy2;
00213 asm volatile ("rep insl" : "=D"(dummy1), "=c"(dummy2)
00214 : "d" (port), "D" (addr), "c"(count)
00215 : "memory");
00216 }
00217
00218 L4_INLINE void
00219 l4util_out8(l4_uint8_t value, l4_uint16_t port)
00220 {
00221 asm volatile ("outb %b0, %w1" : : "a" (value), "Nd" (port));
00222 }
00223
00224 L4_INLINE void
00225 l4util_out16(l4_uint16_t value, l4_uint16_t port)
00226 {
00227 asm volatile ("outw %w0, %w1" : : "a" (value), "Nd" (port));
00228 }
00229
00230 L4_INLINE void
00231 l4util_out32(l4_uint32_t value, l4_uint16_t port)
00232 {
00233 asm volatile ("outl %0, %w1" : : "a" (value), "Nd" (port));
00234 }
00235
00236 L4_INLINE void
00237 l4util_outs8(l4_uint16_t port, l4_umword_t addr, l4_umword_t count)
00238 {
00239 l4_umword_t dummy1, dummy2;
00240 asm volatile ("rep outsb" : "=S"(dummy1), "=c"(dummy2)
00241 : "d" (port), "S" (addr), "c"(count)
00242 : "memory");
00243 }
00244
00245 L4_INLINE void
00246 l4util_outs16(l4_uint16_t port, l4_umword_t addr, l4_umword_t count)
00247 {
00248 l4_umword_t dummy1, dummy2;
00249 asm volatile ("rep outsw" : "=S"(dummy1), "=c"(dummy2)
00250 : "d" (port), "S" (addr), "c"(count)
00251 : "memory");
00252 }
00253
00254 L4_INLINE void
00255 l4util_outs32(l4_uint16_t port, l4_umword_t addr, l4_umword_t count)
00256 {
00257 l4_umword_t dummy1, dummy2;
00258 asm volatile ("rep outsl" : "=S"(dummy1), "=c"(dummy2)
00259 : "d" (port), "S" (addr), "c"(count)
00260 : "memory");
00261 }
00262
00263 L4_INLINE void
00264 l4util_iodelay(void)
00265 {
00266 asm volatile ("outb %al, $0x80");
00267 }
00268
00269 #endif

```

## 17.95 x86/I4f/I4/util/port\_io.h File Reference

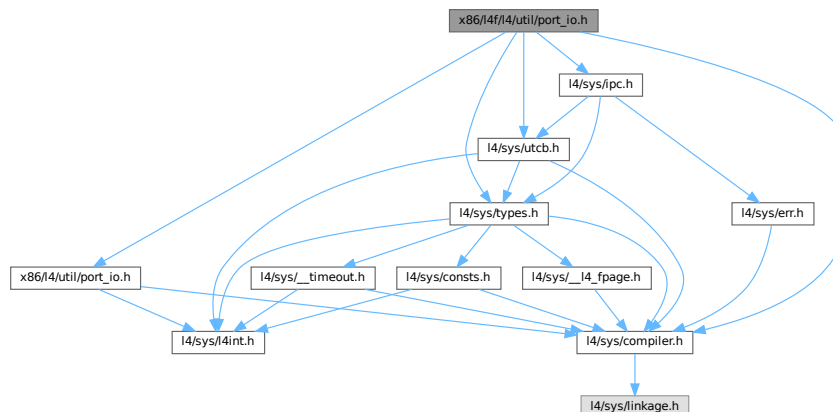
port I/O functions

```

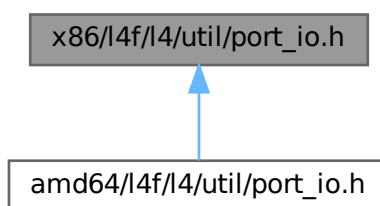
#include <l4/sys/compiler.h>
#include <l4/sys/types.h>
#include <x86/l4/util/port_io.h>
#include <l4/sys/utcb.h>

```

```
#include <l4/sys/ipc.h>
Include dependency graph for port_io.h:
```



This graph shows which files directly or indirectly include this file:



## Functions

- [L4\\_BEGIN\\_DECLS](#) `int l4util_ioport_map (l4_cap_idx_t sigma0id, unsigned port_start, unsigned log2size)`  
Map a range of I/O ports.

## 17.95.1 Detailed Description

port I/O functions

Date

06/2003

Author

Frank Mehnert [fm3@os.inf.tu-dresden.de](mailto:fm3@os.inf.tu-dresden.de)

Definition in file [port\\_io.h](#).

## 17.96 port\_io.h

[Go to the documentation of this file.](#)

```

00001 /*****
00009 /*****
00010
00011 */
00012 * (c) 2003-2009 Author(s)
00013 * economic rights: Technische Universität Dresden (Germany)
00014 * License: see LICENSE.spdx (in this directory or the directories above)
00015 */
00016
00017 #ifndef _L4UTIL_PORT_IO_API_H
00018 #define _L4UTIL_PORT_IO_API_H
00019
00020 #include <l4/sys/compiler.h>
00021 #include <l4/sys/types.h>
00022
00023 #include <x86/l4/util/port_io.h>
00024
00025 L4_BEGIN_DECLS
00026
00027 L4_INLINE int
00028 l4util_ioport_map(l4_cap_idx_t sigma0id,
00029 unsigned port_start, unsigned log2size);
00030
00031 L4_END_DECLS
00032
00033 /*****
00034 *** Implementation
00035 *****/
00036
00037 #include <l4/sys/utcb.h>
00038 #include <l4/sys/ipc.h>
00039
00040 L4_INLINE int
00041 l4util_ioport_map(l4_cap_idx_t sigma0id,
00042 unsigned port_start, unsigned log2size)
00043 {
00044 l4_fpage_t iofp;
00045 l4_msgtag_t tag;
00046 long err;
00047
00048 iofp = l4_iofpage(port_start, log2size);
00049 l4_utcb_mr()->mr[0] = iofp.raw;
00050 l4_utcb_br()->bdr = 0;
00051 l4_utcb_br()->br[0] = L4_ITEM_MAP;
00052 l4_utcb_br()->br[1] = iofp.raw;
00053 tag = l4_ipc_call(sigma0id, l4_utcb(),
00054 l4_msgtag(L4_PROTO_IO_PAGE_FAULT, 1, 0, 0),
00055 L4_IPC_NEVER);
00056
00057 if ((err = l4_ipc_error(tag, l4_utcb())))
00058 return err;
00059
00060 return l4_msgtag_items(tag) > 0 ? 0 : -L4_ENOENT;
00061 }
00062
00063 #endif
00064

```

## 17.97 \_\_kip-arch.h

```

00001 /*
00002 * (c) 2013 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * economic rights: Technische Universität Dresden (Germany)
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 struct l4_kip_platform_info_arch
00010 {};

```

## 17.98 \_\_kip-arch.h

```

00001 /*
00002 * (c) 2013 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * economic rights: Technische Universität Dresden (Germany)
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00014 struct l4_kip_platform_info_arch
00015 {
00016 struct
00017 {
00018 l4_uint32_t MIDR, CTR, TCMTR, TLBTR, MPIDR, REVIDR;
00019 l4_uint32_t ID_PFR[2], ID_DFR0, ID_AFR0, ID_MMFR[4], ID_ISAR[6];
00020 } cpuinfo;
00021 };

```

## 17.99 \_\_kip-arch.h

```

00001 /*
00002 * (c) 2013 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * economic rights: Technische Universität Dresden (Germany)
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00014 struct l4_kip_platform_info_arch
00015 {
00016 struct
00017 {
00018 l4_uint64_t MIDR, MPIDR, REVIDR;
00019 l4_uint64_t ID_PFR[3], ID_DFR0, ID_AFR0, ID_MMFR[4], ID_ISAR[7], ID_MVFR[3];
00020 l4_uint64_t ID_AA64DFR[2], ID_AA64ISAR[3], ID_AA64MMFR[3], ID_AA64PFR[2];
00021 } cpuinfo;
00022 };

```

## 17.100 \_\_kip-arch.h

```

00001 /*
00002 * (c) 2013 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * economic rights: Technische Universität Dresden (Germany)
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 struct l4_kip_platform_info_arch
00010 {};

```

## 17.101 amd64/l4/sys/\_\_vcpu-arch.h File Reference

AMD64-specific vCPU interface.



### 17.101.1 Detailed Description

AMD64-specific vCPU interface.

Definition in file [\\_\\_vcpu-arch.h](#).

### 17.101.2 Enumeration Type Documentation

#### 17.101.2.1 anonymous enum

anonymous enum

#### Enumerator

|                       |                                                                                                                                                                                      |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| L4_VCPU_STATE_VERSION | Architecture-specific version ID. This ID must match the version field in the <a href="#">l4_vcpu_state_t</a> structure after enabling vCPU mode or extended vCPU mode for a thread. |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Definition at line 16 of file [\\_\\_vcpu-arch.h](#).

## 17.102 \_\_vcpu-arch.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00012 #pragma once
00013
00014 #include <l4/sys/types.h>
00015
00016 enum
00017 {
00024 L4_VCPU_STATE_VERSION = 0x26,
00025
00026 L4_VCPU_STATE_SIZE = 0x200,
00027 L4_VCPU_STATE_EXT_SIZE = L4_PAGESIZE,
00028 };
00029
00034 enum L4_vcpu_state_offset
00035 {
00036 L4_VCPU_OFFSET_EXT_STATE = 0x400,
00037 L4_VCPU_OFFSET_EXT_INFOS = 0x200,
00038 };
00039
00043 typedef struct l4_vcpu_arch_state_t
00044 {
00045 l4_umword_t host_fs_base;
00046 l4_umword_t host_gs_base;
00047 l4_uint16_t host_ds, host_es, host_fs, host_gs;
00048
00049 l4_uint16_t const user_ds32;
00050 l4_uint16_t const user_cs64;
00051 l4_uint16_t const user_cs32;
00052 } l4_vcpu_arch_state_t;
00053
00054
00059 typedef struct l4_vcpu_regs_t
00060 {
00061 l4_umword_t r15;
00062 l4_umword_t r14;
00063 l4_umword_t r13;

```

```

00064 l4_umword_t r12;
00065 l4_umword_t r11;
00066 l4_umword_t r10;
00067 l4_umword_t r9;
00068 l4_umword_t r8;
00069
00070 l4_umword_t di;
00071 l4_umword_t si;
00072 l4_umword_t bp;
00073 l4_umword_t pfa;
00074 l4_umword_t bx;
00075 l4_umword_t dx;
00076 l4_umword_t cx;
00077 l4_umword_t ax;
00078
00079 l4_umword_t trapno;
00080 l4_umword_t err;
00081
00082 l4_umword_t ip;
00083 l4_umword_t cs;
00084 l4_umword_t flags;
00085 l4_umword_t sp;
00086 l4_umword_t ss;
00087 l4_umword_t fs_base;
00088 l4_umword_t gs_base;
00089 l4_uint16_t ds, es, fs, gs;
00090
00091 } l4_vcpu_regs_t;
00092
00097 typedef struct l4_vcpu_ipc_regs_t
00098 {
00099 l4_umword_t _res[1];
00100 l4_umword_t label;
00101 l4_umword_t _res2[5];
00102 l4_msgtag_t tag;
00103 } l4_vcpu_ipc_regs_t;

```

### 17.103 arm/l4/sys/\_\_vcpu-arch.h File Reference

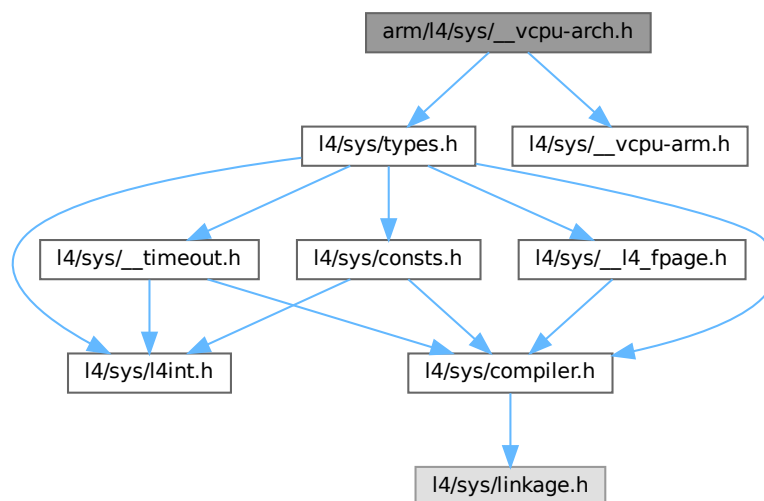
ARM-specific vCPU interface.

```

#include <l4/sys/types.h>
#include <l4/sys/__vcpu-arm.h>

```

Include dependency graph for \_\_vcpu-arch.h:





## Data Structures

- struct [l4\\_vcpu\\_regs\\_t](#)  
*vCPU registers.*
- struct [l4\\_vcpu\\_arch\\_state\\_t](#)  
*Architecture-specific vCPU state.*
- struct [l4\\_vcpu\\_ipc\\_regs\\_t](#)  
*vCPU message registers.*

## Typedefs

- typedef struct [l4\\_vcpu\\_regs\\_t](#) **[l4\\_vcpu\\_regs\\_t](#)**  
*vCPU registers.*
- typedef struct [l4\\_vcpu\\_arch\\_state\\_t](#) **[l4\\_vcpu\\_arch\\_state\\_t](#)**  
*Architecture-specific vCPU state.*
- typedef struct [l4\\_vcpu\\_ipc\\_regs\\_t](#) **[l4\\_vcpu\\_ipc\\_regs\\_t](#)**  
*vCPU message registers.*

## Enumerations

- enum { [L4\\_VCPU\\_STATE\\_VERSION](#) = 0x38 , [L4\\_VCPU\\_STATE\\_SIZE](#) = 0x100 , [L4\\_VCPU\\_STATE\\_EXT\\_SIZE](#) = 0x400 }
  - enum [L4\\_vcpu\\_state\\_offset](#) { [L4\\_VCPU\\_OFFSET\\_EXT\\_STATE](#) = 0x180 , [L4\\_VCPU\\_OFFSET\\_EXT\\_INFOS](#) = 0x100 }
  - enum [L4\\_vcpu\\_e\\_field\\_ids](#) { }
- Offsets for vCPU state layouts.*
- IDs for extended vCPU state fields.*

### 17.103.1 Detailed Description

ARM-specific vCPU interface.

Definition in file [\\_\\_vcpu-arch.h](#).

### 17.103.2 Enumeration Type Documentation

#### 17.103.2.1 anonymous enum

anonymous enum

#### Enumerator

|                                       |                                                                                                                                                                                      |
|---------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">L4_VCPU_STATE_VERSION</a> | Architecture-specific version ID. This ID must match the version field in the <a href="#">l4_vcpu_state_t</a> structure after enabling vCPU mode or extended vCPU mode for a thread. |
|---------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Definition at line 17 of file [\\_\\_vcpu-arch.h](#).

### 17.103.2.2 L4\_vcpu\_e\_field\_ids

enum `L4_vcpu_e_field_ids`

IDs for extended vCPU state fields.

Bits 14..15: are the field size:

- 0 = 32bit field
- 1 = register width field
- 2 = 64bit field

#### Enumerator

|                    |                          |
|--------------------|--------------------------|
| L4_VCPU_E_VTMR_CFG | vtimer irq configuration |
|--------------------|--------------------------|

Definition at line 99 of file `__vcpu-arch.h`.

## 17.104 \_\_vcpu-arch.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00012 #pragma once
00013
00014 #include <l4/sys/types.h>
00015 #include <l4/sys/__vcpu-arm.h>
00016
00017 enum
00018 {
00025 L4_VCPU_STATE_VERSION = 0x38,
00026
00027 L4_VCPU_STATE_SIZE = 0x100,
00028 L4_VCPU_STATE_EXT_SIZE = 0x400,
00029 };
00030
00035 enum L4_vcpu_state_offset
00036 {
00037 L4_VCPU_OFFSET_EXT_STATE = 0x180,
00038 L4_VCPU_OFFSET_EXT_INFOS = 0x100,
00039 };
00040
00041 L4_INLINE l4_arm_vcpu_e_info_t const *
00042 l4_vcpu_e_info(void const *vcpu) L4_NOTHROW
00043 {
00044 return (l4_arm_vcpu_e_info_t const *) ((l4_addr_t)vcpu
00045 + L4_VCPU_OFFSET_EXT_INFOS);
00046 }
00047
00048 L4_INLINE void *l4_vcpu_e_ptr(void const *vcpu, unsigned id) L4_NOTHROW
00049 { return (void *) ((l4_addr_t)vcpu + L4_VCPU_OFFSET_EXT_STATE + (id & 0xfff)); }
00050
00055 typedef struct l4_vcpu_regs_t
00056 {
00057 l4_umword_t pfa;
00058 l4_umword_t err;
00059
00060 l4_umword_t r[13];
00061
00062 l4_umword_t sp;
00063 l4_umword_t lr;

```

```

00064 l4_umword_t _dummy;
00065 l4_umword_t ip;
00066 l4_umword_t flags;
00067 l4_umword_t tpidruro;
00068 l4_umword_t tpidrurw;
00069 } l4_vcpu_regs_t;
00070
00074 typedef struct l4_vcpu_arch_state_t
00075 {
00076 l4_umword_t host_tpidruro;
00077 } l4_vcpu_arch_state_t;
00078
00083 typedef struct l4_vcpu_ipc_regs_t
00084 {
00085 l4_msgtag_t tag;
00086 l4_umword_t _d1[3];
00087 l4_umword_t label;
00088 l4_umword_t _d2[8];
00089 } l4_vcpu_ipc_regs_t;
00090
00099 enum L4_vcpu_e_field_ids
00100 {
00101 L4_VCPU_E_HCR = 0x8008,
00102 L4_VCPU_E_TTBRO = 0x8010,
00103 L4_VCPU_E_TTBRI = 0x8018,
00104 L4_VCPU_E_TTBCE = 0x0020,
00105 L4_VCPU_E_SCTLR = 0x0024,
00106 L4_VCPU_E_DACR = 0x0028,
00107 L4_VCPU_E_FCSEIDR = 0x002c,
00108
00109 L4_VCPU_E_CNTVCTL = 0x0030,
00110 L4_VCPU_E_CNTVOFF = 0x8038,
00111
00112 L4_VCPU_E_VMPIDR = 0x0040,
00113 L4_VCPU_E_VPIDR = 0x0044,
00114
00115 L4_VCPU_E_VTMR_CFG = 0x0048,
00116
00117 L4_VCPU_E_GIC_HCR = 0x0050,
00118 L4_VCPU_E_GIC_VTR = 0x0054,
00119 L4_VCPU_E_GIC_VMCR = 0x0058,
00120 L4_VCPU_E_GIC_MISR = 0x005c,
00121 L4_VCPU_E_GIC_EISR = 0x0060,
00122 L4_VCPU_E_GIC_ELSR = 0x0064,
00123 L4_VCPU_E_GIC_V2_LR0 = 0x0068,
00124 L4_VCPU_E_GIC_V3_LR0 = 0x8068,
00125 };

```

## 17.105 arm64/l4/sys/\_\_vcpu-arch.h File Reference

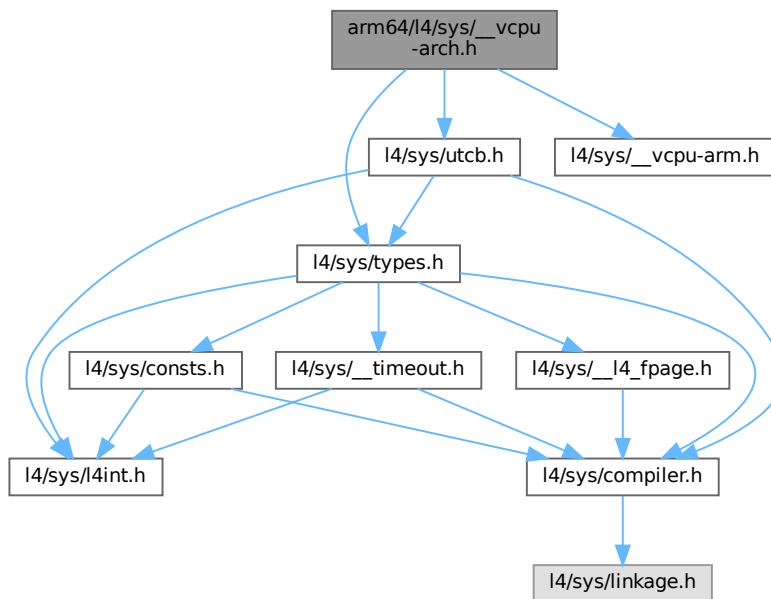
ARM64-specific vCPU interface.

```

#include <l4/sys/types.h>
#include <l4/sys/utcb.h>
#include <l4/sys/__vcpu-arm.h>

```

Include dependency graph for `__vcpu-arch.h`:



## Data Structures

- struct `l4_vcpu_arch_state_t`  
*Architecture-specific vCPU state.*
- struct `l4_vcpu_ipc_regs_t`  
*vCPU message registers.*

## Typedefs

- typedef `l4_exc_regs_t` `l4_vcpu_regs_t`  
*vCPU registers.*
- typedef struct `l4_vcpu_arch_state_t` `l4_vcpu_arch_state_t`  
*Architecture-specific vCPU state.*
- typedef struct `l4_vcpu_ipc_regs_t` `l4_vcpu_ipc_regs_t`  
*vCPU message registers.*

## Enumerations

- enum { `L4_VCPU_STATE_VERSION` = 0x38 , `L4_VCPU_STATE_SIZE` = 0x200 , `L4_VCPU_STATE_EXT_SIZE` = 0x800 }
  - enum `l4_vcpu_state_offset` { `L4_VCPU_OFFSET_EXT_STATE` = 0x280 , `L4_VCPU_OFFSET_EXT_INFOS` = 0x200 }  
*Offsets for vCPU state layouts.*
  - enum `l4_vcpu_e_field_ids` { }
- IDs for extended vCPU state fields.*

## 17.105.1 Detailed Description

ARM64-specific vCPU interface.

Definition in file [\\_\\_vcpu-arch.h](#).

## 17.105.2 Enumeration Type Documentation

### 17.105.2.1 anonymous enum

anonymous enum

#### Enumerator

|                       |                                                                                                                                                                                      |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| L4_VCPU_STATE_VERSION | Architecture-specific version ID. This ID must match the version field in the <a href="#">l4_vcpu_state_t</a> structure after enabling vCPU mode or extended vCPU mode for a thread. |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Definition at line 18 of file [\\_\\_vcpu-arch.h](#).

### 17.105.2.2 L4\_vcpu\_e\_field\_ids

enum [L4\\_vcpu\\_e\\_field\\_ids](#)

IDs for extended vCPU state fields.

Bits 14..15: are the field size:

- 0 = 32bit field
- 1 = register width field
- 2 = 64bit field

#### Enumerator

|                    |                          |
|--------------------|--------------------------|
| L4_VCPU_E_VTMR_CFG | vtimer irq configuration |
|--------------------|--------------------------|

Definition at line 85 of file [\\_\\_vcpu-arch.h](#).

## 17.106 \_\_vcpu-arch.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00012 #pragma once
00013
00014 #include <l4/sys/types.h>
00015 #include <l4/sys/utcb.h>
00016 #include <l4/sys/__vcpu-arm.h>
00017
00018 enum
00019 {
00026 L4_VCPU_STATE_VERSION = 0x38,
00027
00028 L4_VCPU_STATE_SIZE = 0x200,
00029 L4_VCPU_STATE_EXT_SIZE = 0x800,
00030 };
00031
00036 enum L4_vcpu_state_offset
00037 {
00038 L4_VCPU_OFFSET_EXT_STATE = 0x280,
00039 L4_VCPU_OFFSET_EXT_INFOS = 0x200,
00040 };
00041
00042 L4_INLINE l4_arm_vcpu_e_info_t const *
00043 l4_vcpu_e_info(void const *vcpu) L4_NOTHROW
00044 {
00045 return (l4_arm_vcpu_e_info_t const *)((l4_addr_t)vcpu
00046 + L4_VCPU_OFFSET_EXT_INFOS);
00047 }
00048
00049 L4_INLINE void *l4_vcpu_e_ptr(void const *vcpu, unsigned id) L4_NOTHROW
00050 { return (void *)((l4_addr_t)vcpu + L4_VCPU_OFFSET_EXT_STATE + (id & 0xfff)); }
00051
00056 typedef l4_exc_regs_t l4_vcpu_regs_t;
00057
00061 typedef struct l4_vcpu_arch_state_t
00062 {
00063 l4_umword_t host_tpidruro;
00064 } l4_vcpu_arch_state_t;
00065
00070 typedef struct l4_vcpu_ipc_regs_t
00071 {
00072 l4_msgtag_t tag;
00073 l4_umword_t label;
00074 l4_umword_t _d1[3];
00075 } l4_vcpu_ipc_regs_t;
00076
00085 enum L4_vcpu_e_field_ids
00086 {
00087 L4_VCPU_E_HCR = 0x8008,
00088 L4_VCPU_E_SCTLR = 0x0010,
00089 L4_VCPU_E_CPACR = 0x0014,
00090
00091 L4_VCPU_E_CNTVCTL = 0x0018,
00092 L4_VCPU_E_CNTVOFF = 0x8020,
00093
00094 L4_VCPU_E_VMPIDR = 0x8028,
00095 L4_VCPU_E_VPIDR = 0x0030,
00096
00097 L4_VCPU_E_VTMR_CFG = 0x0034,
00098 L4_VCPU_E_VTCR = 0x8038,
00099
00100 L4_VCPU_E_GIC_HCR = 0x0040,
00101 L4_VCPU_E_GIC_VTR = 0x0044,
00102 L4_VCPU_E_GIC_VMCR = 0x0048,
00103 L4_VCPU_E_GIC_MISR = 0x004c,
00104 L4_VCPU_E_GIC_EISR = 0x0050,
00105 L4_VCPU_E_GIC_ELSR = 0x0054,
00106 L4_VCPU_E_GIC_V2_LR0 = 0x0058,
00107 L4_VCPU_E_GIC_V3_LR0 = 0x8058,
00108 };

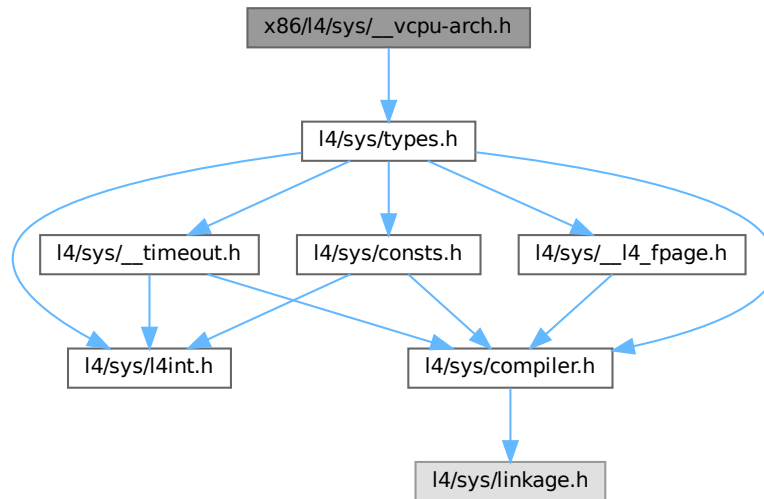
```

## 17.107 x86/l4/sys/\_\_vcpu-arch.h File Reference

x86-specific vCPU interface.

```
#include <l4/sys/types.h>
```

Include dependency graph for \_\_vcpu-arch.h:



### Data Structures

- struct `l4_vcpu_regs_t`  
*vCPU registers.*
- struct `l4_vcpu_arch_state_t`  
*Architecture-specific vCPU state.*
- struct `l4_vcpu_ipc_regs_t`  
*vCPU message registers.*

### Typedefs

- typedef struct `l4_vcpu_regs_t` `l4_vcpu_regs_t`  
*vCPU registers.*
- typedef struct `l4_vcpu_arch_state_t` `l4_vcpu_arch_state_t`  
*Architecture-specific vCPU state.*
- typedef struct `l4_vcpu_ipc_regs_t` `l4_vcpu_ipc_regs_t`  
*vCPU message registers.*

### Enumerations

- enum { `L4_VCPU_STATE_VERSION` = 0x46 , `L4_VCPU_STATE_SIZE` = 0x200 , `L4_VCPU_STATE_EXT_SIZE` = `L4_PAGESIZE` }
  - enum `L4_vcpu_state_offset` { `L4_VCPU_OFFSET_EXT_STATE` = 0x400 , `L4_VCPU_OFFSET_EXT_INFOS` = 0x200 }
- Offsets for vCPU state layouts.*

### 17.107.1 Detailed Description

x86-specific vCPU interface.

Definition in file [\\_\\_vcpu-arch.h](#).

### 17.107.2 Enumeration Type Documentation

#### 17.107.2.1 anonymous enum

anonymous enum

#### Enumerator

|                       |                                                                                                                                                                                      |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| L4_VCPU_STATE_VERSION | Architecture-specific version ID. This ID must match the version field in the <a href="#">l4_vcpu_state_t</a> structure after enabling vCPU mode or extended vCPU mode for a thread. |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Definition at line 16 of file [\\_\\_vcpu-arch.h](#).

## 17.108 \_\_vcpu-arch.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00012 #pragma once
00013
00014 #include <l4/sys/types.h>
00015
00016 enum
00017 {
00024 L4_VCPU_STATE_VERSION = 0x46,
00025
00026 L4_VCPU_STATE_SIZE = 0x200,
00027 L4_VCPU_STATE_EXT_SIZE = L4_PAGESIZE,
00028 };
00029
00034 enum L4_vcpu_state_offset
00035 {
00036 L4_VCPU_OFFSET_EXT_STATE = 0x400,
00037 L4_VCPU_OFFSET_EXT_INFOS = 0x200,
00038 };
00039
00044 typedef struct l4_vcpu_regs_t
00045 {
00046 l4_umword_t es;
00047 l4_umword_t ds;
00048 l4_umword_t gs;
00049 l4_umword_t fs;
00050
00051 l4_umword_t di;
00052 l4_umword_t si;
00053 l4_umword_t bp;
00054 l4_umword_t pfa;
00055 l4_umword_t bx;
00056 l4_umword_t dx;
00057 l4_umword_t cx;
00058 l4_umword_t ax;
00059
00060 l4_umword_t trapno;

```



```

00061 l4_umword_t err;
00062
00063 l4_umword_t ip;
00064 l4_umword_t dummy1;
00065 l4_umword_t flags;
00066 l4_umword_t sp;
00067 l4_umword_t ss;
00068 } l4_vcpu_regs_t;
00069
00073 typedef struct l4_vcpu_arch_state_t { } l4_vcpu_arch_state_t;
00074
00079 typedef struct l4_vcpu_ipc_regs_t
00080 {
00081 l4_umword_t _res[2];
00082 l4_umword_t label;
00083 l4_umword_t _res2[3];
00084 l4_msgtag_t tag;
00085 } l4_vcpu_ipc_regs_t;

```

## 17.109 ktrace\_events.h

```

00001 /* Note, automatically generated from Fiasco binary */
00002 #pragma once
00003
00004 enum L4_ktrace_tbuf_entry_fixed
00005 {
00006 l4_ktrace_tbuf_unused = 0,
00007 l4_ktrace_tbuf_pf = 1,
00008 l4_ktrace_tbuf_ipc = 2,
00009 l4_ktrace_tbuf_ipc_res = 3,
00010 l4_ktrace_tbuf_ipc_trace = 4,
00011 l4_ktrace_tbuf_ke = 5,
00012 l4_ktrace_tbuf_ke_reg = 6,
00013 l4_ktrace_tbuf_breakpoint = 7,
00014 l4_ktrace_tbuf_ke_bin = 8,
00015 l4_ktrace_tbuf_dynentries = 9,
00016 l4_ktrace_tbuf_max = 128,
00017 l4_ktrace_tbuf_hidden = 128,
00018 };
00019
00020 typedef unsigned long L4_ktrace_t__Address;
00021 typedef unsigned long L4_ktrace_t__Cap_index;
00022 typedef void L4_ktrace_t__Context;
00023 typedef void L4_ktrace_t__Context__Drq;
00024 typedef unsigned L4_ktrace_t__Context__Drq_log__Type;
00025 typedef unsigned L4_ktrace_t__Cpu_number;
00026 typedef void L4_ktrace_t__Irq_base;
00027 typedef void L4_ktrace_t__Irq_chip;
00028 typedef void L4_ktrace_t__Kobject;
00029 typedef unsigned long L4_ktrace_t__L4_error;
00030 typedef unsigned long L4_ktrace_t__L4_msg_tag;
00031 typedef unsigned long L4_ktrace_t__L4_obj_ref;
00032 typedef unsigned L4_ktrace_t__L4_timeout_pair;
00033 typedef unsigned long L4_ktrace_t__Mword;
00034 typedef void L4_ktrace_t__Rcu_item;
00035 typedef void L4_ktrace_t__Sched_context;
00036 typedef long L4_ktrace_t__Smword;
00037 typedef void L4_ktrace_t__Space;
00038 typedef unsigned short L4_ktrace_t__Unsigned16;
00039 typedef unsigned int L4_ktrace_t__Unsigned32;
00040 typedef unsigned long long L4_ktrace_t__Unsigned64;
00041 typedef unsigned char L4_ktrace_t__Unsigned8;
00042 typedef void L4_ktrace_t__cxx__Type_info;
00043
00044 typedef struct __attribute__((packed))
00045 {
00046 L4_ktrace_t__Mword_number; /* 0+8 */
00047 L4_ktrace_t__Address_ip; /* 8+8 */
00048 L4_ktrace_t__Unsigned64_tsc; /* 16+8 */
00049 L4_ktrace_t__Context *ctx; /* 24+8 */
00050 L4_ktrace_t__Unsigned32_pmc1; /* 32+4 */
00051 L4_ktrace_t__Unsigned32_pmc2; /* 36+4 */
00052 L4_ktrace_t__Unsigned32_kclock; /* 40+4 */
00053 L4_ktrace_t__Unsigned8_type; /* 44+1 */
00054 L4_ktrace_t__Unsigned8_cpu; /* 45+1 */
00055 union __attribute__((packed))
00056 {
00057 struct __attribute__((packed))
00058 {
00059 char __pre_pad[2];
00060 void *func; /* 48+8 */
00061 L4_ktrace_t__Context *thread; /* 56+8 */
00062 L4_ktrace_t__Context__Drq *rq; /* 64+8 */

```

```

00063 L4_ktrace_t__Cpu_number target_cpu; /* 72+4 */
00064 L4_ktrace_t__Context__Drq_log__Type type; /* 76+4 */
00065 char wait; /* 80+1 */
00066 } drq; /* 88 */
00067 struct __attribute__((__packed__))
00068 {
00069 char __pre_pad[2];
00070 L4_ktrace_t__Mword state; /* 48+8 */
00071 L4_ktrace_t__Mword ip; /* 56+8 */
00072 L4_ktrace_t__Mword sp; /* 64+8 */
00073 L4_ktrace_t__Mword space; /* 72+8 */
00074 L4_ktrace_t__Mword err; /* 80+8 */
00075 unsigned char type; /* 88+1 */
00076 unsigned char trap; /* 89+1 */
00077 } vcpu; /* 96 */
00078 struct __attribute__((__packed__))
00079 {
00080 char __pre_pad[2];
00081 L4_ktrace_t__Sword op; /* 48+8 */
00082 L4_ktrace_t__Cap_index buffer; /* 56+8 */
00083 L4_ktrace_t__Mword id; /* 64+8 */
00084 L4_ktrace_t__Mword ram; /* 72+8 */
00085 L4_ktrace_t__Mword newo; /* 80+8 */
00086 } factory; /* 88 */
00087 struct __attribute__((__packed__))
00088 {
00089 char __pre_pad[2];
00090 L4_ktrace_t__Mword gate_dbg_id; /* 48+8 */
00091 L4_ktrace_t__Mword thread_dbg_id; /* 56+8 */
00092 L4_ktrace_t__Mword label; /* 64+8 */
00093 } gate; /* 72 */
00094 struct __attribute__((__packed__))
00095 {
00096 char __pre_pad[2];
00097 L4_ktrace_t__Irq_base *obj; /* 48+8 */
00098 L4_ktrace_t__Irq_chip *chip; /* 56+8 */
00099 L4_ktrace_t__Mword pin; /* 64+8 */
00100 } irq; /* 72 */
00101 struct __attribute__((__packed__))
00102 {
00103 char __pre_pad[2];
00104 L4_ktrace_t__Kobject *obj; /* 48+8 */
00105 L4_ktrace_t__Mword id; /* 56+8 */
00106 L4_ktrace_t__cxx__Type_info *type; /* 64+8 */
00107 L4_ktrace_t__Mword ram; /* 72+8 */
00108 } destroy; /* 80 */
00109 struct __attribute__((__packed__))
00110 {
00111 char __pre_pad[2];
00112 L4_ktrace_t__Cpu_number cpu; /* 48+4 */
00113 char __pad_l[4];
00114 L4_ktrace_t__Rcu_item *item; /* 56+8 */
00115 void *cb; /* 64+8 */
00116 unsigned char event; /* 72+1 */
00117 } rcu; /* 80 */
00118 struct __attribute__((__packed__))
00119 {
00120 char __pre_pad[2];
00121 L4_ktrace_t__Mword id; /* 48+8 */
00122 L4_ktrace_t__Mword mask; /* 56+8 */
00123 L4_ktrace_t__Mword fpage; /* 64+8 */
00124 char map; /* 72+1 */
00125 } tmap; /* 80 */
00126 struct __attribute__((__packed__))
00127 {
00128 char __pre_pad[2];
00129 L4_ktrace_t__Address _address; /* 48+8 */
00130 int _len; /* 56+4 */
00131 char __pad_l[4];
00132 L4_ktrace_t__Mword _value; /* 64+8 */
00133 int _mode; /* 72+4 */
00134 } bp; /* 80 */
00135 struct __attribute__((__packed__))
00136 {
00137 char __pre_pad[2];
00138 L4_ktrace_t__Context *dst; /* 48+8 */
00139 L4_ktrace_t__Context *dst_orig; /* 56+8 */
00140 L4_ktrace_t__Address kernel_ip; /* 64+8 */
00141 L4_ktrace_t__Mword lock_cnt; /* 72+8 */
00142 L4_ktrace_t__Space *from_space; /* 80+8 */
00143 L4_ktrace_t__Sched_context *from_sched; /* 88+8 */
00144 L4_ktrace_t__Mword from_prio; /* 96+8 */
00145 } context_switch; /* 104 */
00146 struct __attribute__((__packed__))
00147 {
00148 } empty; /* 48 */
00149 struct __attribute__((__packed__))

```

```

00150 {
00151 char __pre_pad[2];
00152 L4_ktrace_t__L4_msg_tag_tag; /* 48+8 */
00153 L4_ktrace_t__Mword_dword[2]; /* 56+16 */
00154 L4_ktrace_t__L4_obj_ref_dst; /* 72+8 */
00155 L4_ktrace_t__Mword_dbg_id; /* 80+8 */
00156 L4_ktrace_t__Mword_label; /* 88+8 */
00157 L4_ktrace_t__L4_timeout_pair_timeout; /* 96+4 */
00158 char __pad_1[4];
00159 L4_ktrace_t__Unsigned64_to_abs_rcv; /* 104+8 */
00160 } ipc; /* 112 */
00161 struct __attribute__((__packed__))
00162 {
00163 L4_ktrace_t__Unsigned8_have_snd; /* 46+1 */
00164 L4_ktrace_t__Unsigned8_is_np; /* 47+1 */
00165 L4_ktrace_t__L4_msg_tag_tag; /* 48+8 */
00166 L4_ktrace_t__Mword_dword[2]; /* 56+16 */
00167 L4_ktrace_t__L4_error_result; /* 72+8 */
00168 L4_ktrace_t__Mword_from; /* 80+8 */
00169 L4_ktrace_t__L4_obj_ref_dst; /* 88+8 */
00170 L4_ktrace_t__Mword_pair_event; /* 96+8 */
00171 } ipc_res; /* 104 */
00172 struct __attribute__((__packed__))
00173 {
00174 char __pre_pad[2];
00175 union __attribute__((__packed__)) {
00176 char msg[80]; /* 0+80 */
00177 struct __attribute__((__packed__)) {
00178 char tag[2]; /* 0+2 */
00179 char __pad_1[6];
00180 char *ptr; /* 8+8 */
00181 } mptr; /* 0+16 */
00182 } msg; /* 48+80 */
00183 } ke; /* 128 */
00184 struct __attribute__((__packed__))
00185 {
00186 char _msg[80]; /* 46+80 */
00187 } ke_bin; /* 128 */
00188 struct __attribute__((__packed__))
00189 {
00190 char __pre_pad[2];
00191 L4_ktrace_t__Mword v[3]; /* 48+24 */
00192 union __attribute__((__packed__)) {
00193 char msg[56]; /* 0+56 */
00194 struct __attribute__((__packed__)) {
00195 char tag[2]; /* 0+2 */
00196 char __pad_1[6];
00197 char *ptr; /* 8+8 */
00198 } mptr; /* 0+16 */
00199 } msg; /* 72+56 */
00200 } ke_reg; /* 128 */
00201 struct __attribute__((__packed__))
00202 {
00203 char __pre_pad[2];
00204 L4_ktrace_t__Address_pfa; /* 48+8 */
00205 L4_ktrace_t__Mword_error; /* 56+8 */
00206 L4_ktrace_t__Space *space; /* 64+8 */
00207 } pf; /* 72 */
00208 struct __attribute__((__packed__))
00209 {
00210 unsigned short mode; /* 46+2 */
00211 L4_ktrace_t__Context *owner; /* 48+8 */
00212 unsigned short id; /* 56+2 */
00213 unsigned short prio; /* 58+2 */
00214 char __pad_1[4];
00215 long left; /* 64+8 */
00216 unsigned long quantum; /* 72+8 */
00217 } sched; /* 80 */
00218 struct __attribute__((__packed__))
00219 {
00220 char _trapno; /* 46+1 */
00221 char __pad_1[1];
00222 L4_ktrace_t__Unsigned16_error; /* 48+2 */
00223 char __pad_2[6];
00224 L4_ktrace_t__Mword_rbp; /* 56+8 */
00225 L4_ktrace_t__Mword_cr2; /* 64+8 */
00226 L4_ktrace_t__Mword_rax; /* 72+8 */
00227 L4_ktrace_t__Mword_rflags; /* 80+8 */
00228 L4_ktrace_t__Mword_rsp; /* 88+8 */
00229 L4_ktrace_t__Unsigned16_cs; /* 96+2 */
00230 L4_ktrace_t__Unsigned16_ds; /* 98+2 */
00231 } trap; /* 104 */
00232 struct __attribute__((__packed__))
00233 {
00234 char _padding[80]; /* 46+80 */
00235 char __post_pad[2]; /* 126+2 */
00236 } fullsize; /* 128 */

```

```

00237 struct __attribute__((__packed__))
00238 {
00239 char __pre_pad[2];
00240 L4_ktrace_t__Cap_index cap_idx; /* 48+8 */
00241 } ieh; /* 56 */
00242 struct __attribute__((__packed__))
00243 {
00244 char __pre_pad[2];
00245 L4_ktrace_t__Mword pfa; /* 48+8 */
00246 L4_ktrace_t__Cap_index cap_idx; /* 56+8 */
00247 L4_ktrace_t__Mword err; /* 64+8 */
00248 } ipfh; /* 72 */
00249 struct __attribute__((__packed__))
00250 {
00251 char __pre_pad[2];
00252 L4_ktrace_t__Mword id; /* 48+8 */
00253 L4_ktrace_t__Mword ip; /* 56+8 */
00254 L4_ktrace_t__Mword sp; /* 64+8 */
00255 L4_ktrace_t__Mword op; /* 72+8 */
00256 } exregs; /* 80 */
00257 struct __attribute__((__packed__))
00258 {
00259 char __pre_pad[2];
00260 L4_ktrace_t__Mword state; /* 48+8 */
00261 L4_ktrace_t__Address user_ip; /* 56+8 */
00262 L4_ktrace_t__Cpu_number src_cpu; /* 64+4 */
00263 L4_ktrace_t__Cpu_number target_cpu; /* 68+4 */
00264 } migration; /* 72 */
00265 struct __attribute__((__packed__))
00266 {
00267 char __pre_pad[2];
00268 L4_ktrace_t__Address user_ip; /* 48+8 */
00269 } timer; /* 56 */
00270 struct __attribute__((__packed__))
00271 {
00272 char __pre_pad[2];
00273 L4_ktrace_t__Mword exitcode; /* 48+8 */
00274 L4_ktrace_t__Mword exitinfo1; /* 56+8 */
00275 L4_ktrace_t__Mword exitinfo2; /* 64+8 */
00276 L4_ktrace_t__Mword rip; /* 72+8 */
00277 } svm; /* 80 */
00278 } m;
00279 } l4_tracebuffer_entry_t;

```

## 17.110 ktrace\_events.h

```

00001 /* Note, automatically generated from Fiasco binary */
00002 #pragma once
00003
00004 enum L4_ktrace_tbuf_entry_fixed
00005 {
00006 l4_ktrace_tbuf_unused = 0,
00007 l4_ktrace_tbuf_pf = 1,
00008 l4_ktrace_tbuf_ipc = 2,
00009 l4_ktrace_tbuf_ipc_res = 3,
00010 l4_ktrace_tbuf_ipc_trace = 4,
00011 l4_ktrace_tbuf_ke = 5,
00012 l4_ktrace_tbuf_ke_reg = 6,
00013 l4_ktrace_tbuf_breakpoint = 7,
00014 l4_ktrace_tbuf_ke_bin = 8,
00015 l4_ktrace_tbuf_dynentries = 9,
00016 l4_ktrace_tbuf_max = 128,
00017 l4_ktrace_tbuf_hidden = 128,
00018 };
00019
00020 typedef unsigned long L4_ktrace_t__Address;
00021 typedef unsigned long L4_ktrace_t__Cap_index;
00022 typedef void L4_ktrace_t__Context;
00023 typedef void L4_ktrace_t__Context__Drq;
00024 typedef unsigned L4_ktrace_t__Context__Drq_log__Type;
00025 typedef unsigned L4_ktrace_t__Cpu_number;
00026 typedef void L4_ktrace_t__Irq_base;
00027 typedef void L4_ktrace_t__Irq_chip;
00028 typedef void L4_ktrace_t__Kobject;
00029 typedef unsigned long L4_ktrace_t__L4_error;
00030 typedef unsigned long L4_ktrace_t__L4_msg_tag;
00031 typedef unsigned long L4_ktrace_t__L4_obj_ref;
00032 typedef unsigned L4_ktrace_t__L4_timeout_pair;
00033 typedef unsigned long L4_ktrace_t__Mword;
00034 typedef void L4_ktrace_t__Rcu_item;
00035 typedef void L4_ktrace_t__Sched_context;
00036 typedef long L4_ktrace_t__Sword;
00037 typedef void L4_ktrace_t__Space;

```

```

00038 typedef unsigned int L4_ktrace_t__Unsigned32;
00039 typedef unsigned long long L4_ktrace_t__Unsigned64;
00040 typedef unsigned char L4_ktrace_t__Unsigned8;
00041 typedef void L4_ktrace_t__cxx_Type_info;
00042
00043 typedef struct __attribute__((packed))
00044 {
00045 L4_ktrace_t__Mword _number; /* 0+4 */
00046 L4_ktrace_t__Address _ip; /* 4+4 */
00047 L4_ktrace_t__Unsigned64 _tsc; /* 8+8 */
00048 L4_ktrace_t__Context *_ctx; /* 16+4 */
00049 L4_ktrace_t__Unsigned32 _pmc1; /* 20+4 */
00050 L4_ktrace_t__Unsigned32 _pmc2; /* 24+4 */
00051 L4_ktrace_t__Unsigned32 _kclock; /* 28+4 */
00052 L4_ktrace_t__Unsigned8 _type; /* 32+1 */
00053 L4_ktrace_t__Unsigned8 _cpu; /* 33+1 */
00054 union __attribute__((__packed__))
00055 {
00056 struct __attribute__((__packed__))
00057 {
00058 char __pre_pad[2];
00059 void *func; /* 36+4 */
00060 L4_ktrace_t__Context *thread; /* 40+4 */
00061 L4_ktrace_t__Context__Drq *rq; /* 44+4 */
00062 L4_ktrace_t__Cpu_number target_cpu; /* 48+4 */
00063 L4_ktrace_t__Context__Drq_log__Type type; /* 52+4 */
00064 char wait; /* 56+1 */
00065 } drq; /* 64 */
00066 struct __attribute__((__packed__))
00067 {
00068 char __pre_pad[2];
00069 L4_ktrace_t__Mword state; /* 36+4 */
00070 L4_ktrace_t__Mword ip; /* 40+4 */
00071 L4_ktrace_t__Mword sp; /* 44+4 */
00072 L4_ktrace_t__Mword space; /* 48+4 */
00073 L4_ktrace_t__Mword err; /* 52+4 */
00074 unsigned char type; /* 56+1 */
00075 unsigned char trap; /* 57+1 */
00076 } vcpu; /* 64 */
00077 struct __attribute__((__packed__))
00078 {
00079 char __pre_pad[2];
00080 L4_ktrace_t__Smword op; /* 36+4 */
00081 L4_ktrace_t__Cap_index buffer; /* 40+4 */
00082 L4_ktrace_t__Mword id; /* 44+4 */
00083 L4_ktrace_t__Mword ram; /* 48+4 */
00084 L4_ktrace_t__Mword newo; /* 52+4 */
00085 } factory; /* 56 */
00086 struct __attribute__((__packed__))
00087 {
00088 char __pre_pad[2];
00089 L4_ktrace_t__Mword gate_dbg_id; /* 36+4 */
00090 L4_ktrace_t__Mword thread_dbg_id; /* 40+4 */
00091 L4_ktrace_t__Mword label; /* 44+4 */
00092 } gate; /* 48 */
00093 struct __attribute__((__packed__))
00094 {
00095 char __pre_pad[2];
00096 L4_ktrace_t__Irq_base *obj; /* 36+4 */
00097 L4_ktrace_t__Irq_chip *chip; /* 40+4 */
00098 L4_ktrace_t__Mword pin; /* 44+4 */
00099 } irq; /* 48 */
00100 struct __attribute__((__packed__))
00101 {
00102 char __pre_pad[2];
00103 L4_ktrace_t__Kobject *obj; /* 36+4 */
00104 L4_ktrace_t__Mword id; /* 40+4 */
00105 L4_ktrace_t__cxx_Type_info *type; /* 44+4 */
00106 L4_ktrace_t__Mword ram; /* 48+4 */
00107 } destroy; /* 56 */
00108 struct __attribute__((__packed__))
00109 {
00110 char __pre_pad[2];
00111 L4_ktrace_t__Cpu_number cpu; /* 36+4 */
00112 L4_ktrace_t__Rcu_item *item; /* 40+4 */
00113 void *cb; /* 44+4 */
00114 unsigned char event; /* 48+1 */
00115 } rcu; /* 56 */
00116 struct __attribute__((__packed__))
00117 {
00118 char __pre_pad[2];
00119 L4_ktrace_t__Mword id; /* 36+4 */
00120 L4_ktrace_t__Mword mask; /* 40+4 */
00121 L4_ktrace_t__Mword fpage; /* 44+4 */
00122 char map; /* 48+1 */
00123 } tmap; /* 56 */
00124 struct __attribute__((__packed__))

```

```

00125 {
00126 char __pre_pad[2];
00127 L4_ktrace_t__Address _address; /* 36+4 */
00128 int _len; /* 40+4 */
00129 L4_ktrace_t__Mword _value; /* 44+4 */
00130 int _mode; /* 48+4 */
00131 } bp; /* 56 */
00132 struct __attribute__((__packed__))
00133 {
00134 char __pre_pad[2];
00135 L4_ktrace_t__Context *dst; /* 36+4 */
00136 L4_ktrace_t__Context *dst_orig; /* 40+4 */
00137 L4_ktrace_t__Address kernel_ip; /* 44+4 */
00138 L4_ktrace_t__Mword lock_cnt; /* 48+4 */
00139 L4_ktrace_t__Space *from_space; /* 52+4 */
00140 L4_ktrace_t__Sched_context *from_sched; /* 56+4 */
00141 L4_ktrace_t__Mword from_prio; /* 60+4 */
00142 } context_switch; /* 64 */
00143 struct __attribute__((__packed__))
00144 {
00145 } empty; /* 40 */
00146 struct __attribute__((__packed__))
00147 {
00148 char __pre_pad[2];
00149 L4_ktrace_t__L4_msg_tag _tag; /* 36+4 */
00150 L4_ktrace_t__Mword _dword[2]; /* 40+8 */
00151 L4_ktrace_t__L4_obj_ref _dst; /* 48+4 */
00152 L4_ktrace_t__Mword _dbg_id; /* 52+4 */
00153 L4_ktrace_t__Mword _label; /* 56+4 */
00154 L4_ktrace_t__L4_timeout_pair _timeout; /* 60+4 */
00155 } ipc; /* 64 */
00156 struct __attribute__((__packed__))
00157 {
00158 L4_ktrace_t__Unsigned8 _have_snd; /* 34+1 */
00159 L4_ktrace_t__Unsigned8 _is_np; /* 35+1 */
00160 L4_ktrace_t__L4_msg_tag _tag; /* 36+4 */
00161 L4_ktrace_t__Mword _dword[2]; /* 40+8 */
00162 L4_ktrace_t__L4_error _result; /* 48+4 */
00163 L4_ktrace_t__Mword _from; /* 52+4 */
00164 L4_ktrace_t__L4_obj_ref _dst; /* 56+4 */
00165 L4_ktrace_t__Mword _pair_event; /* 60+4 */
00166 } ipc_res; /* 64 */
00167 struct __attribute__((__packed__))
00168 {
00169 char __pre_pad[2];
00170 union __attribute__((__packed__)) {
00171 char msg[24]; /* 0+24 */
00172 struct __attribute__((__packed__)) {
00173 char tag[2]; /* 0+2 */
00174 char __pad_1[2];
00175 char *ptr; /* 4+4 */
00176 } mptr; /* 0+8 */
00177 } msg; /* 36+24 */
00178 } ke; /* 64 */
00179 struct __attribute__((__packed__))
00180 {
00181 char _msg[24]; /* 34+24 */
00182 } ke_bin; /* 64 */
00183 struct __attribute__((__packed__))
00184 {
00185 char __pre_pad[2];
00186 L4_ktrace_t__Mword v[3]; /* 36+12 */
00187 union __attribute__((__packed__)) {
00188 char msg[12]; /* 0+12 */
00189 struct __attribute__((__packed__)) {
00190 char tag[2]; /* 0+2 */
00191 char __pad_1[2];
00192 char *ptr; /* 4+4 */
00193 } mptr; /* 0+8 */
00194 } msg; /* 48+12 */
00195 } ke_reg; /* 64 */
00196 struct __attribute__((__packed__))
00197 {
00198 char __pre_pad[2];
00199 L4_ktrace_t__Address _pfa; /* 36+4 */
00200 L4_ktrace_t__Mword _error; /* 40+4 */
00201 L4_ktrace_t__Space *_space; /* 44+4 */
00202 } pf; /* 48 */
00203 struct __attribute__((__packed__))
00204 {
00205 unsigned short mode; /* 34+2 */
00206 L4_ktrace_t__Context *owner; /* 36+4 */
00207 unsigned short id; /* 40+2 */
00208 unsigned short prio; /* 42+2 */
00209 long left; /* 44+4 */
00210 unsigned long quantum; /* 48+4 */
00211 } sched; /* 56 */

```

```

00212 struct __attribute__((__packed__))
00213 {
00214 char __pre_pad[2];
00215 L4_ktrace_t__Unsigned32 _error; /* 36+4 */
00216 L4_ktrace_t__Mword _cpsr; /* 40+4 */
00217 L4_ktrace_t__Mword _sp; /* 44+4 */
00218 } trap; /* 48 */
00219 struct __attribute__((__packed__))
00220 {
00221 char _padding[24]; /* 34+24 */
00222 char __post_pad[6]; /* 58+6 */
00223 } fullsize; /* 64 */
00224 struct __attribute__((__packed__))
00225 {
00226 char __pre_pad[2];
00227 L4_ktrace_t__Cap_index cap_idx; /* 36+4 */
00228 } ieh; /* 40 */
00229 struct __attribute__((__packed__))
00230 {
00231 char __pre_pad[2];
00232 L4_ktrace_t__Mword pfa; /* 36+4 */
00233 L4_ktrace_t__Cap_index cap_idx; /* 40+4 */
00234 L4_ktrace_t__Mword err; /* 44+4 */
00235 } ipfh; /* 48 */
00236 struct __attribute__((__packed__))
00237 {
00238 char __pre_pad[2];
00239 L4_ktrace_t__Mword id; /* 36+4 */
00240 L4_ktrace_t__Mword ip; /* 40+4 */
00241 L4_ktrace_t__Mword sp; /* 44+4 */
00242 L4_ktrace_t__Mword op; /* 48+4 */
00243 } exregs; /* 56 */
00244 struct __attribute__((__packed__))
00245 {
00246 char __pre_pad[2];
00247 L4_ktrace_t__Mword state; /* 36+4 */
00248 L4_ktrace_t__Address user_ip; /* 40+4 */
00249 L4_ktrace_t__Cpu_number src_cpu; /* 44+4 */
00250 L4_ktrace_t__Cpu_number target_cpu; /* 48+4 */
00251 } migration; /* 56 */
00252 struct __attribute__((__packed__))
00253 {
00254 char __pre_pad[2];
00255 L4_ktrace_t__Address user_ip; /* 36+4 */
00256 } timer; /* 40 */
00257 } m;
00258 } l4_tracebuffer_entry_t;

```

## 17.111 ktrace\_events.h

```

00001 /* Note, automatically generated from Fiasco binary */
00002 #pragma once
00003
00004 enum L4_ktrace_tbuf_entry_fixed
00005 {
00006 l4_ktrace_tbuf_unused = 0,
00007 l4_ktrace_tbuf_pf = 1,
00008 l4_ktrace_tbuf_ipc = 2,
00009 l4_ktrace_tbuf_ipc_res = 3,
00010 l4_ktrace_tbuf_ipc_trace = 4,
00011 l4_ktrace_tbuf_ke = 5,
00012 l4_ktrace_tbuf_ke_reg = 6,
00013 l4_ktrace_tbuf_breakpoint = 7,
00014 l4_ktrace_tbuf_ke_bin = 8,
00015 l4_ktrace_tbuf_dynentries = 9,
00016 l4_ktrace_tbuf_max = 128,
00017 l4_ktrace_tbuf_hidden = 128,
00018 };
00019
00020 typedef unsigned long L4_ktrace_t__Address;
00021 typedef unsigned long L4_ktrace_t__Cap_index;
00022 typedef void L4_ktrace_t__Context;
00023 typedef void L4_ktrace_t__Context__Drq;
00024 typedef unsigned L4_ktrace_t__Context__Drq_log__Type;
00025 typedef unsigned L4_ktrace_t__Cpu_number;
00026 typedef void L4_ktrace_t__Irq_base;
00027 typedef void L4_ktrace_t__Irq_chip;
00028 typedef void L4_ktrace_t__Kobject;
00029 typedef unsigned long L4_ktrace_t__L4_error;
00030 typedef unsigned long L4_ktrace_t__L4_msg_tag;
00031 typedef unsigned long L4_ktrace_t__L4_obj_ref;
00032 typedef unsigned L4_ktrace_t__L4_timeout_pair;
00033 typedef unsigned long L4_ktrace_t__Mword;

```

```

00034 typedef void L4_ktrace_t__Rcu_item;
00035 typedef void L4_ktrace_t__Sched_context;
00036 typedef long L4_ktrace_t__Smword;
00037 typedef void L4_ktrace_t__Space;
00038 typedef unsigned int L4_ktrace_t__Unsigned32;
00039 typedef unsigned long long L4_ktrace_t__Unsigned64;
00040 typedef unsigned char L4_ktrace_t__Unsigned8;
00041 typedef void L4_ktrace_t__cxx__Type_info;
00042
00043 typedef struct __attribute__((packed))
00044 {
00045 L4_ktrace_t__Mword _number; /* 0+8 */
00046 L4_ktrace_t__Address _ip; /* 8+8 */
00047 L4_ktrace_t__Unsigned64 _tsc; /* 16+8 */
00048 L4_ktrace_t__Context *_ctx; /* 24+8 */
00049 L4_ktrace_t__Unsigned32 _pmc1; /* 32+4 */
00050 L4_ktrace_t__Unsigned32 _pmc2; /* 36+4 */
00051 L4_ktrace_t__Unsigned32 _kclock; /* 40+4 */
00052 L4_ktrace_t__Unsigned8 _type; /* 44+1 */
00053 L4_ktrace_t__Unsigned8 _cpu; /* 45+1 */
00054 union __attribute__((__packed__))
00055 {
00056 struct __attribute__((__packed__))
00057 {
00058 char __pre_pad[2];
00059 void *func; /* 48+8 */
00060 L4_ktrace_t__Context *thread; /* 56+8 */
00061 L4_ktrace_t__Context__Drq *rq; /* 64+8 */
00062 L4_ktrace_t__Cpu_number target_cpu; /* 72+4 */
00063 L4_ktrace_t__Context__Drq_log__Type type; /* 76+4 */
00064 char wait; /* 80+1 */
00065 } drq; /* 88 */
00066 struct __attribute__((__packed__))
00067 {
00068 char __pre_pad[2];
00069 L4_ktrace_t__Mword state; /* 48+8 */
00070 L4_ktrace_t__Mword ip; /* 56+8 */
00071 L4_ktrace_t__Mword sp; /* 64+8 */
00072 L4_ktrace_t__Mword space; /* 72+8 */
00073 L4_ktrace_t__Mword err; /* 80+8 */
00074 unsigned char type; /* 88+1 */
00075 unsigned char trap; /* 89+1 */
00076 } vcpu; /* 96 */
00077 struct __attribute__((__packed__))
00078 {
00079 char __pre_pad[2];
00080 L4_ktrace_t__Smword op; /* 48+8 */
00081 L4_ktrace_t__Cap_index buffer; /* 56+8 */
00082 L4_ktrace_t__Mword id; /* 64+8 */
00083 L4_ktrace_t__Mword ram; /* 72+8 */
00084 L4_ktrace_t__Mword newo; /* 80+8 */
00085 } factory; /* 88 */
00086 struct __attribute__((__packed__))
00087 {
00088 char __pre_pad[2];
00089 L4_ktrace_t__Mword gate_dbg_id; /* 48+8 */
00090 L4_ktrace_t__Mword thread_dbg_id; /* 56+8 */
00091 L4_ktrace_t__Mword label; /* 64+8 */
00092 gate; /* 72 */
00093 } struct __attribute__((__packed__))
00094 {
00095 char __pre_pad[2];
00096 L4_ktrace_t__Irq_base *obj; /* 48+8 */
00097 L4_ktrace_t__Irq_chip *chip; /* 56+8 */
00098 L4_ktrace_t__Mword pin; /* 64+8 */
00099 } irq; /* 72 */
00100 struct __attribute__((__packed__))
00101 {
00102 char __pre_pad[2];
00103 L4_ktrace_t__Kobject *obj; /* 48+8 */
00104 L4_ktrace_t__Mword id; /* 56+8 */
00105 L4_ktrace_t__cxx__Type_info *type; /* 64+8 */
00106 L4_ktrace_t__Mword ram; /* 72+8 */
00107 } destroy; /* 80 */
00108 struct __attribute__((__packed__))
00109 {
00110 char __pre_pad[2];
00111 L4_ktrace_t__Cpu_number cpu; /* 48+4 */
00112 char __pad1[4];
00113 L4_ktrace_t__Rcu_item *item; /* 56+8 */
00114 void *cb; /* 64+8 */
00115 unsigned char event; /* 72+1 */
00116 } rcu; /* 80 */
00117 struct __attribute__((__packed__))
00118 {
00119 char __pre_pad[2];
00120 L4_ktrace_t__Mword id; /* 48+8 */

```



```

00121 L4_ktrace_t__Mword mask; /* 56+8 */
00122 L4_ktrace_t__Mword fpage; /* 64+8 */
00123 char map; /* 72+1 */
00124 } tmap; /* 80 */
00125 struct __attribute__((__packed__))
00126 {
00127 char __pre_pad[2];
00128 L4_ktrace_t__Address _address; /* 48+8 */
00129 int _len; /* 56+4 */
00130 char __pad_1[4];
00131 L4_ktrace_t__Mword _value; /* 64+8 */
00132 int _mode; /* 72+4 */
00133 } bp; /* 80 */
00134 struct __attribute__((__packed__))
00135 {
00136 char __pre_pad[2];
00137 L4_ktrace_t__Context *dst; /* 48+8 */
00138 L4_ktrace_t__Context *dst_orig; /* 56+8 */
00139 L4_ktrace_t__Address kernel_ip; /* 64+8 */
00140 L4_ktrace_t__Mword lock_cnt; /* 72+8 */
00141 L4_ktrace_t__Space *from_space; /* 80+8 */
00142 L4_ktrace_t__Sched_context *from_sched; /* 88+8 */
00143 L4_ktrace_t__Mword from_prio; /* 96+8 */
00144 } context_switch; /* 104 */
00145 struct __attribute__((__packed__))
00146 {
00147 } empty; /* 48 */
00148 struct __attribute__((__packed__))
00149 {
00150 char __pre_pad[2];
00151 L4_ktrace_t__L4_msg_tag _tag; /* 48+8 */
00152 L4_ktrace_t__Mword _dword[2]; /* 56+16 */
00153 L4_ktrace_t__L4_obj_ref _dst; /* 72+8 */
00154 L4_ktrace_t__Mword _dbg_id; /* 80+8 */
00155 L4_ktrace_t__Mword _label; /* 88+8 */
00156 L4_ktrace_t__L4_timeout_pair _timeout; /* 96+4 */
00157 char __pad_1[4];
00158 L4_ktrace_t__Unsigned64 _to_abs_rcv; /* 104+8 */
00159 } ipc; /* 112 */
00160 struct __attribute__((__packed__))
00161 {
00162 L4_ktrace_t__Unsigned8 _have_snd; /* 46+1 */
00163 L4_ktrace_t__Unsigned8 _is_np; /* 47+1 */
00164 L4_ktrace_t__L4_msg_tag _tag; /* 48+8 */
00165 L4_ktrace_t__Mword _dword[2]; /* 56+16 */
00166 L4_ktrace_t__L4_error_result; /* 72+8 */
00167 L4_ktrace_t__Mword _from; /* 80+8 */
00168 L4_ktrace_t__L4_obj_ref _dst; /* 88+8 */
00169 L4_ktrace_t__Mword _pair_event; /* 96+8 */
00170 } ipc_res; /* 104 */
00171 struct __attribute__((__packed__))
00172 {
00173 char __pre_pad[2];
00174 union __attribute__((__packed__)) {
00175 char msg[80]; /* 0+80 */
00176 struct __attribute__((__packed__)) {
00177 char tag[2]; /* 0+2 */
00178 char __pad_1[6];
00179 char *ptr; /* 8+8 */
00180 } mptr; /* 0+16 */
00181 } msg; /* 48+80 */
00182 } ke; /* 128 */
00183 struct __attribute__((__packed__))
00184 {
00185 char _msg[80]; /* 46+80 */
00186 } ke_bin; /* 128 */
00187 struct __attribute__((__packed__))
00188 {
00189 char __pre_pad[2];
00190 L4_ktrace_t__Mword v[3]; /* 48+24 */
00191 union __attribute__((__packed__)) {
00192 char msg[56]; /* 0+56 */
00193 struct __attribute__((__packed__)) {
00194 char tag[2]; /* 0+2 */
00195 char __pad_1[6];
00196 char *ptr; /* 8+8 */
00197 } mptr; /* 0+16 */
00198 } msg; /* 72+56 */
00199 } ke_reg; /* 128 */
00200 struct __attribute__((__packed__))
00201 {
00202 char __pre_pad[2];
00203 L4_ktrace_t__Address _pfa; /* 48+8 */
00204 L4_ktrace_t__Mword _error; /* 56+8 */
00205 L4_ktrace_t__Space *_space; /* 64+8 */
00206 } pf; /* 72 */
00207 struct __attribute__((__packed__))

```

```

00208 {
00209 unsigned short mode; /* 46+2 */
00210 L4_ktrace_t__Context *owner; /* 48+8 */
00211 unsigned short id; /* 56+2 */
00212 unsigned short prio; /* 58+2 */
00213 char __pad_l[4];
00214 long left; /* 64+8 */
00215 unsigned long quantum; /* 72+8 */
00216 } sched; /* 80 */
00217 struct __attribute__((__packed__))
00218 {
00219 char __pre_pad[2];
00220 L4_ktrace_t__Unsigned32 _error; /* 48+4 */
00221 char __pad_l[4];
00222 L4_ktrace_t__Mword _cpsr; /* 56+8 */
00223 L4_ktrace_t__Mword _sp; /* 64+8 */
00224 } trap; /* 72 */
00225 struct __attribute__((__packed__))
00226 {
00227 char _padding[80]; /* 46+80 */
00228 char __post_pad[2]; /* 126+2 */
00229 } fullsize; /* 128 */
00230 struct __attribute__((__packed__))
00231 {
00232 char __pre_pad[2];
00233 L4_ktrace_t__Cap_index cap_idx; /* 48+8 */
00234 } ieh; /* 56 */
00235 struct __attribute__((__packed__))
00236 {
00237 char __pre_pad[2];
00238 L4_ktrace_t__Mword pfa; /* 48+8 */
00239 L4_ktrace_t__Cap_index cap_idx; /* 56+8 */
00240 L4_ktrace_t__Mword err; /* 64+8 */
00241 } ipfh; /* 72 */
00242 struct __attribute__((__packed__))
00243 {
00244 char __pre_pad[2];
00245 L4_ktrace_t__Mword id; /* 48+8 */
00246 L4_ktrace_t__Mword ip; /* 56+8 */
00247 L4_ktrace_t__Mword sp; /* 64+8 */
00248 L4_ktrace_t__Mword op; /* 72+8 */
00249 } exregs; /* 80 */
00250 struct __attribute__((__packed__))
00251 {
00252 char __pre_pad[2];
00253 L4_ktrace_t__Mword state; /* 48+8 */
00254 L4_ktrace_t__Address user_ip; /* 56+8 */
00255 L4_ktrace_t__Cpu_number src_cpu; /* 64+4 */
00256 L4_ktrace_t__Cpu_number target_cpu; /* 68+4 */
00257 } migration; /* 72 */
00258 struct __attribute__((__packed__))
00259 {
00260 char __pre_pad[2];
00261 L4_ktrace_t__Address user_ip; /* 48+8 */
00262 } timer; /* 56 */
00263 } m;
00264 } l4_tracebuffer_entry_t;

```

## 17.112 ktrace\_events.h

```

00001 /* Note, automatically generated from Fiasco binary */
00002 #pragma once
00003
00004 enum L4_ktrace_tbuf_entry_fixed
00005 {
00006 l4_ktrace_tbuf_unused = 0,
00007 l4_ktrace_tbuf_pf = 1,
00008 l4_ktrace_tbuf_ipc = 2,
00009 l4_ktrace_tbuf_ipc_res = 3,
00010 l4_ktrace_tbuf_ipc_trace = 4,
00011 l4_ktrace_tbuf_ke = 5,
00012 l4_ktrace_tbuf_ke_reg = 6,
00013 l4_ktrace_tbuf_breakpoint = 7,
00014 l4_ktrace_tbuf_ke_bin = 8,
00015 l4_ktrace_tbuf_dynentries = 9,
00016 l4_ktrace_tbuf_max = 128,
00017 l4_ktrace_tbuf_hidden = 128,
00018 };
00019
00020 typedef unsigned long L4_ktrace_t__Address;
00021 typedef unsigned long L4_ktrace_t__Cap_index;
00022 typedef void L4_ktrace_t__Context;
00023 typedef void L4_ktrace_t__Context__Drq;

```

```

00024 typedef unsigned L4_ktrace_t__Context__Drq_log__Type;
00025 typedef unsigned L4_ktrace_t__Cpu_number;
00026 typedef void L4_ktrace_t__Irq_base;
00027 typedef void L4_ktrace_t__Irq_chip;
00028 typedef void L4_ktrace_t__Kobject;
00029 typedef unsigned long L4_ktrace_t__L4_error;
00030 typedef unsigned long L4_ktrace_t__L4_msg_tag;
00031 typedef unsigned long L4_ktrace_t__L4_obj_ref;
00032 typedef unsigned L4_ktrace_t__L4_timeout_pair;
00033 typedef unsigned long L4_ktrace_t__Mword;
00034 typedef void L4_ktrace_t__Rcu_item;
00035 typedef void L4_ktrace_t__Sched_context;
00036 typedef long L4_ktrace_t__Smword;
00037 typedef void L4_ktrace_t__Space;
00038 typedef unsigned short L4_ktrace_t__Unsigned16;
00039 typedef unsigned int L4_ktrace_t__Unsigned32;
00040 typedef unsigned long long L4_ktrace_t__Unsigned64;
00041 typedef unsigned char L4_ktrace_t__Unsigned8;
00042 typedef void L4_ktrace_t__cxx__Type_info;
00043
00044 typedef struct __attribute__((packed))
00045 {
00046 L4_ktrace_t__Mword _number; /* 0+4 */
00047 L4_ktrace_t__Address _ip; /* 4+4 */
00048 L4_ktrace_t__Unsigned64 _tsc; /* 8+8 */
00049 L4_ktrace_t__Context *_ctx; /* 16+4 */
00050 L4_ktrace_t__Unsigned32 _pmc1; /* 20+4 */
00051 L4_ktrace_t__Unsigned32 _pmc2; /* 24+4 */
00052 L4_ktrace_t__Unsigned32 _kclock; /* 28+4 */
00053 L4_ktrace_t__Unsigned8 _type; /* 32+1 */
00054 L4_ktrace_t__Unsigned8 _cpu; /* 33+1 */
00055 union __attribute__((__packed__))
00056 {
00057 struct __attribute__((__packed__))
00058 {
00059 char __pre_pad[2];
00060 void *func; /* 36+4 */
00061 L4_ktrace_t__Context *thread; /* 40+4 */
00062 L4_ktrace_t__Context__Drq *rq; /* 44+4 */
00063 L4_ktrace_t__Cpu_number target_cpu; /* 48+4 */
00064 L4_ktrace_t__Context__Drq_log__Type type; /* 52+4 */
00065 char wait; /* 56+1 */
00066 } drq; /* 64 */
00067 struct __attribute__((__packed__))
00068 {
00069 char __pre_pad[2];
00070 L4_ktrace_t__Mword state; /* 36+4 */
00071 L4_ktrace_t__Mword ip; /* 40+4 */
00072 L4_ktrace_t__Mword sp; /* 44+4 */
00073 L4_ktrace_t__Mword space; /* 48+4 */
00074 L4_ktrace_t__Mword err; /* 52+4 */
00075 unsigned char type; /* 56+1 */
00076 unsigned char trap; /* 57+1 */
00077 } vcpu; /* 64 */
00078 struct __attribute__((__packed__))
00079 {
00080 char __pre_pad[2];
00081 L4_ktrace_t__Smword op; /* 36+4 */
00082 L4_ktrace_t__Cap_index buffer; /* 40+4 */
00083 L4_ktrace_t__Mword id; /* 44+4 */
00084 L4_ktrace_t__Mword ram; /* 48+4 */
00085 L4_ktrace_t__Mword newo; /* 52+4 */
00086 } factory; /* 56 */
00087 struct __attribute__((__packed__))
00088 {
00089 char __pre_pad[2];
00090 L4_ktrace_t__Mword gate_dbg_id; /* 36+4 */
00091 L4_ktrace_t__Mword thread_dbg_id; /* 40+4 */
00092 L4_ktrace_t__Mword label; /* 44+4 */
00093 } gate; /* 48 */
00094 struct __attribute__((__packed__))
00095 {
00096 char __pre_pad[2];
00097 L4_ktrace_t__Irq_base *obj; /* 36+4 */
00098 L4_ktrace_t__Irq_chip *chip; /* 40+4 */
00099 L4_ktrace_t__Mword pin; /* 44+4 */
00100 } irq; /* 48 */
00101 struct __attribute__((__packed__))
00102 {
00103 char __pre_pad[2];
00104 L4_ktrace_t__Kobject *obj; /* 36+4 */
00105 L4_ktrace_t__Mword id; /* 40+4 */
00106 L4_ktrace_t__cxx__Type_info *type; /* 44+4 */
00107 L4_ktrace_t__Mword ram; /* 48+4 */
00108 } destroy; /* 56 */
00109 struct __attribute__((__packed__))
00110 {

```

```

00111 char __pre_pad[2];
00112 L4_ktrace_t__Cpu_number cpu; /* 36+4 */
00113 L4_ktrace_t__Rcu_item *item; /* 40+4 */
00114 void *cb; /* 44+4 */
00115 unsigned char event; /* 48+1 */
00116 } rcu; /* 56 */
00117 struct __attribute__((__packed__))
00118 {
00119 char __pre_pad[2];
00120 L4_ktrace_t__Mword id; /* 36+4 */
00121 L4_ktrace_t__Mword mask; /* 40+4 */
00122 L4_ktrace_t__Mword fpage; /* 44+4 */
00123 char map; /* 48+1 */
00124 } tmap; /* 56 */
00125 struct __attribute__((__packed__))
00126 {
00127 char __pre_pad[2];
00128 L4_ktrace_t__Address _address; /* 36+4 */
00129 int _len; /* 40+4 */
00130 L4_ktrace_t__Mword _value; /* 44+4 */
00131 int _mode; /* 48+4 */
00132 } bp; /* 56 */
00133 struct __attribute__((__packed__))
00134 {
00135 char __pre_pad[2];
00136 L4_ktrace_t__Context *dst; /* 36+4 */
00137 L4_ktrace_t__Context *dst_orig; /* 40+4 */
00138 L4_ktrace_t__Address kernel_ip; /* 44+4 */
00139 L4_ktrace_t__Mword lock_cnt; /* 48+4 */
00140 L4_ktrace_t__Space *from_space; /* 52+4 */
00141 L4_ktrace_t__Sched_context *from_sched; /* 56+4 */
00142 L4_ktrace_t__Mword from_prio; /* 60+4 */
00143 } context_switch; /* 64 */
00144 struct __attribute__((__packed__))
00145 {
00146 } empty; /* 40 */
00147 struct __attribute__((__packed__))
00148 {
00149 char __pre_pad[2];
00150 L4_ktrace_t__L4_msg_tag _tag; /* 36+4 */
00151 L4_ktrace_t__Mword _dword[2]; /* 40+8 */
00152 L4_ktrace_t__L4_obj_ref _dst; /* 48+4 */
00153 L4_ktrace_t__Mword _dbg_id; /* 52+4 */
00154 L4_ktrace_t__Mword _label; /* 56+4 */
00155 L4_ktrace_t__L4_timeout_pair _timeout; /* 60+4 */
00156 } ipc; /* 64 */
00157 struct __attribute__((__packed__))
00158 {
00159 L4_ktrace_t__Unsigned8 _have_snd; /* 34+1 */
00160 L4_ktrace_t__Unsigned8 _is_np; /* 35+1 */
00161 L4_ktrace_t__L4_msg_tag _tag; /* 36+4 */
00162 L4_ktrace_t__Mword _dword[2]; /* 40+8 */
00163 L4_ktrace_t__L4_error _result; /* 48+4 */
00164 L4_ktrace_t__Mword _from; /* 52+4 */
00165 L4_ktrace_t__L4_obj_ref _dst; /* 56+4 */
00166 L4_ktrace_t__Mword _pair_event; /* 60+4 */
00167 } ipc_res; /* 64 */
00168 struct __attribute__((__packed__))
00169 {
00170 char __pre_pad[2];
00171 union __attribute__((__packed__)) {
00172 char msg[24]; /* 0+24 */
00173 struct __attribute__((__packed__)) {
00174 char tag[2]; /* 0+2 */
00175 char __pad_1[2];
00176 char *ptr; /* 4+4 */
00177 } mptr; /* 0+8 */
00178 } msg; /* 36+24 */
00179 } ke; /* 64 */
00180 struct __attribute__((__packed__))
00181 {
00182 char _msg[24]; /* 34+24 */
00183 } ke_bin; /* 64 */
00184 struct __attribute__((__packed__))
00185 {
00186 char __pre_pad[2];
00187 L4_ktrace_t__Mword v[3]; /* 36+12 */
00188 union __attribute__((__packed__)) {
00189 char msg[12]; /* 0+12 */
00190 struct __attribute__((__packed__)) {
00191 char tag[2]; /* 0+2 */
00192 char __pad_1[2];
00193 char *ptr; /* 4+4 */
00194 } mptr; /* 0+8 */
00195 } msg; /* 48+12 */
00196 } ke_reg; /* 64 */
00197 struct __attribute__((__packed__))

```

```

00198 {
00199 char __pre_pad[2];
00200 L4_ktrace_t__Address _pfa; /* 36+4 */
00201 L4_ktrace_t__Mword _error; /* 40+4 */
00202 L4_ktrace_t__Space *_space; /* 44+4 */
00203 } pf; /* 48 */
00204 struct __attribute__((__packed__))
00205 {
00206 unsigned short mode; /* 34+2 */
00207 L4_ktrace_t__Context *owner; /* 36+4 */
00208 unsigned short id; /* 40+2 */
00209 unsigned short prio; /* 42+2 */
00210 long left; /* 44+4 */
00211 unsigned long quantum; /* 48+4 */
00212 } sched; /* 56 */
00213 struct __attribute__((__packed__))
00214 {
00215 L4_ktrace_t__Unsigned8 _trapno; /* 34+1 */
00216 char __pad_1[1];
00217 L4_ktrace_t__Unsigned16 _error; /* 36+2 */
00218 char __pad_2[2];
00219 L4_ktrace_t__Mword _ebp; /* 40+4 */
00220 L4_ktrace_t__Mword _cr2; /* 44+4 */
00221 L4_ktrace_t__Mword _eax; /* 48+4 */
00222 L4_ktrace_t__Mword _eflags; /* 52+4 */
00223 L4_ktrace_t__Mword _esp; /* 56+4 */
00224 L4_ktrace_t__Unsigned16 _cs; /* 60+2 */
00225 L4_ktrace_t__Unsigned16 _ds; /* 62+2 */
00226 } trap; /* 64 */
00227 struct __attribute__((__packed__))
00228 {
00229 char __padding[24]; /* 34+24 */
00230 char __post_pad[6]; /* 58+6 */
00231 } fullsize; /* 64 */
00232 struct __attribute__((__packed__))
00233 {
00234 char __pre_pad[2];
00235 L4_ktrace_t__Cap_index cap_idx; /* 36+4 */
00236 } ieh; /* 40 */
00237 struct __attribute__((__packed__))
00238 {
00239 char __pre_pad[2];
00240 L4_ktrace_t__Mword pfa; /* 36+4 */
00241 L4_ktrace_t__Cap_index cap_idx; /* 40+4 */
00242 L4_ktrace_t__Mword err; /* 44+4 */
00243 } ipfh; /* 48 */
00244 struct __attribute__((__packed__))
00245 {
00246 char __pre_pad[2];
00247 L4_ktrace_t__Mword id; /* 36+4 */
00248 L4_ktrace_t__Mword ip; /* 40+4 */
00249 L4_ktrace_t__Mword sp; /* 44+4 */
00250 L4_ktrace_t__Mword op; /* 48+4 */
00251 } exregs; /* 56 */
00252 struct __attribute__((__packed__))
00253 {
00254 char __pre_pad[2];
00255 L4_ktrace_t__Mword state; /* 36+4 */
00256 L4_ktrace_t__Address user_ip; /* 40+4 */
00257 L4_ktrace_t__Cpu_number src_cpu; /* 44+4 */
00258 L4_ktrace_t__Cpu_number target_cpu; /* 48+4 */
00259 } migration; /* 56 */
00260 struct __attribute__((__packed__))
00261 {
00262 char __pre_pad[2];
00263 L4_ktrace_t__Address user_ip; /* 36+4 */
00264 } timer; /* 40 */
00265 struct __attribute__((__packed__))
00266 {
00267 char __pre_pad[2];
00268 L4_ktrace_t__Mword exitcode; /* 36+4 */
00269 L4_ktrace_t__Mword exitinfo1; /* 40+4 */
00270 L4_ktrace_t__Mword exitinfo2; /* 44+4 */
00271 L4_ktrace_t__Mword rip; /* 48+4 */
00272 } svm; /* 56 */
00273 } m;
00274 } l4_tracebuffer_entry_t;

```

## 17.113 amd64/l4/sys/linkage.h File Reference

Linkage.

## Macros

- **#define L4\_CV**  
*Define calling convention.*

### 17.113.1 Detailed Description

Linkage.

Definition in file [linkage.h](#).

## 17.114 linkage.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00009 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #ifndef __L4__SYS__ARCH_AMD64__LINKAGE_H__
00015 #define __L4__SYS__ARCH_AMD64__LINKAGE_H__
00016
00017 #ifdef __ASSEMBLY__
00018
00019 #ifndef ENTRY
00020 #define ENTRY(name) \
00021 .globl name; \
00022 .p2align(2); \
00023 name:
00024
00025 #endif /* __ASSEMBLY__ */
00026 #endif /* ! ENTRY */
00027
00028 #define L4_FASTCALL(x) x
00029 #define l4_fastcall
00030
00036 #define L4_CV
00037
00038 #endif /* ! __L4__SYS__ARCH_AMD64__LINKAGE_H__ */

```

### 17.115 arm/l4/sys/linkage.h File Reference

Linkage.

## Macros

- **#define L4\_CV**  
*Define calling convention.*

### 17.115.1 Detailed Description

Linkage.

Definition in file [linkage.h](#).

## 17.116 linkage.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #ifndef __L4__SYS__ARCH_ARM__LINKAGE_H__
00014 #define __L4__SYS__ARCH_ARM__LINKAGE_H__
00015
00016 #ifdef __ASSEMBLY__
00017 #ifndef ENTRY
00018 #define ENTRY(name) \
00019 .globl name; \
00020 .p2align(2); \
00021 name:
00022 #endif
00023 #endif
00024
00025 #define L4_FASTCALL(x) x
00026 #define l4_fastcall
00027
00033 #define L4_CV
00034
00035 #ifdef __PIC__
00036 # define L4_LONG_CALL
00037 #else
00038 # define L4_LONG_CALL __attribute__((long_call))
00039 #endif
00040
00041 #endif /* ! __L4__SYS__ARCH_ARM__LINKAGE_H__ */

```

## 17.117 linkage.h

```

00001 #pragma once
00002
00008 #define L4_CV
00009
00010 #define L4_FASTCALL(x) x
00011 #define l4_fastcall

```

## 17.118 x86/I4/sys/linkage.h File Reference

Linkage.

### Macros

- **#define L4\_CV**  
*Define calling convention.*

### 17.118.1 Detailed Description

Linkage.

Definition in file [linkage.h](#).

## 17.119 linkage.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00009 * Frank Mehnert <fm3@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #ifndef __L4__SYS__ARCH_X86__LINKAGE_H__
00015 #define __L4__SYS__ARCH_X86__LINKAGE_H__
00016
00017 #ifdef __ASSEMBLY__
00018
00019 #ifndef ENTRY
00020 #define ENTRY(name) \
00021 .globl name; \
00022 .p2align(2); \
00023 name:
00024
00025 #endif /* ! ENTRY */
00026 #endif /* __ASSEMBLY__ */
00027
00028 #define L4_FASTCALL(x) x __attribute__((regparm(3)))
00029 #define l4_fastcall __attribute__((regparm(3)))
00030
00036 #define L4_CV __attribute__((regparm(0)))
00037
00038 #endif /* ! __L4__SYS__ARCH_X86__LINKAGE_H__ */

```

## 17.120 arm/l4/sys/mem\_op.h File Reference

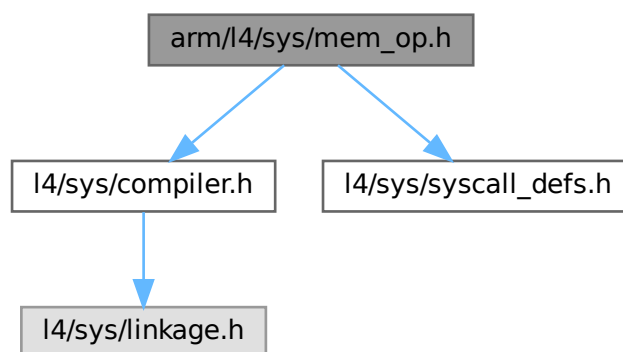
Memory access functions (ARM specific).

```

#include <l4/sys/compiler.h>
#include <l4/sys/syscall_defs.h>

```

Include dependency graph for mem\_op.h:



### Enumerations

- enum `L4_mem_op_widths` { `L4_MEM_WIDTH_1BYTE` = 0, `L4_MEM_WIDTH_2BYTE` = 1, `L4_MEM_WIDTH_4BYTE` = 2 }
- Memory access width definitions.*



## Functions

- unsigned long [l4\\_mem\\_read](#) (unsigned long virtaddress, unsigned width)  
*Read user task memory from kernel privilege level.*
- void [l4\\_mem\\_write](#) (unsigned long virtaddress, unsigned width, unsigned long value)  
*Write user task memory from kernel privilege level.*
- unsigned long [l4\\_mem\\_arm\\_op\\_call](#) (unsigned long op, unsigned long va, unsigned long width, unsigned long value)  
*Implementations.*

### 17.120.1 Detailed Description

Memory access functions (ARM specific).

#### Date

2010-10

#### Author

Adam Lackorzynski [adam@os.inf.tu-dresden.de](mailto:adam@os.inf.tu-dresden.de)

Definition in file [mem\\_op.h](#).

## 17.121 mem\_op.h

[Go to the documentation of this file.](#)

```

00001
00009 /*
00010 * (c) 2010 Author(s)
00011 * economic rights: Technische Universität Dresden (Germany)
00012 *
00013 * License: see LICENSE.spdx (in this directory or the directories above)
00014 */
00015 #ifndef __L4SYS__INCLUDE__ARCH_ARM_MEM_OP_H__
00016 #define __L4SYS__INCLUDE__ARCH_ARM_MEM_OP_H__
00017
00018 #include <l4/sys/compiler.h>
00019 #include <l4/sys/syscall_defs.h>
00020
00021 L4_BEGIN_DECLS
00022
00035
00040 enum L4_mem_op_widths
00041 {
00042 L4_MEM_WIDTH_1BYTE = 0,
00043 L4_MEM_WIDTH_2BYTE = 1,
00044 L4_MEM_WIDTH_4BYTE = 2,
00045 };
00046
00059 L4_INLINE unsigned long
00060 l4_mem_read(unsigned long virtaddress, unsigned width);
00061
00074 L4_INLINE void
00075 l4_mem_write(unsigned long virtaddress, unsigned width,
00076 unsigned long value);
00077
00078 enum L4_mem_ops
00079 {
00080 L4_MEM_OP_MEM_READ = 0x10,
00081 L4_MEM_OP_MEM_WRITE = 0x11,
00082 };
00083
00087 L4_INLINE unsigned long

```

```

00088 l4_mem_arm_op_call(unsigned long op,
00089 unsigned long va,
00090 unsigned long width,
00091 unsigned long value);
00092
00093
00094
00095 L4_INLINE unsigned long
00096 l4_mem_arm_op_call(unsigned long op,
00097 unsigned long va,
00098 unsigned long width,
00099 unsigned long value)
00100 {
00101 register unsigned long _op __asm__ ("r0") = op;
00102 register unsigned long _va __asm__ ("r1") = va;
00103 register unsigned long _width __asm__ ("r2") = width;
00104 register unsigned long _value __asm__ ("r3") = value;
00105
00106 __asm__ __volatile__
00107 ("@ l4_cache_op_arm_call(start) \n\t"
00108 "mov r5, %[sc] \n\t"
00109 "blx __l4_sys_syscall \n\t"
00110 "@ l4_cache_op_arm_call(end) \n\t"
00111 :
00112 "=r" (_op),
00113 "=r" (_va),
00114 "=r" (_width),
00115 "=r" (_value)
00116 :
00117 [sc] "i" (L4_SYSCALL_MEM_OP),
00118 "0" (_op),
00119 "1" (_va),
00120 "2" (_width),
00121 "3" (_value)
00122 :
00123 "cc", "memory", "r5", "ip", "lr"
00124);
00125
00126 return _value;
00127 }
00128
00129 L4_INLINE unsigned long
00130 l4_mem_read(unsigned long virtaddress, unsigned width)
00131 {
00132 return l4_mem_arm_op_call(L4_MEM_OP_MEM_READ, virtaddress, width, 0);
00133 }
00134
00135 L4_INLINE void
00136 l4_mem_write(unsigned long virtaddress, unsigned width,
00137 unsigned long value)
00138 {
00139 l4_mem_arm_op_call(L4_MEM_OP_MEM_WRITE, virtaddress, width, value);
00140 }
00141
00142 L4_END_DECLS
00143
00144 #endif /* ! __L4SYS__INCLUDE__ARCH_ARM__MEM_OP_H__ */

```

## 17.122 vm.h

```

00001
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include <l4/sys/__vm-svm.h>
00011 #include <l4/sys/__vm-vmx.h>

```

## 17.123 arm/l4/sys/vm.h File Reference

ARM virtualization interface.

## Data Structures

- struct [l4\\_vm\\_tz\\_state](#)  
*state structure for TrustZone VMs*

### 17.123.1 Detailed Description

ARM virtualization interface.

Definition in file [vm.h](#).

## 17.124 vm.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00019
00024 struct l4_vm_tz_state_mode
00025 {
00026 l4_umword_t sp;
00027 l4_umword_t lr;
00028 l4_umword_t spsr;
00029 };
00030
00031 struct l4_vm_tz_state_irq_inject
00032 {
00033 l4_uint32_t group;
00034 l4_uint32_t irqs[8];
00035 };
00036
00041 struct l4_vm_tz_state
00042 {
00043 l4_umword_t r[13]; // r0 - r12
00044
00045 l4_umword_t sp_usr;
00046 l4_umword_t lr_usr;
00047
00048 struct l4_vm_tz_state_mode irq;
00049
00050 l4_umword_t r_fiq[5]; // r8 - r12
00051 struct l4_vm_tz_state_mode fiq;
00052 struct l4_vm_tz_state_mode abt;
00053 struct l4_vm_tz_state_mode und;
00054 struct l4_vm_tz_state_mode svc;
00055
00056 l4_umword_t pc;
00057 l4_umword_t cpsr;
00058
00059 l4_umword_t pending_events;
00060 l4_uint32_t cpacr;
00061 l4_umword_t cpl0_fpexc;
00062
00063 l4_umword_t pfs;
00064 l4_umword_t pfa;
00065 l4_umword_t exit_reason;
00066
00067 struct l4_vm_tz_state_irq_inject irq_inject;
00068 };
00069
00070 enum L4_vm_exit_reason
00071 {
00072 L4_vm_exit_reason_vmm_call = 1,
00073 L4_vm_exit_reason_inst_abort = 2,
00074 L4_vm_exit_reason_data_abort = 3,
00075 L4_vm_exit_reason_irq = 4,

```

```
00076 L4_vm_exit_reason_fiq = 5,
00077 L4_vm_exit_reason_undef = 6,
00078 };
00079
00080 L4_INLINE int
00081 l4_vm_tz_irq_inject(struct l4_vm_tz_state *state, unsigned irq);
00082
00083 L4_INLINE int
00084 l4_vm_tz_irq_inject(struct l4_vm_tz_state *state, unsigned irq)
00085 {
00086 if (irq > sizeof(state->irq_inject.irqs) * 8)
00087 return -L4_EINVAL;
00088
00089 unsigned g = irq / 32;
00090 state->irq_inject.group |= 1 « g;
00091 state->irq_inject.irqs[g] |= 1 « (irq & 31);
00092
00093 return 0;
00094 }
```

## 17.125 vm.h

```
00001
00002
```

## 17.126 vm.h

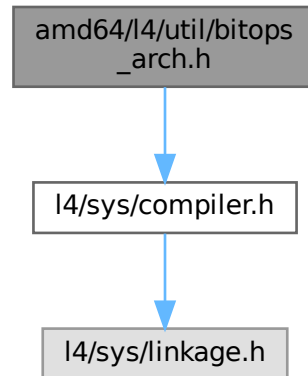
```
00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Henning Schild <hschild@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #pragma once
00014
00015 #include <l4/sys/__vm-svm.h>
00016 #include <l4/sys/__vm-vmx.h>
```

## 17.127 amd64/l4/util/bitops\_arch.h File Reference

amd64 bit manipulation functions

```
#include <l4/sys/compiler.h>
```

Include dependency graph for bitops\_arch.h:



### 17.127.1 Detailed Description

amd64 bit manipulation functions

Definition in file [bitops\\_arch.h](#).

## 17.128 bitops\_arch.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * Copyright (C) 2000-2009 Technische Universität Dresden (Germany)
00003 * Copyright (C) 2016, 2022, 2024 Kernkonzept GmbH. All rights reserved.
00004 * Author(s): Lars Reuther <reuther@os.inf.tu-dresden.de>
00005 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00006 * Frank Mehnert <frank.mehnert@kernkonzept.com>
00007 *
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010
00016
00017 #pragma once
00018
00019 #include <l4/sys/compiler.h>
00020
00021 /*
00022 * Note: The following Assembler statement may produce wrong code:
00023 * asm volatile ("btsl %1, %2" : "=ccc"(r) : "Jr"(63), "m"(m) : "memory");
00024 *
00025 * The compiler might chose the first variant because the bit number is smaller
00026 * than 64. However, 'bts' is encoded as 32-bit variant ('btsl') and thus only
00027 * supports immediate bit values up to 31. Some assemblers support immediate
00028 * offsets > 31 by adapting the memory address accordingly but GAS does not.
00029 * With GAS, the instruction will encode an immediate value of 63 but the CPU
00030 * will set bit 31 instead of bit 63!
00031 *
00032 * Therefore, if we would use 'btsl' instead of 'btsq', the correct constraint
00033 * for the bit number parameter would be "Ir" instead of "Jr".
00034 */
00035
00036 L4_BEGIN_DECLS
00037

```

```

00038 /* set bit */
00039 #define __L4UTIL_BITOPS_HAVE_ARCH_SET_BIT
00040 L4_INLINE void
00041 l4util_set_bit(int b, volatile l4_umword_t * dest)
00042 {
00043 __asm__ __volatile__
00044 (
00045 "lock; btsq %1,%0 \n\t"
00046 :
00047 :
00048 "m" (*dest), /* 0 mem, destination operand */
00049 "Jr" ((l4_umword_t)b) /* 1, bit number */
00050 :
00051 "memory", "cc"
00052);
00053 }
00054
00055 /* clear bit */
00056 #define __L4UTIL_BITOPS_HAVE_ARCH_CLEAR_BIT
00057 L4_INLINE void
00058 l4util_clear_bit(int b, volatile l4_umword_t * dest)
00059 {
00060 __asm__ __volatile__
00061 (
00062 "lock; btrq %1,%0 \n\t"
00063 :
00064 :
00065 "m" (*dest), /* 0 mem, destination operand */
00066 "Jr" ((l4_umword_t)b) /* 1, bit number */
00067 :
00068 "memory", "cc"
00069);
00070 }
00071
00072 /* change bit */
00073 #define __L4UTIL_BITOPS_HAVE_ARCH_COMPLEMENT_BIT
00074 L4_INLINE void
00075 l4util_complement_bit(int b, volatile l4_umword_t * dest)
00076 {
00077 __asm__ __volatile__
00078 (
00079 "lock; btcq %1,%0 \n\t"
00080 :
00081 :
00082 "m" (*dest), /* 0 mem, destination operand */
00083 "Jr" ((l4_umword_t)b) /* 1, bit number */
00084 :
00085 "memory", "cc"
00086);
00087 }
00088
00089 /* test bit */
00090 #define __L4UTIL_BITOPS_HAVE_ARCH_TEST_BIT
00091 L4_INLINE int
00092 l4util_test_bit(int b, const volatile l4_umword_t * dest)
00093 {
00094 l4_int8_t bit;
00095
00096 __asm__ __volatile__
00097 (
00098 "btq %2,%1 \n\t"
00099 :
00100 "=@ccc" (bit) /* 0, old bit value */
00101 :
00102 "m" (*dest), /* 1 mem, destination operand */
00103 "Jr" ((l4_umword_t)b) /* 2, bit number */
00104 :
00105 "memory"
00106);
00107
00108 return bit;
00109 }
00110
00111 /* bit test and set */
00112 #define __L4UTIL_BITOPS_HAVE_ARCH_BIT_TEST_AND_SET
00113 L4_INLINE int
00114 l4util_bts(int b, volatile l4_umword_t * dest)
00115 {
00116 l4_int8_t bit;
00117
00118 __asm__ __volatile__
00119 (
00120 "lock; btsq %2,%1 \n\t"
00121 :
00122 "=@ccc" (bit) /* 0, old bit value */
00123 :
00124 "m" (*dest), /* 1 mem, destination operand */

```

```

00125 "Jr" ((l4_umword_t)b) /* 2, bit number */
00126 :
00127 "memory"
00128);
00129
00130 return bit;
00131 }
00132
00133 /* bit test and reset */
00134 #define __L4UTIL_BITOPS_HAVE_ARCH_BIT_TEST_AND_RESET
00135 L4_INLINE int
00136 l4util_btr(int b, volatile l4_umword_t * dest)
00137 {
00138 l4_int8_t bit;
00139
00140 __asm__ __volatile__
00141 (
00142 "lock; btrq %2,%1 \n\t"
00143 :
00144 "=@ccc" (bit) /* 0, old bit value */
00145 :
00146 "m" (*dest), /* 1 mem, destination operand */
00147 "Jr" ((l4_umword_t)b) /* 2, bit number */
00148 :
00149 "memory"
00150);
00151
00152 return bit;
00153 }
00154
00155 /* bit test and complement */
00156 #define __L4UTIL_BITOPS_HAVE_ARCH_BIT_TEST_AND_COMPLEMENT
00157 L4_INLINE int
00158 l4util_btc(int b, volatile l4_umword_t * dest)
00159 {
00160 l4_int8_t bit;
00161
00162 __asm__ __volatile__
00163 (
00164 "lock; btcq %2,%1 \n\t"
00165 :
00166 "=@ccc" (bit) /* 0, old bit value */
00167 :
00168 "m" (*dest), /* 1 mem, destination operand */
00169 "Jr" ((l4_umword_t)b) /* 2, bit number */
00170 :
00171 "memory"
00172);
00173
00174 return bit;
00175 }
00176
00177 /* bit scan reverse */
00178 #define __L4UTIL_BITOPS_HAVE_ARCH_BIT_SCAN_REVERSE
00179 L4_INLINE int
00180 l4util_bsr(l4_umword_t word)
00181 {
00182 l4_umword_t tmp;
00183
00184 if (L4_UNLIKELY(word == 0))
00185 return -1;
00186
00187 __asm__ __volatile__
00188 (
00189 "bsrq %1,%0 \n\t"
00190 :
00191 "=r" (tmp) /* 0, index of most significant set bit */
00192 :
00193 "r" (word) /* 1, argument */
00194);
00195
00196 return tmp;
00197 }
00198
00199 /* bit scan forward */
00200 #define __L4UTIL_BITOPS_HAVE_ARCH_BIT_SCAN_FORWARD
00201 L4_INLINE int
00202 l4util_bsf(l4_umword_t word)
00203 {
00204 l4_umword_t tmp;
00205
00206 if (L4_UNLIKELY(word == 0))
00207 return -1;
00208
00209 __asm__ __volatile__
00210 (
00211 "bsfq %1,%0 \n\t"

```

```

00212 :
00213 "=r" (tmp) /* 0, index of least significant set bit */
00214 :
00215 "r" (word) /* 1, argument */
00216);
00217
00218 return tmp;
00219 }
00220
00221 #define __L4UTIL_BITOPS_HAVE_ARCH_FIND_FIRST_SET_BIT
00222 L4_INLINE int
00223 l4util_find_first_set_bit(const void * dest, l4_size_t size)
00224 {
00225 l4_mword_t dummy0, dummy1, res;
00226
00227 __asm__ __volatile__
00228 (
00229 "repe; scasq \n\t"
00230 "jz 1f \n\t"
00231 "lea -8(%%rdi),%%rdi \n\t"
00232 "bsf (%%rdi),%%rax \n\t"
00233 "1: \n\t"
00234 "sub %%rbx,%%rdi \n\t"
00235 "shl $3,%%rdi \n\t"
00236 "add %%rdi,%%rax \n\t"
00237 :
00238 "=a" (res), "=c" (dummy0), "=D" (dummy1)
00239 :
00240 "a" (0), "b" (dest), "c" ((size + 63) >> 6), "D" (dest)
00241 :
00242 "cc", "memory");
00243
00244 return res;
00245 }
00246
00247 #define __L4UTIL_BITOPS_HAVE_ARCH_FIND_FIRST_ZERO_BIT
00248 L4_INLINE int
00249 l4util_find_first_zero_bit(const void * dest, l4_size_t size)
00250 {
00251 l4_mword_t dummy0, dummy1, dummy2, res;
00252
00253 if (!size)
00254 return 0;
00255
00256 __asm__ __volatile__
00257 (
00258 "repe; scasq \n\t"
00259 "je 1f \n\t"
00260 "xor -8(%%rdi),%%rax \n\t"
00261 "sub $8,%%rdi \n\t"
00262 "bsf %%rax,%%rdx \n\t"
00263 "1: \n\t"
00264 "sub %%rsi,%%rdi \n\t"
00265 "shl $3,%%rdi \n\t"
00266 "add %%rdi,%%rdx \n\t"
00267 :
00268 "=a" (dummy0), "=c" (dummy1), "=d" (res), "=D" (dummy2)
00269 :
00270 "a" (~0UL), "c" ((size + 63) >> 6), "d" (0), "D" (dest), "S" (dest)
00271 :
00272 "cc", "memory");
00273
00274 return res;
00275 }
00276
00277 L4_END_DECLS

```

## 17.129 arm/l4/util/bitops\_arch.h File Reference

ARM specific implementation of bitops functions.

### 17.129.1 Detailed Description

ARM specific implementation of bitops functions.

Definition in file [bitops\\_arch.h](#).



## 17.130 bitops\_arch.h

[Go to the documentation of this file.](#)

```
00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010 #ifndef __L4UTIL__ARCH_ARM__BITOPS_ARCH_H__
00011 #define __L4UTIL__ARCH_ARM__BITOPS_ARCH_H__
00012
00013
00014 #endif /* ! __L4UTIL__ARCH_ARM__BITOPS_ARCH_H__ */
```

## 17.131 x86/I4/util/bitops\_arch.h File Reference

x86 bit manipulation functions

### 17.131.1 Detailed Description

x86 bit manipulation functions

Definition in file [bitops\\_arch.h](#).

## 17.132 bitops\_arch.h

[Go to the documentation of this file.](#)

```
00001 /*
00002 * Copyright (C) 2000-2009 Technische Universität Dresden (Germany)
00003 * Copyright (C) 2016, 2022, 2024 Kernkonzept GmbH. All rights reserved.
00004 * Author(s): Lars Reuther <reuther@os.inf.tu-dresden.de>
00005 * Frank Mehnert <frank.mehnert@kernkonzept.com>
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009
00015
00016 #pragma once
00017
00018 L4_BEGIN_DECLS
00019
00020 /* set bit */
00021 #define __L4UTIL_BITOPS_HAVE_ARCH_SET_BIT
00022 L4_INLINE void
00023 l4util_set_bit(int b, volatile l4_umword_t * dest)
00024 {
00025 __asm__ __volatile__
00026 (
00027 "lock; btsl %1,%0 \n\t"
00028 :
00029 :
00030 "m" (*dest), /* 0 mem, destination operand */
00031 "Ir" (b) /* 1, bit number */
00032 :
00033 "memory", "cc"
00034);
00035 }
00036
00037 /* clear bit */
00038 #define __L4UTIL_BITOPS_HAVE_ARCH_CLEAR_BIT
00039 L4_INLINE void
00040 l4util_clear_bit(int b, volatile l4_umword_t * dest)
00041 {
00042 __asm__ __volatile__
00043 (
```

```

00044 "lock; btrl %1,%0 \n\t"
00045 :
00046 :
00047 "m" (*dest), /* 0 mem, destination operand */
00048 "Ir" (b) /* 1, bit number */
00049 :
00050 "memory", "cc"
00051);
00052 }
00053
00054 /* change bit */
00055 #define __L4UTIL_BITOPS_HAVE_ARCH_COMPLEMENT_BIT
00056 L4_INLINE void
00057 l4util_complement_bit(int b, volatile l4_umword_t * dest)
00058 {
00059 __asm__ __volatile__
00060 (
00061 "lock; btcl %1,%0 \n\t"
00062 :
00063 :
00064 "m" (*dest), /* 0 mem, destination operand */
00065 "Ir" (b) /* 1, bit number */
00066 :
00067 "memory", "cc"
00068);
00069 }
00070
00071 /* test bit */
00072 #define __L4UTIL_BITOPS_HAVE_ARCH_TEST_BIT
00073 L4_INLINE int
00074 l4util_test_bit(int b, const volatile l4_umword_t * dest)
00075 {
00076 l4_int8_t bit;
00077
00078 __asm__ __volatile__
00079 (
00080 "btl %2,%1 \n\t"
00081 :
00082 "=@ccc" (bit) /* 0, old bit value */
00083 :
00084 "m" (*dest), /* 1 mem, destination operand */
00085 "Ir" (b) /* 2, bit number */
00086 :
00087 "memory"
00088);
00089
00090 return bit;
00091 }
00092
00093 /* bit test and set */
00094 #define __L4UTIL_BITOPS_HAVE_ARCH_BIT_TEST_AND_SET
00095 L4_INLINE int
00096 l4util_bts(int b, volatile l4_umword_t * dest)
00097 {
00098 l4_int8_t bit;
00099
00100 __asm__ __volatile__
00101 (
00102 "lock; btsl %2,%1 \n\t"
00103 :
00104 "=@ccc" (bit) /* 0, old bit value */
00105 :
00106 "m" (*dest), /* 1 mem, destination operand */
00107 "Ir" (b) /* 2, bit number */
00108 :
00109 "memory"
00110);
00111
00112 return bit;
00113 }
00114
00115 /* bit test and reset */
00116 #define __L4UTIL_BITOPS_HAVE_ARCH_BIT_TEST_AND_RESET
00117 L4_INLINE int
00118 l4util_btr(int b, volatile l4_umword_t * dest)
00119 {
00120 l4_int8_t bit;
00121
00122 __asm__ __volatile__
00123 (
00124 "lock; btrl %2,%1 \n\t"
00125 :
00126 "=@ccc" (bit) /* 0, old bit value */
00127 :
00128 "m" (*dest), /* 1 mem, destination operand */
00129 "Ir" (b) /* 2, bit number */
00130 :

```

```

00131 "memory"
00132);
00133
00134 return bit;
00135 }
00136
00137 /* bit test and complement */
00138 #define __L4UTIL_BITOPS_HAVE_ARCH_BIT_TEST_AND_COMPLEMENT
00139 L4_INLINE int
00140 l4util_btc(int b, volatile l4_umword_t * dest)
00141 {
00142 l4_int8_t bit;
00143
00144 __asm__ __volatile__
00145 (
00146 "lock; btl %2,%1 \n\t"
00147 :
00148 "=@ccc" (bit) /* 0, old bit value */
00149 :
00150 "m" (*dest), /* 1 mem, destination operand */
00151 "Ir" (b) /* 2, bit number */
00152 :
00153 "memory"
00154);
00155
00156 return bit;
00157 }
00158
00159 /* bit scan reverse */
00160 #define __L4UTIL_BITOPS_HAVE_ARCH_BIT_SCAN_REVERSE
00161 L4_INLINE int
00162 l4util_bsr(l4_umword_t word)
00163 {
00164 int tmp;
00165
00166 if (L4_UNLIKELY(word == 0))
00167 return -1;
00168
00169 __asm__ __volatile__
00170 (
00171 "bsrl %1,%0 \n\t"
00172 :
00173 "=r" (tmp) /* 0, index of most significant set bit */
00174 :
00175 "r" (word) /* 1, argument */
00176);
00177
00178 return tmp;
00179 }
00180
00181 /* bit scan forward */
00182 #define __L4UTIL_BITOPS_HAVE_ARCH_BIT_SCAN_FORWARD
00183 L4_INLINE int
00184 l4util_bsf(l4_umword_t word)
00185 {
00186 int tmp;
00187
00188 if (L4_UNLIKELY(word == 0))
00189 return -1;
00190
00191 __asm__ __volatile__
00192 (
00193 "bsfl %1,%0 \n\t"
00194 :
00195 "=r" (tmp) /* 0, index of least significant set bit */
00196 :
00197 "r" (word) /* 1, argument */
00198);
00199
00200 return tmp;
00201 }
00202
00203 #define __L4UTIL_BITOPS_HAVE_ARCH_FIND_FIRST_SET_BIT
00204 L4_INLINE int
00205 l4util_find_first_set_bit(const void * dest, l4_size_t size)
00206 {
00207 l4_mword_t dummy0, dummy1, res;
00208
00209 __asm__ __volatile__
00210 (
00211 "repe; scasl \n\t"
00212 "jz 1f \n\t"
00213 "lea -4(%edi),%edi \n\t"
00214 "bsf (%edi),%eax \n\t"
00215 "1: \n\t"
00216 "sub %esi,%edi \n\t"
00217 "shl $3,%edi \n\t"

```

```

00218 "add %%edi,%%eax \n\t"
00219 :
00220 "a" (res), "c" (dummy0), "=D" (dummy1)
00221 :
00222 "a" (0), "c" ((size + 31) >> 5), "D" (dest), "S" (dest)
00223 :
00224 "cc", "memory");
00225
00226 return res;
00227 }
00228
00229 #define __L4UTIL_BITOPS_HAVE_ARCH_FIND_FIRST_ZERO_BIT
00230 L4_INLINE int
00231 l4util_find_first_zero_bit(const void * dest, l4_size_t size)
00232 {
00233 l4_mword_t dummy0, dummy1, dummy2, res;
00234
00235 if (!size)
00236 return 0;
00237
00238 __asm__ __volatile__
00239 (
00240 "repe; scasl \n\t"
00241 "je lf \n\t"
00242 "xor -4(%%edi),%%eax \n\t"
00243 "sub $4,%%edi \n\t"
00244 "bsf %%eax,%%edx \n\t"
00245 "l: \n\t"
00246 "sub %%esi,%%edi \n\t"
00247 "shl $3,%%edi \n\t"
00248 "add %%edi,%%edx \n\t"
00249 :
00250 "a" (dummy0), "c" (dummy1), "=d" (res), "=D" (dummy2)
00251 :
00252 "a" (~0UL), "c" ((size + 31) >> 5), "d" (0), "D" (dest), "S" (dest)
00253 :
00254 "cc", "memory");
00255
00256 return res;
00257 }
00258
00259 L4_END_DECLS

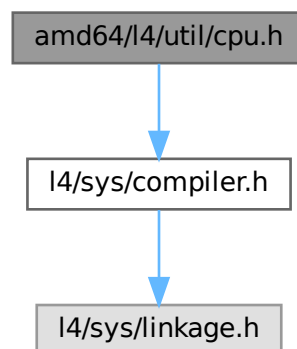
```

## 17.133 amd64/l4/util/cpu.h File Reference

CPU related functions.

```
#include <l4/sys/compiler.h>
```

Include dependency graph for cpu.h:



## Functions

- int [l4util\\_cpu\\_has\\_cpuid](#) (void)  
*Check whether the CPU supports the "cpuid" instruction.*
- unsigned int [l4util\\_cpu\\_capabilities](#) (void)  
*Returns the CPU capabilities if the "cpuid" instruction is available.*
- unsigned int [l4util\\_cpu\\_capabilities\\_nocheck](#) (void)  
*Returns the CPU capabilities.*
- void [l4util\\_cpu\\_cpuid](#) (unsigned long mode, unsigned long \*eax, unsigned long \*ebx, unsigned long \*ecx, unsigned long \*edx)  
*Generic CPUID access function.*

### 17.133.1 Detailed Description

CPU related functions.

#### Author

Frank Mehnert [fm3@os.inf.tu-dresden.de](mailto:fm3@os.inf.tu-dresden.de)

Definition in file [cpu.h](#).

## 17.134 cpu.h

[Go to the documentation of this file.](#)

```

00001
00006
00007 /*
00008 * (c) 2004-2009 Author(s)
00009 * economic rights: Technische Universität Dresden (Germany)
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012
00013 #ifndef __L4_UTIL_CPU_H
00014 #define __L4_UTIL_CPU_H
00015
00016 #include <l4/sys/compiler.h>
00017
00018 L4_BEGIN_DECLS
00019
00025
00031 L4_INLINE int l4util_cpu_has_cpuid(void);
00032
00039 L4_INLINE unsigned int l4util_cpu_capabilities(void);
00040
00046 L4_INLINE unsigned int l4util_cpu_capabilities_nocheck(void);
00047
00051 L4_INLINE void
00052 l4util_cpu_cpuid(unsigned long mode,
00053 unsigned long *eax, unsigned long *ebx,
00054 unsigned long *ecx, unsigned long *edx);
00055
00057 static inline void
00058 l4util_cpu_pause(void)
00059 {
00060 __asm__ __volatile__ ("rep; nop");
00061 }
00062
00063 L4_INLINE int
00064 l4util_cpu_has_cpuid(void)
00065 {
00066 return 1;
00067 }
00068
00069 L4_INLINE void

```

```

00070 l4util_cpu_cpuid(unsigned long mode,
00071 unsigned long *eax, unsigned long *ebx,
00072 unsigned long *ecx, unsigned long *edx)
00073 {
00074 asm volatile("cpuid"
00075 : "=a" (*eax),
00076 "=b" (*ebx),
00077 "=c" (*ecx),
00078 "=d" (*edx)
00079 : "a" (mode)
00080);
00081 }
00082
00083 L4_INLINE unsigned int
00084 l4util_cpu_capabilities_nocheck(void)
00085 {
00086 unsigned long dummy, capability;
00087
00088 /* get CPU capabilities */
00089 l4util_cpu_cpuid(1, &dummy, &dummy, &dummy, &capability);
00090
00091 return capability;
00092 }
00093
00094 L4_INLINE unsigned int
00095 l4util_cpu_capabilities(void)
00096 {
00097 if (!l4util_cpu_has_cpuid())
00098 return 0; /* CPU has not cpuid instruction */
00099
00100 return l4util_cpu_capabilities_nocheck();
00101 }
00102
00103 L4_END_DECLS
00104
00105 #endif
00106

```

## 17.135 arm/l4/util/cpu.h File Reference

CPU related functions.

### 17.135.1 Detailed Description

CPU related functions.

Author

Adam Lackorzynski [adam@os.inf.tu-dresden.de](mailto:adam@os.inf.tu-dresden.de)

Definition in file [cpu.h](#).

## 17.136 cpu.h

[Go to the documentation of this file.](#)

```

00001
00002
00003 /*
00004 * (c) 2004-2009 Author(s)
00005 * economic rights: Technische Universität Dresden (Germany)
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00009 #ifndef __L4_UTIL__ARCH_ARM__CPU_H__
00010 #define __L4_UTIL__ARCH_ARM__CPU_H__
00011
00012 /* Nothing yet */
00013
00014 #endif /* __L4_UTIL__ARCH_ARM__CPU_H__ */

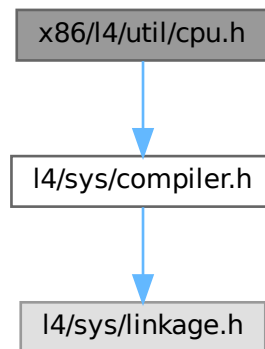
```

## 17.137 x86/l4/util/cpu.h File Reference

CPU related functions.

```
#include <l4/sys/compiler.h>
```

Include dependency graph for cpu.h:



### Functions

- int [l4util\\_cpu\\_has\\_cpuid](#) (void)  
*Check whether the CPU supports the "cpuid" instruction.*
- unsigned int [l4util\\_cpu\\_capabilities](#) (void)  
*Returns the CPU capabilities if the "cpuid" instruction is available.*
- unsigned int [l4util\\_cpu\\_capabilities\\_nocheck](#) (void)  
*Returns the CPU capabilities.*
- void [l4util\\_cpu\\_cpuid](#) (unsigned long mode, unsigned long \*eax, unsigned long \*ebx, unsigned long \*ecx, unsigned long \*edx)  
*Generic CPUID access function.*

### 17.137.1 Detailed Description

CPU related functions.

#### Author

Frank Mehnert [fm3@os.inf.tu-dresden.de](mailto:fm3@os.inf.tu-dresden.de)

Definition in file [cpu.h](#).

## 17.138 cpu.h

[Go to the documentation of this file.](#)

```

00001
00006
00007 /*
00008 * (c) 2004-2009 Author(s)
00009 * economic rights: Technische Universität Dresden (Germany)
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012
00013 #ifndef __L4_UTIL_CPU_H
00014 #define __L4_UTIL_CPU_H
00015
00016 #include <l4/sys/compiler.h>
00017
00018 L4_BEGIN_DECLS
00019
00025
00031 L4_INLINE int l4util_cpu_has_cpuid(void);
00032
00039 L4_INLINE unsigned int l4util_cpu_capabilities(void);
00040
00046 L4_INLINE unsigned int l4util_cpu_capabilities_nocheck(void);
00047
00051 L4_INLINE void
00052 l4util_cpu_cpuid(unsigned long mode,
00053 unsigned long *eax, unsigned long *ebx,
00054 unsigned long *ecx, unsigned long *edx);
00055
00057 static inline void
00058 l4util_cpu_pause(void)
00059 {
00060 __asm__ __volatile__ ("rep; nop");
00061 }
00062
00063 L4_INLINE int
00064 l4util_cpu_has_cpuid(void)
00065 {
00066 unsigned long eax;
00067
00068 asm volatile("pushl %%ebx \t\n"
00069 "pushfl \t\n"
00070 "popl %%eax \t\n" /* get eflags */
00071 "movl %%eax, %%ebx \t\n" /* save it */
00072 "xorl $0x200000, %%eax \t\n" /* toggle ID bit */
00073 "pushl %%eax \t\n"
00074 "popfl \t\n" /* set again */
00075 "pushfl \t\n"
00076 "popl %%eax \t\n" /* get it again */
00077 "xorl %%ebx, %%eax \t\n"
00078 "pushl %%ebx \t\n"
00079 "popfl \t\n" /* restore saved flags */
00080 "popl %%ebx \t\n"
00081 : "=a" (eax)
00082 : /* no input */
00083);
00084
00085 return eax & 0x200000;
00086 }
00087
00088 L4_INLINE void
00089 l4util_cpu_cpuid(unsigned long mode,
00090 unsigned long *eax, unsigned long *ebx,
00091 unsigned long *ecx, unsigned long *edx)
00092 {
00093 asm volatile("pushl %%ebx \t\n"
00094 "cpuid \t\n"
00095 "movl %%ebx, %%esi \t\n"
00096 "popl %%ebx \t\n"
00097 : "=a" (*eax),
00098 "=S" (*ebx),
00099 "=c" (*ecx),
00100 "=d" (*edx)
00101 : "a" (mode));
00102 }
00103
00104 L4_INLINE unsigned int
00105 l4util_cpu_capabilities_nocheck(void)
00106 {
00107 unsigned long dummy, capability;
00108
00109 /* get CPU capabilities */
00110 l4util_cpu_cpuid(1, &dummy, &dummy, &dummy, &capability);
00111

```



```

00112 return capability;
00113 }
00114
00115 L4_INLINE unsigned int
00116 l4util_cpu_capabilities(void)
00117 {
00118 if (!l4util_cpu_has_cpuid())
00119 return 0; /* CPU has not cpuid instruction */
00120
00121 return l4util_cpu_capabilities_nocheck();
00122 }
00123
00124 L4_END_DECLS
00125
00126 #endif
00127

```

## 17.139 amd64/l4/util/l4\_macros.h File Reference

Main function.

### 17.139.1 Detailed Description

Main function.

#### Date

08/29/2000

#### Author

Frank Mehnert [fm3@os.inf.tu-dresden.de](mailto:fm3@os.inf.tu-dresden.de)

Definition in file [l4\\_macros.h](#).

## 17.140 l4\_macros.h

[Go to the documentation of this file.](#)

```

00001
00007
00008 /*
00009 * (c) 2006-2009 Author(s)
00010 * economic rights: Technische Universität Dresden (Germany)
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013
00014 #ifndef _L4UTIL__ARCH_AMD64__L4_MACROS_H
00015 #define _L4UTIL__ARCH_AMD64__L4_MACROS_H
00016
00017 #include_next <l4/util/l4_macros.h>
00018
00019 #ifndef l4_addr_fmt
00020 # define l4_addr_fmt "%016lx"
00021 #endif
00022
00023 #endif /* !_L4UTIL__ARCH_AMD64__L4_MACROS_H */

```

## 17.141 arm/l4/util/l4\_macros.h File Reference

Main function.

### 17.141.1 Detailed Description

Main function.

Date

08/29/2000

Author

Frank Mehnert [fm3@os.inf.tu-dresden.de](mailto:fm3@os.inf.tu-dresden.de)

Definition in file [l4\\_macros.h](#).

## 17.142 l4\_macros.h

[Go to the documentation of this file.](#)

```
00001
00007
00008 /*
00009 * (c) 2006-2009 Author(s)
00010 * economic rights: Technische Universität Dresden (Germany)
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013
00014 #ifndef _L4UTIL__ARCH_ARM__L4_MACROS_H
00015 #define _L4UTIL__ARCH_ARM__L4_MACROS_H
00016
00017 #include_next <l4/util/l4_macros.h>
00018
00019 #ifndef l4_addr_fmt
00020 # define l4_addr_fmt "%08lx"
00021 #endif
00022
00023 #endif /* !_L4UTIL__ARCH_ARM__L4_MACROS_H */
```

## 17.143 l4/util/l4\_macros.h File Reference

Some useful generic macros, L4f version.

### 17.143.1 Detailed Description

Some useful generic macros, L4f version.

Date

11/12/2002

Author

Lars Reuther [reuther@os.inf.tu-dresden.de](mailto:reuther@os.inf.tu-dresden.de)

Definition in file [l4\\_macros.h](#).

## 17.144 l4\_macros.h

[Go to the documentation of this file.](#)

```

00001 /*****
00008 */
00009 * (c) 2000-2009 Author(s)
00010 * economic rights: Technische Universität Dresden (Germany)
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013
00014 /*****
00015
00016 #pragma once
00017
00018 /*****
00019 *** generic macros
00020 *****/
00021
00022 /* generate L4 thread id printf string */
00023 #ifndef l4util_idstr
00024 # define l4util_idfmt "%lx"
00025 # define l4util_idfmt_adjust "%04lx"
00026 # define l4util_idstr(tid) (tid » L4_CAP_SHIFT)
00027 #endif
00028

```

## 17.145 x86/l4/util/l4\_macros.h File Reference

Main function.

### 17.145.1 Detailed Description

Main function.

#### Date

08/29/2000

#### Author

Frank Mehnert [fm3@os.inf.tu-dresden.de](mailto:fm3@os.inf.tu-dresden.de)

Definition in file [l4\\_macros.h](#).

## 17.146 l4\_macros.h

[Go to the documentation of this file.](#)

```

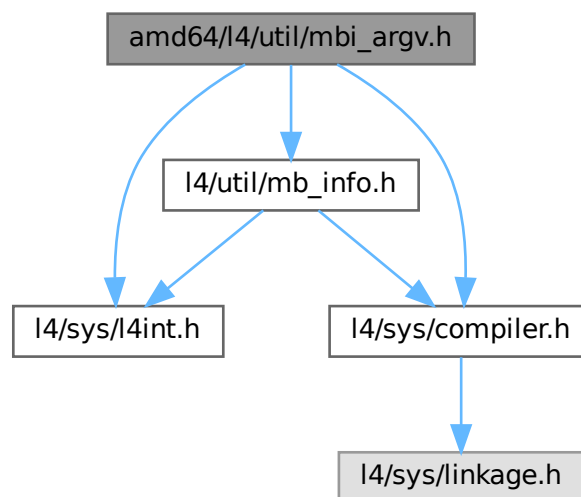
00001
00007
00008 /*
00009 * (c) 2006-2009 Author(s)
00010 * economic rights: Technische Universität Dresden (Germany)
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013
00014
00015 #ifndef _L4UTIL__ARCH_X86__L4_MACROS_H
00016 #define _L4UTIL__ARCH_X86__L4_MACROS_H
00017
00018 #include_next <l4/util/l4_macros.h>
00019
00020 #ifndef l4_addr_fmt
00021 # define l4_addr_fmt "%08lx"
00022 #endif
00023
00024 #endif /* !_L4UTIL__ARCH_X86__L4_MACROS_H */

```

## 17.147 amd64/l4/util/mbi\_argv.h File Reference

command line handling

```
#include <l4/sys/l4int.h>
#include <l4/util/mb_info.h>
#include <l4/sys/compiler.h>
Include dependency graph for mbi_argv.h:
```



### 17.147.1 Detailed Description

command line handling

Date

2003

Author

Frank Mehnert [fm3@os.inf.tu-dresden.de](mailto:fm3@os.inf.tu-dresden.de)

Definition in file [mbi\\_argv.h](#).

## 17.148 mbi\_argv.h

[Go to the documentation of this file.](#)

```

00001
00007
00008 /*
00009 * (c) 2003-2009 Author(s)
00010 * economic rights: Technische Universität Dresden (Germany)
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013
00014 #ifndef L4UTIL_MBI_ARGV
00015 #define L4UTIL_MBI_ARGV
00016
00017 #include <l4/sys/l4int.h>
00018 #include <l4/util/mb_info.h>
00019 #include <l4/sys/compiler.h>
00020
00021 L4_BEGIN_DECLS
00022
00023 L4_CV void l4util_mbi_to_argv(l4_mword_t flag, l4util_mb_info_t *mbi);
00024
00025 extern int l4util_argc;
00026 extern char *l4util_argv[];
00027
00028 L4_END_DECLS
00029
00030 #endif
00031

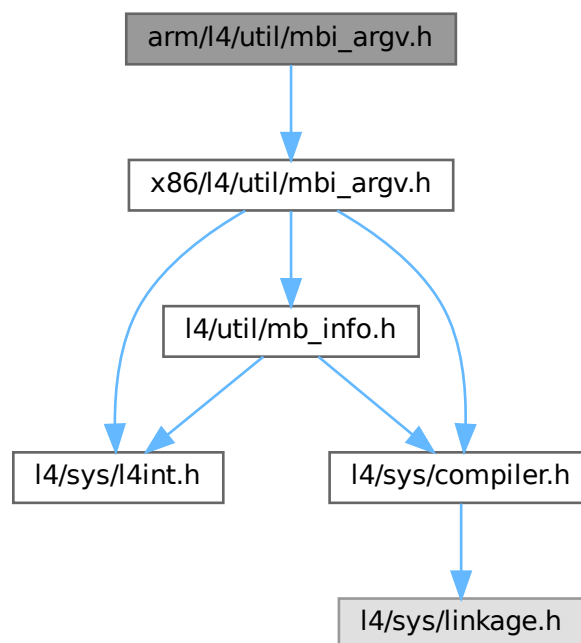
```

## 17.149 arm/l4/util/mbi\_argv.h File Reference

Multiboot.

```
#include <x86/l4/util/mbi_argv.h>
```

Include dependency graph for mbi\_argv.h:



### 17.149.1 Detailed Description

Multiboot.

Definition in file [mbi\\_argv.h](#).

## 17.150 mbi\_argv.h

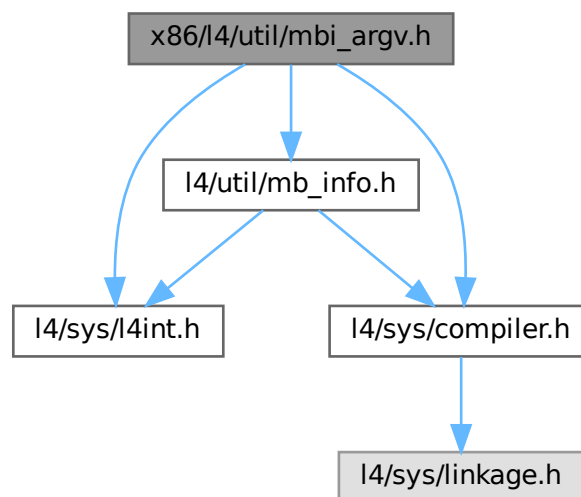
[Go to the documentation of this file.](#)

```
00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010 /* If this persists, move it to a generic place */
00011 #include <x86/l4/util/mbi_argv.h>
```

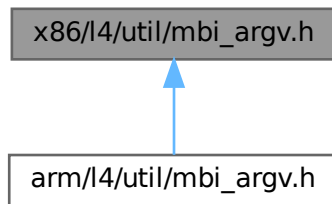
## 17.151 x86/l4/util/mbi\_argv.h File Reference

command line handling

```
#include <l4/sys/l4int.h>
#include <l4/util/mb_info.h>
#include <l4/sys/compiler.h>
Include dependency graph for mbi_argv.h:
```



This graph shows which files directly or indirectly include this file:



### 17.151.1 Detailed Description

command line handling

#### Date

2003

#### Author

Frank Mehnert [fm3@os.inf.tu-dresden.de](mailto:fm3@os.inf.tu-dresden.de)

Definition in file [mbi\\_argv.h](#).

## 17.152 mbi\_argv.h

[Go to the documentation of this file.](#)

```

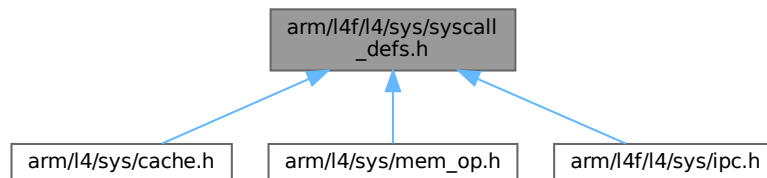
00001
00007
00008 /*
00009 * (c) 2003-2009 Author(s)
00010 * economic rights: Technische Universität Dresden (Germany)
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013
00014 #ifndef L4UTIL_MBI_ARGV
00015 #define L4UTIL_MBI_ARGV
00016
00017 #include <l4/sys/l4int.h>
00018 #include <l4/util/mb_info.h>
00019 #include <l4/sys/compiler.h>
00020
00021 L4_BEGIN_DECLS
00022
00023 void l4util_mbi_to_argv(l4_mword_t flag, l4util_mb_info_t *mbi);
00024
00025 extern int l4util_argc;
00026 extern char *l4util_argv[];
00027
00028 L4_END_DECLS
00029
00030 #endif
00031

```

## 17.153 arm/l4f/l4/sys/syscall\_defs.h File Reference

Syscall entry definitions.

This graph shows which files directly or indirectly include this file:



### 17.153.1 Detailed Description

Syscall entry definitions.

Definition in file [syscall\\_defs.h](#).

## 17.154 syscall\_defs.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2010 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #ifndef __L4SYS__ARCH_ARM__L4API_L4F__SYSCALLL_DEFS_H__
00012 #define __L4SYS__ARCH_ARM__L4API_L4F__SYSCALLL_DEFS_H__
00013
00014 #define L4_SYSCALLL_INVOKE (0)
00015 #define L4_SYSCALLL_MEM_OP (1)
00016
00017 #endif /* __L4SYS__ARCH_ARM__L4API_L4F__SYSCALLL_DEFS_H__ */

```

## 17.155 contrib/libio-io/l4/io/io.h File Reference

```

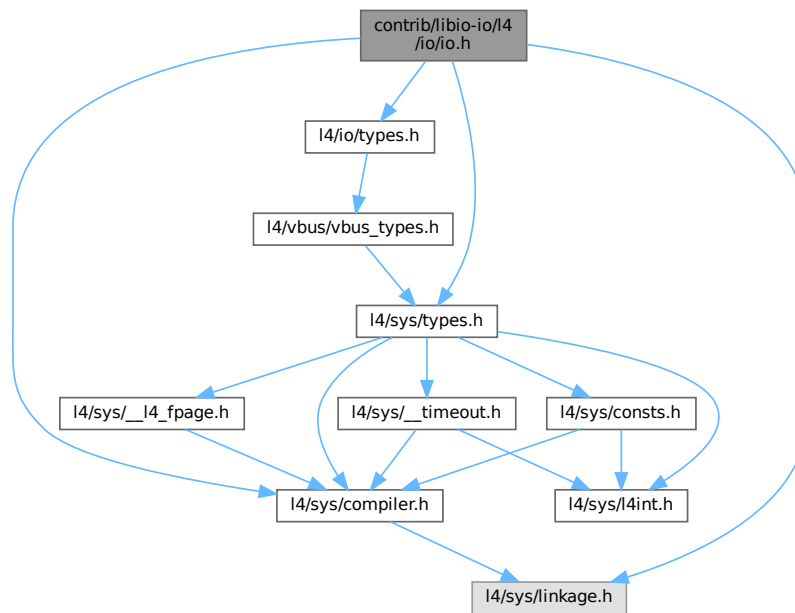
#include <l4/sys/compiler.h>
#include <l4/sys/types.h>
#include <l4/sys/linkage.h>

```



```
#include <l4/io/types.h>
```

Include dependency graph for io.h:



## Functions

- [l4\\_cap\\_idx\\_t l4io\\_request\\_icu](#) (void)  
*Request the ICU object of the client.*
- long [l4io\\_request\\_iomem](#) ([l4\\_addr\\_t](#) phys, unsigned long size, int flags, [l4\\_addr\\_t](#) \*virt)  
*Request an IO memory region.*
- long [l4io\\_request\\_iomem\\_region](#) ([l4\\_addr\\_t](#) phys, [l4\\_addr\\_t](#) virt, unsigned long size, int flags)  
*Request an IO memory region and map it to a specified region.*
- long [l4io\\_release\\_iomem](#) ([l4\\_addr\\_t](#) virt, unsigned long size)  
*Release an IO memory region.*
- long [l4io\\_request\\_ioport](#) (unsigned portnum, unsigned len)  
*Request an IO port region.*
- long [l4io\\_release\\_ioport](#) (unsigned portnum, unsigned len)  
*Release an IO port region.*
- [l4io\\_device\\_handle\\_t l4io\\_get\\_root\\_device](#) (void)  
*Get root device handle of the device bus.*
- int [l4io\\_iterate\\_devices](#) ([l4io\\_device\\_handle\\_t](#) \*devhandle, [l4io\\_device\\_t](#) \*dev, [l4io\\_resource\\_handle\\_t](#) \*reshandle)  
*Iterate over the device bus.*
- int [l4io\\_lookup\\_device](#) (const char \*devname, [l4io\\_device\\_handle\\_t](#) \*dev\_handle, [l4io\\_device\\_t](#) \*dev, [l4io\\_resource\\_handle\\_t](#) \*res\_handle)  
*Find a device by name.*
- int [l4io\\_lookup\\_resource](#) ([l4io\\_device\\_handle\\_t](#) devhandle, enum [l4io\\_resource\\_types\\_t](#) type, [l4io\\_resource\\_handle\\_t](#) \*reshandle, [l4io\\_resource\\_t](#) \*res)  
*Request a specific resource from a device description.*

- [l4\\_addr\\_t l4io\\_request\\_resource\\_iomem](#) ([l4io\\_device\\_handle\\_t](#) devhandle, [l4io\\_resource\\_handle\\_t](#) \*reshandle)  
*Request IO memory.*
- void [l4io\\_request\\_all\\_ioports](#) (void(\*res\_cb)([l4vbus\\_resource\\_t](#) const \*res))  
*Request all available IO-port resources.*
- int [l4io\\_has\\_resource](#) (enum [l4io\\_resource\\_types\\_t](#) type, [l4vbus\\_paddr\\_t](#) start, [l4vbus\\_paddr\\_t](#) end)  
*Check if a resource is available.*

## 17.155.1 Function Documentation

### 17.155.1.1 l4io\_get\_root\_device()

```
l4io_device_handle_t l4io_get_root_device (
 void) [inline]
```

Get root device handle of the device bus.

#### Returns

root device handle

Definition at line 257 of file [io.h](#).

### 17.155.1.2 l4io\_iterate\_devices()

```
int l4io_iterate_devices (
 l4io_device_handle_t * devhandle,
 l4io_device_t * dev,
 l4io_resource_handle_t * reshandle)
```

Iterate over the device bus.

#### Parameters

|         |                  |                                                                               |
|---------|------------------|-------------------------------------------------------------------------------|
| in, out | <i>devhandle</i> | Device handle to start iterating at. The next device handle is returned here. |
| out     | <i>dev</i>       | Device information, may be NULL.                                              |
| out     | <i>reshandle</i> | Resource handle.                                                              |

#### Returns

0 on success, error code otherwise

References [L4\\_CV](#), and [L4\\_EXPORT](#).

### 17.155.1.3 l4io\_request\_all\_ioports()

```
void l4io_request_all_ioports (
 void(* res_cb)(l4vbus_resource_t const *res))
```

Request all available IO-port resources.

#### Parameters

|                     |                                                                                    |
|---------------------|------------------------------------------------------------------------------------|
| <code>res_cb</code> | Callback function called for every port resource found, give NULL for no callback. |
|---------------------|------------------------------------------------------------------------------------|

References [L4\\_CV](#), and [L4\\_EXPORT](#).

#### 17.155.1.4 l4io\_request\_icu()

```
l4_cap_idx_t l4io_request_icu (
 void)
```

Request the ICU object of the client.

##### Returns

Client ICU object, an invalid capability selector is returned if no ICU is available.

References [L4\\_CV](#), and [L4\\_EXPORT](#).

## 17.156 io.h

[Go to the documentation of this file.](#)

```
00001
00004 /*
00005 * (c) 2008-2010 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00006 * Alexander Warg <warg@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010
00011
00012 #pragma once
00013
00014 #include <l4/sys/compiler.h>
00015 #include <l4/sys/types.h>
00016 #include <l4/sys/linkage.h>
00017 #include <l4/io/types.h>
00018
00019 L4_BEGIN_DECLS
00020
00024
00037 L4_CV long L4_EXPORT
00038 l4io_request_irq(int irqnum, l4_cap_idx_t irqcap);
00039
00046 L4_CV l4_cap_idx_t L4_EXPORT
00047 l4io_request_icu(void);
00048
00060 L4_CV long L4_EXPORT
00061 l4io_release_irq(int irqnum, l4_cap_idx_t irq_cap);
00062
00087 L4_CV long L4_EXPORT
00088 l4io_request_iomem(l4_addr_t phys, unsigned long size, int flags,
00089 l4_addr_t *virt);
00090
00112 L4_CV long L4_EXPORT
00113 l4io_request_iomem_region(l4_addr_t phys, l4_addr_t virt,
00114 unsigned long size, int flags);
00115
00123 L4_CV long L4_EXPORT
00124 l4io_release_iomem(l4_addr_t virt, unsigned long size);
00125
00135 L4_CV long L4_EXPORT
00136 l4io_request_ioport(unsigned portnum, unsigned len);
00137
00147 L4_CV long L4_EXPORT
00148 l4io_release_ioport(unsigned portnum, unsigned len);
00149
00150
00151 /* ----- Device handling ----- */
```

```

00152
00158 L4_INLINE
00159 l4io_device_handle_t l4io_get_root_device(void);
00160
00171 L4_CV int L4_EXPORT
00172 l4io_iterate_devices(l4io_device_handle_t *devhandle,
00173 l4io_device_t *dev, l4io_resource_handle_t *reshandle);
00174
00187 L4_CV int L4_EXPORT
00188 l4io_lookup_device(const char *devname,
00189 l4io_device_handle_t *dev_handle,
00190 l4io_device_t *dev, l4io_resource_handle_t *res_handle);
00191
00207 L4_CV int L4_EXPORT
00208 l4io_lookup_resource(l4io_device_handle_t devhandle,
00209 enum l4io_resource_types_t type,
00210 l4io_resource_handle_t *reshandle,
00211 l4io_resource_t *res);
00212
00213
00214 /* ----- Convenience functions ----- */
00215
00228 L4_CV l4_addr_t L4_EXPORT
00229 l4io_request_resource_iomem(l4io_device_handle_t devhandle,
00230 l4io_resource_handle_t *reshandle);
00231
00238 L4_CV void L4_EXPORT
00239 l4io_request_all_ioports(void (*res_cb)(l4vbus_resource_t const *res));
00240
00249 L4_CV int L4_EXPORT
00250 l4io_has_resource(enum l4io_resource_types_t type,
00251 l4vbus_paddr_t start, l4vbus_paddr_t end);
00252
00253 /* ----- */
00254 /* Implementations */
00255
00256 L4_INLINE
00257 l4io_device_handle_t l4io_get_root_device(void)
00258 { return 0; }
00259
00260 L4_END_DECLS

```

## 17.157 l4/cxx/alloc.h File Reference

Alloc list.

### Data Structures

- class [L4::Alloc\\_list](#)  
*A simple list-based allocator.*

### Namespaces

- namespace [L4](#)  
*L4 low-level kernel interface.*

### 17.157.1 Detailed Description

Alloc list.

Definition in file [alloc.h](#).

## 17.158 alloc.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2004-2009 Alexander Warg <warg@os.inf.tu-dresden.de>,
00007 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00014 namespace L4 {
00015
00020 class Alloc_list
00021 {
00022 public:
00023 Alloc_list() : _free(0) {}
00024 Alloc_list(void *blk, unsigned long size) : _free(0)
00025 { free(blk, size); }
00026
00027 void free(void *blk, unsigned long size);
00028 void *alloc(unsigned long size);
00029
00030 private:
00031 struct Elem
00032 {
00033 Elem *next;
00034 unsigned long size;
00035 };
00036
00037 Elem *_free;
00038 };
00039 }

```

## 17.159 arith

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00009 #pragma once
00010
00011 namespace cxx { namespace arith {
00012
00027 template<typename N, typename D>
00028 constexpr N
00029 div_ceil(N const &n, D const &d)
00030 {
00031 // Since C++11 the "quotient is truncated towards zero (fractional part is
00032 // discarded)". Thus a negative quotient is already ceiled, whereas a
00033 // positive quotient is floored. Furthermore, since C++11 the sign of the
00034 // % operator is no longer implementation defined, thus we can use n % d to
00035 // detect if the quotient is positive (n % d >= 0) and was truncated (n % d !=
00036 // 0). In that case, we add one to round to the next largest integer.
00037 return n / d + (n % d > 0);
00038 }
00039
00047 template< unsigned long V >
00048 struct Ld
00049 {
00050 enum { value = Ld<V / 2>::value + 1 };
00051 };
00052
00053 template<>
00054 struct Ld<0>
00055 {
00056 enum { value = ~0UL };
00057 };
00058
00059 template<>
00060 struct Ld<1>
00061 {
00062 enum { value = 0 };
00063 };

```

```

00064
00072 constexpr unsigned
00073 log2u(unsigned val)
00074 {
00075 return 8 * sizeof(val) - __builtin_clz(val) - 1;
00076 }
00077
00079 constexpr unsigned
00080 log2u(unsigned long val)
00081 {
00082 return 8 * sizeof(val) - __builtin_clzl(val) - 1;
00083 }
00084
00086 constexpr unsigned
00087 log2u(unsigned long long val)
00088 {
00089 return 8 * sizeof(val) - __builtin_clzll(val) - 1;
00090 }
00091
00100 template<typename T>
00101 constexpr unsigned
00102 log2u_ceil(T val)
00103 {
00104 return val == 1 ? 0 : log2u(val - 1) + 1;
00105 }
00106
00107 }

```

## 17.160 l4/cxx/avl\_map File Reference

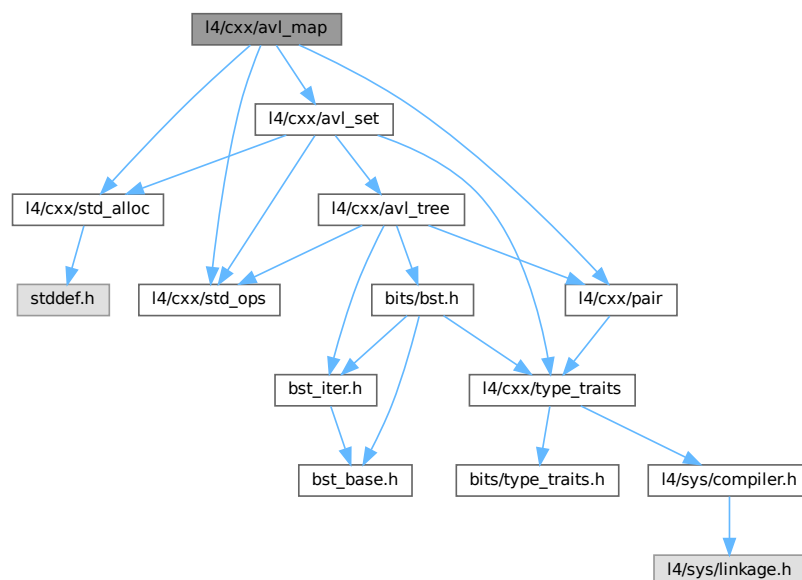
AVL map.

```

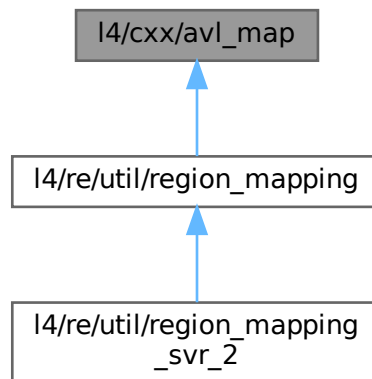
#include <l4/cxx/std_alloc>
#include <l4/cxx/std_ops>
#include <l4/cxx/pair>
#include <l4/cxx/avl_set>

```

Include dependency graph for avl\_map:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct `cxx::Bits::Avl_map_get_key< KEY_TYPE >`  
*Key-getter for `Avl_map`.*
- class `cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC >`  
*AVL tree based associative container.*

## Namespaces

- namespace `cxx`  
*Our C++ library.*
- namespace `cxx::Bits`  
*Internal helpers for the `cxx` package.*

## 17.160.1 Detailed Description

AVL map.

Definition in file [avl\\_map](#).

## 17.161 avl\_map

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */

```

```

00012
00013 #pragma once
00014
00015 #include <l4/cxx/std_alloc>
00016 #include <l4/cxx/std_ops>
00017 #include <l4/cxx/pair>
00018 #include <l4/cxx/avl_set>
00019
00020 namespace cxx {
00021 namespace Bits {
00022
00024 template<typename KEY_TYPE>
00025 struct Avl_map_get_key
00026 {
00027 typedef KEY_TYPE Key_type;
00028 template<typename NODE>
00029 static Key_type const &key_of(NODE const *n)
00030 { return n->item.first; }
00031 };
00032 }
00033
00042 template< typename KEY_TYPE, typename DATA_TYPE,
00043 template<typename A> class COMPARE = Lt_functor,
00044 template<typename B> class ALLOC = New_allocator >
00045 class Avl_map :
00046 public Bits::Base_avl_set<Pair<KEY_TYPE, DATA_TYPE>,
00047 COMPARE<KEY_TYPE>, ALLOC,
00048 Bits::Avl_map_get_key<KEY_TYPE> >
00049 {
00050 private:
00051 typedef Pair<KEY_TYPE, DATA_TYPE> Local_item_type;
00052 typedef Bits::Base_avl_set<Local_item_type, COMPARE<KEY_TYPE>, ALLOC,
00053 Bits::Avl_map_get_key<KEY_TYPE> > Base_type;
00054
00055 public:
00057 typedef COMPARE<KEY_TYPE> Key_compare;
00059 typedef KEY_TYPE Key_type;
00061 typedef DATA_TYPE Data_type;
00063 typedef typename Base_type::Node Node;
00065 typedef typename Base_type::Node_allocator Node_allocator;
00066
00067 typedef typename Base_type::Iterator Iterator;
00068 typedef typename Base_type::Iterator iterator;
00069 typedef typename Base_type::Const_iterator Const_iterator;
00070 typedef typename Base_type::Const_iterator const_iterator;
00071 typedef typename Base_type::Rev_iterator Rev_iterator;
00072 typedef typename Base_type::Rev_iterator reverse_iterator;
00073 typedef typename Base_type::Const_rev_iterator Const_rev_iterator;
00074 typedef typename Base_type::Const_rev_iterator const_reverse_iterator;
00075
00080 Avl_map(Node_allocator const &alloc = Node_allocator())
00081 : Base_type(alloc)
00082 {}
00083
00099 cxx::Pair<Iterator, int> insert(Key_type const &key, Data_type const &data)
00100 { return Base_type::insert(Pair<Key_type, Data_type>(key, data)); }
00101
00102 template<typename... Args>
00103 cxx::Pair<Iterator, int> emplace(Args &&...args)
00104 { return Base_type::emplace(cxx::Forward<Args>(args)...); }
00105
00111 Data_type const &operator [] (Key_type const &key) const
00112 { return this->find_node(key)->second; }
00113
00123 Data_type &operator [] (Key_type const &key)
00124 {
00125 Node n = this->find_node(key);
00126 if (n)
00127 return const_cast<Data_type&>(n->second);
00128 else
00129 return insert(key, Data_type()).first->second;
00130 }
00131 };
00132
00133 }
00134

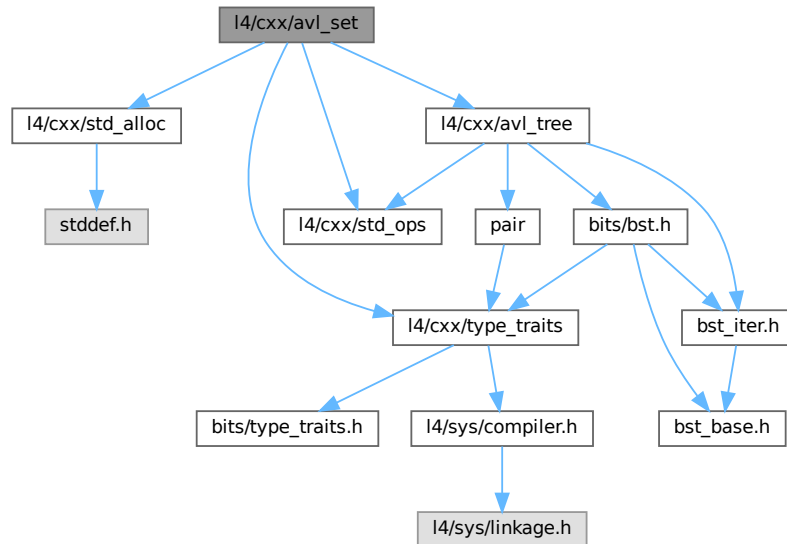
```

## 17.162 l4/cxx/avl\_set File Reference

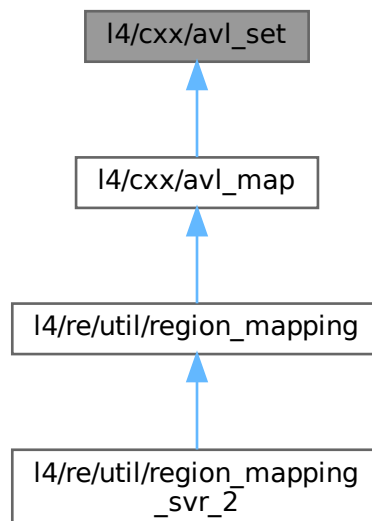
AVL set.



```
#include <l4/cxx/std_alloc>
#include <l4/cxx/std_ops>
#include <l4/cxx/type_traits>
#include <l4/cxx/avl_tree>
Include dependency graph for avl_set:
```



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct `cxx::Bits::Avl_set_get_key< KEY_TYPE >`  
*Internal, key-getter for `Avl_set` nodes.*
- class `cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >`  
*Internal: AVL set with internally managed nodes.*
- class `cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Node`  
*A smart pointer to a tree item.*
- class `cxx::Avl_set< ITEM_TYPE, COMPARE, ALLOC >`  
*AVL set for simple comparable items.*

## Namespaces

- namespace `cxx`  
*Our C++ library.*
- namespace `cxx::Bits`  
*Internal helpers for the `cxx` package.*

### 17.162.1 Detailed Description

AVL set.

Definition in file [avl\\_set](#).

## 17.163 avl\_set

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>,
00008 * Carsten Weinhold <weinhold@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013
00014 #pragma once
00015
00016 #include <l4/cxx/std_alloc>
00017 #include <l4/cxx/std_ops>
00018 #include <l4/cxx/type_traits>
00019 #include <l4/cxx/avl_tree>
00020
00021 struct Avl_set_tester;
00022
00023 namespace cxx {
00024 namespace Bits {
00025 template< typename Node, typename Key, typename Node_op >
00026 class Avl_set_iter : public __Bst_iter_b<Node, Node_op>
00027 {
00028 private:
00029 typedef __Bst_iter_b<Node, Node_op> Base;
00030
00031 typedef typename Type_traits<Key>::Non_const_type Non_const_key;
00032
00033 typedef Avl_set_iter<Node, Non_const_key, Node_op> Non_const_iter;
00034
00035 using Base::_n;
00036 using Base::_r;
00037 using Base::inc;
00038 public:
00039 Avl_set_iter() = default;

```

```

00053
00058 Avl_set_iter(Node const *t) : Base(t) {}
00059
00064 Avl_set_iter(Base const &o) : Base(o) {}
00065
00067 Avl_set_iter(Non_const_iter const &o)
00068 : Base(o) {}
00069
00071 Avl_set_iter &operator = (Non_const_iter const &o)
00072 { Base::operator = (o); return *this; }
00073
00078 Key &operator * () const { return const_cast<Node*>(_n)->item; }
00083 Key *operator -> () const { return &const_cast<Node*>(_n)->item; }
00087 Avl_set_iter &operator ++ () { inc(); return *this; }
00091 Avl_set_iter operator ++ (int)
00092 { Avl_set_iter tmp = *this; inc(); return tmp; }
00093
00094 };
00095
00097 template<typename KEY_TYPE>
00098 struct Avl_set_get_key
00099 {
00100 typedef KEY_TYPE Key_type;
00101 template<typename NODE>
00102 static Key_type const &key_of(NODE const *n)
00103 { return n->item; }
00104 };
00105
00106
00119 template< typename ITEM_TYPE, class COMPARE,
00120 template<typename A> class ALLOC,
00121 typename GET_KEY>
00122 class Base_avl_set
00123 {
00124 friend struct ::Avl_set_tester;
00125
00126 public:
00133 enum
00134 {
00135 E_noent = 2,
00136 E_exist = 17,
00137 E_nomem = 12,
00138 E_inval = 22
00139 };
00141 typedef ITEM_TYPE Item_type;
00143 typedef GET_KEY Get_key;
00145 typedef typename GET_KEY::Key_type Key_type;
00147 typedef typename Type_traits<Item_type>::Const_type Const_item_type;
00149 typedef COMPARE Item_compare;
00150
00151 private:
00153 class _Node : public Avl_tree_node
00154 {
00155 public:
00157 Item_type item;
00158
00159 _Node() = default;
00160
00161 _Node(Item_type const &item) : Avl_tree_node(), item(item) {}
00162
00163 template<typename ...ARGS>
00164 _Node(ARGS &&...args) : Avl_tree_node(), item(cxx::forward<ARGS>(args)...)
00165 {}
00166 };
00167
00168 public:
00172 class Node
00173 {
00174 private:
00175 struct No_type;
00176 friend class Base_avl_set<ITEM_TYPE, COMPARE, ALLOC, GET_KEY>;
00177 _Node const *_n;
00178 explicit Node(_Node const *n) : _n(n) {}
00179
00180 public:
00182 Node() : _n(0) {}
00183
00189 Item_type const &operator * () { return _n->item; }
00195 Item_type const *operator -> () { return &_n->item; }
00196
00201 bool valid() const { return _n; }
00202
00204 operator Item_type const * () { return _n ? &_n->item : 0; }
00205 };
00206
00208 typedef ALLOC<_Node> Node_allocator;
00209

```

```

00210 private:
00211 typedef Avl_tree<_Node, GET_KEY, COMPARE> Tree;
00212 Tree _tree;
00214 Node_allocator _alloc;
00215
00216 Base_avl_set &operator = (Base_avl_set const &) = delete;
00217
00218 typedef typename Tree::Fwd_iter_ops Fwd;
00219 typedef typename Tree::Rev_iter_ops Rev;
00220
00221 public:
00222 typedef typename Type_traits<Item_type>::Param_type Item_param_type;
00223
00225 typedef Avl_set_iter<_Node, Item_type, Fwd> Iterator;
00226 typedef Iterator iterator;
00228 typedef Avl_set_iter<_Node, Const_item_type, Fwd> Const_iterator;
00229 typedef Const_iterator const_iterator;
00231 typedef Avl_set_iter<_Node, Item_type, Rev> Rev_iterator;
00232 typedef Rev_iterator reverse_iterator;
00234 typedef Avl_set_iter<_Node, Const_item_type, Rev> Const_rev_iterator;
00235 typedef Const_rev_iterator const_reverse_iterator;
00236
00243 explicit Base_avl_set(Node_allocator const &alloc = Node_allocator())
00244 : _tree(), _alloc(alloc)
00245 {}
00246
00247 ~Base_avl_set()
00248 {
00249 _tree.remove_all([this](_Node *n)
00250 {
00251 n->~_Node();
00252 _alloc.free(n);
00253 });
00254 }
00255
00263 inline Base_avl_set(Base_avl_set const &o);
00264
00282 cxx::Pair<Iterator, int> insert(Item_type const &item);
00283
00284 template<typename... Args>
00285 cxx::Pair<Iterator, int> emplace(Args&&... args);
00286
00295 int remove(Key_type const &item)
00296 {
00297 _Node *n = _tree.remove(item);
00298
00299 if (n)
00300 {
00301 n->~_Node();
00302 _alloc.free(n);
00303 return 0;
00304 }
00305
00306 return -E_noent;
00307 }
00308
00313 int erase(Key_type const &item)
00314 { return remove(item); }
00315
00324 Node find_node(Key_type const &item) const
00325 { return Node(_tree.find_node(item)); }
00326
00335 Node lower_bound_node(Key_type const &key) const
00336 { return Node(_tree.lower_bound_node(key)); }
00337
00338 Node lower_bound_node(Key_type &&key) const
00339 { return Node(_tree.lower_bound_node(key)); }
00340
00345 Const_iterator begin() const { return _tree.begin(); }
00350 Const_iterator end() const { return _tree.end(); }
00351
00356 Iterator begin() { return _tree.begin(); }
00361 Iterator end() { return _tree.end(); }
00362
00367 Const_rev_iterator rbegin() const { return _tree.rbegin(); }
00372 Const_rev_iterator rend() const { return _tree.rend(); }
00373
00378 Rev_iterator rbegin() { return _tree.rbegin(); }
00383 Rev_iterator rend() { return _tree.rend(); }
00384
00385 Const_iterator find(Key_type const &item) const
00386 { return _tree.find(item); }
00387
00388 #ifdef __DEBUG_L4_AVL
00389 bool rec_dump(bool print)
00390 {
00391 return _tree.rec_dump(print);

```

```

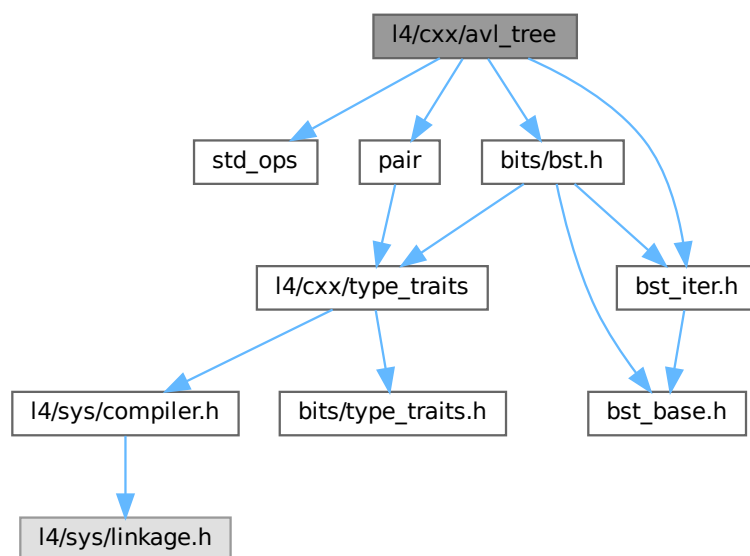
00392 }
00393 #endif
00394 };
00395
00396
00397 //-----
00398 /* Implementation of AVL Tree */
00399
00400 /* Create a copy */
00401 template< typename Item, class Compare, template<typename A> class Alloc, typename KEY_TYPE>
00402 Base_avl_set<Item,Compare,Alloc,KEY_TYPE>::Base_avl_set(Base_avl_set const &o)
00403 : _tree(), _alloc(o._alloc)
00404 {
00405 for (Const_iterator i = o.begin(); i != o.end(); ++i)
00406 insert(*i);
00407 }
00408
00409 /* Insert new _Node. */
00410 template< typename Item, class Compare, template< typename A > class Alloc, typename KEY_TYPE>
00411 Pair<typename Base_avl_set<Item,Compare,Alloc,KEY_TYPE>::Iterator, int>
00412 Base_avl_set<Item,Compare,Alloc,KEY_TYPE>::insert(Item const &item)
00413 {
00414 _Node *n = _alloc.alloc();
00415 if (!n)
00416 return cxx::pair(end(), -E_nomem);
00417 new (n, Nothrow()) _Node(item);
00418 Pair<_Node *, bool> err = _tree.insert(n);
00419 if (!err.second)
00420 {
00421 n->~_Node();
00422 _alloc.free(n);
00423 }
00424 }
00425
00426 return cxx::pair(Iterator(typename Tree::Iterator(err.first, err.first)), err.second ? 0 :
-E_exist);
00427 }
00428
00429 /* In-place insert new _Node. */
00430 template< typename Item, class Compare, template< typename A > class Alloc, typename KEY_TYPE>
00431 template<typename... Args>
00432 Pair<typename Base_avl_set<Item,Compare,Alloc,KEY_TYPE>::Iterator, int>
00433 Base_avl_set<Item,Compare,Alloc,KEY_TYPE>::emplace(Args&&... args)
00434 {
00435 _Node *n = _alloc.alloc();
00436 if (!n)
00437 return cxx::pair(end(), -E_nomem);
00438 new (n, Nothrow()) _Node(cxx::forward<Args>(args)...);
00439 Pair<_Node *, bool> err = _tree.insert(n);
00440 if (!err.second)
00441 {
00442 n->~_Node();
00443 _alloc.free(n);
00444 }
00445 }
00446
00447 return cxx::pair(Iterator(typename Tree::Iterator(err.first, err.first)), err.second ? 0 :
-E_exist);
00448 }
00449
00450 } // namespace Bits
00451
00452 template< typename ITEM_TYPE, class COMPARE = Lt_functor<ITEM_TYPE>,
00453 template<typename A> class ALLOC = New_allocator>
00454 class Avl_set :
00455 public Bits::Base_avl_set<ITEM_TYPE, COMPARE, ALLOC,
00456 Bits::Avl_set_get_key<ITEM_TYPE> >
00457 {
00458 private:
00459 typedef Bits::Base_avl_set<ITEM_TYPE, COMPARE, ALLOC,
00460 Bits::Avl_set_get_key<ITEM_TYPE> > Base;
00461 public:
00462 typedef typename Base::Node_allocator Node_allocator;
00463 Avl_set() = default;
00464 Avl_set(Node_allocator const &alloc)
00465 : Base(alloc)
00466 {}
00467 };
00468
00469 } // namespace cxx

```

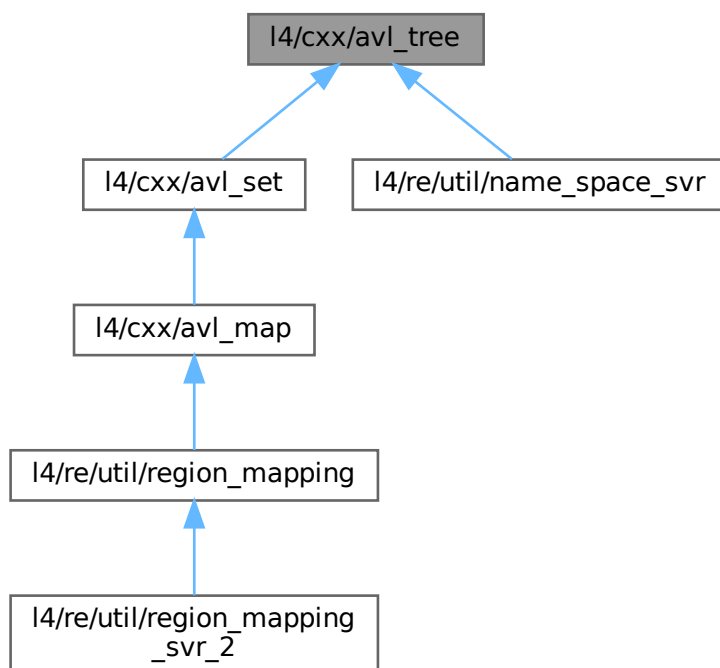
## 17.164 l4/cxx/avl\_tree File Reference

AVL tree.

```
#include "std_ops"
#include "pair"
#include "bits/bst.h"
#include "bits/bst_iter.h"
Include dependency graph for avl_tree:
```



This graph shows which files directly or indirectly include this file:



## Data Structures

- class [cxx::Avl\\_tree\\_node](#)  
*Node of an AVL tree.*
- class [cxx::Avl\\_tree< Node, Get\\_key, Compare >](#)  
*A generic AVL tree.*

## Namespaces

- namespace [cxx](#)  
*Our C++ library.*

### 17.164.1 Detailed Description

AVL tree.

Definition in file [avl\\_tree](#).

## 17.165 avl\_tree

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>,
00008 * Carsten Weinhold <weinhold@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013
00014 #pragma once
00015
00016 #include "std_ops"
00017 #include "pair"
00018
00019 #include "bits/bst.h"
00020 #include "bits/bst_iter.h"
00021
00022 struct Avl_set_tester;
00023
00024 namespace cxx {
00025
00026 class Avl_tree_node : public Bits::Bst_node
00027 {
00028 private:
00029 friend struct ::Avl_set_tester;
00030
00031 private:
00032 template< typename Node, typename Get_key, typename Compare >
00033 friend class Avl_tree;
00034
00035 typedef Bits::Direction Bal;
00036 typedef Bits::Direction Dir;
00037
00038 // We are a final BST node, hide interior.
00039 using Bits::Bst_node::next;
00040 using Bits::Bst_node::next_p;
00041 using Bits::Bst_node::rotate;
00042
00043 Bal _balance;
00044
00045 protected:
00046 Avl_tree_node() = default;
00047
00048 private:
00049 Avl_tree_node(Avl_tree_node const &o) = delete;
00050 Avl_tree_node(Avl_tree_node &&o) = delete;
00051
00052 Avl_tree_node &operator = (Avl_tree_node const &o) = default;
00053 Avl_tree_node &operator = (Avl_tree_node &&o) = default;
00054
00055 explicit Avl_tree_node(bool) : Bits::Bst_node(true), _balance(Dir::N) {}
00056
00057 static Bits::Bst_node *rotate2(Bst_node **t, Bal idir, Bal pre);
00058
00059 bool balanced() const { return _balance == Bal::N; }
00060
00061 Bal balance() const { return _balance; }
00062
00063 void balance(Bal b) { _balance = b; }
00064 };
00065
00066 template< typename Node, typename Get_key,
00067 typename Compare = Lt_functor<typename Get_key::Key_type> >
00068 class Avl_tree : public Bits::Bst<Node, Get_key, Compare>
00069 {
00070 private:
00071 typedef Bits::Bst<Node, Get_key, Compare> Bst;
00072
00073 using Bst::_head;
00074
00075 using Bst::k;
00076
00077 typedef typename Avl_tree_node::Bal Bal;
00078 typedef typename Avl_tree_node::Dir Dir;
00079
00080 Avl_tree(Avl_tree const &o) = delete;
00081 Avl_tree &operator = (Avl_tree const &o) = delete;
00082 Avl_tree(Avl_tree &&o) = delete;
00083 Avl_tree &operator = (Avl_tree &&o) = delete;
00084
00085 public:
00086 typedef typename Bst::Key_type Key_type;

```



```

00124 typedef typename Bst::Key_param_type Key_param_type;
00126
00127 // Grab iterator types from Bst
00130 typedef typename Bst::Iterator Iterator;
00132 typedef typename Bst::Const_iterator Const_iterator;
00134 typedef typename Bst::Rev_iterator Rev_iterator;
00136 typedef typename Bst::Const_rev_iterator Const_rev_iterator;
00138
00146 Pair<Node *, bool> insert(Node *new_node);
00147
00154 Node *remove(Key_param_type key);
00158 Node *erase(Key_param_type key) { return remove(key); }
00159
00161 Avl_tree() = default;
00162
00164 ~Avl_tree() noexcept
00165 {
00166 this->remove_all([](Node *){});
00167 }
00168
00169 #ifdef __DEBUG_L4_AVL
00170 bool rec_dump(Avl_tree_node *n, int depth, int *dp, bool print, char pfx);
00171 bool rec_dump(bool print)
00172 {
00173 int dp=0;
00174 return rec_dump(static_cast<Avl_tree_node *>(_head), 0, &dp, print, '+');
00175 }
00176 #endif
00177 };
00178
00179
00180 //-----
00181 /* IMPLEMENTATION: Bits::__Bst_iter_b */
00182
00183
00184 inline
00185 Bits::Bst_node *
00186 Avl_tree_node::rotate2(Bst_node **t, Bal idir, Bal pre)
00187 {
00188 typedef Bits::Bst_node N;
00189 typedef Avl_tree_node A;
00190 N *tmp[2] = { *t, N::next(*t, idir) };
00191 *t = N::next(tmp[1], !idir);
00192 A *n = static_cast<A*>(*t);
00193
00194 N::next(tmp[0], idir, N::next(n, !idir));
00195 N::next(tmp[1], !idir, N::next(n, idir));
00196 N::next(n, !idir, tmp[0]);
00197 N::next(n, idir, tmp[1]);
00198
00199 n->balance(Bal::N);
00200
00201 if (pre == Bal::N)
00202 {
00203 static_cast<A*>(tmp[0])->balance(Bal::N);
00204 static_cast<A*>(tmp[1])->balance(Bal::N);
00205 return 0;
00206 }
00207
00208 static_cast<A*>(tmp[pre != idir])->balance(!pre);
00209 static_cast<A*>(tmp[pre == idir])->balance(Bal::N);
00210
00211 return N::next(tmp[pre == idir], !pre);
00212 }
00213
00214 //-----
00215 /* Implementation of AVL Tree */
00216
00217 /* Insert new _Node. */
00218 template< typename Node, typename Get_key, class Compare>
00219 Pair<Node *, bool>
00220 Avl_tree<Node, Get_key, Compare>::insert(Node *new_node)
00221 {
00222 typedef Avl_tree_node A;
00223 typedef Bits::Bst_node N;
00224 N **t = &_head; /* search variable */
00225 N **s = &_head; /* node where rebalancing may occur */
00226 Key_param_type new_key = Get_key::key_of(new_node);
00227
00228 // search insertion point
00229 for (N *p; (p = *t);)
00230 {
00231 Dir b = this->dir(new_key, p);
00232 if (b == Dir::N)
00233 return pair(static_cast<Node*>(p), false);
00234
00235 if (!static_cast<A const *>(p)->balanced())

```

```

00236 s = t;
00237
00238 t = A::next_p(p, b);
00239 }
00240
00241 *static_cast<A*>(new_node) = A(true);
00242 *t = new_node;
00243
00244 N *n = *s;
00245 A *a = static_cast<A*>(n);
00246 if (!a->balanced())
00247 {
00248 A::Bal b(this->greater(new_key, n));
00249 if (a->balance() != b)
00250 {
00251 // ok we got in balance the shorter subtree go higher
00252 a->balance(Bal::N);
00253 // propagate the new balance down to the new node
00254 n = A::next(n, b);
00255 }
00256 else if (b == Bal(this->greater(new_key, A::next(n, b))))
00257 {
00258 // left-left or right-right case -> single rotation
00259 A::rotate(s, b);
00260 a->balance(Bal::N);
00261 static_cast<A*>(*s)->balance(Bal::N);
00262 n = A::next(*s, b);
00263 }
00264 else
00265 {
00266 // need a double rotation
00267 n = A::next(A::next(n, b), !b);
00268 n = A::rotate2(s, b, n == new_node ? Bal::N : Bal(this->greater(new_key, n)));
00269 }
00270 }
00271
00272 for (A::Bal b; n && n != new_node; static_cast<A*>(n)->balance(b), n = A::next(n, b))
00273 b = Bal(this->greater(new_key, n));
00274
00275 return pair(new_node, true);
00276 }
00277
00278
00279 /* remove an element */
00280 template< typename Node, typename Get_key, class Compare>
00281 inline
00282 Node *Avl_tree<Node, Get_key, Compare>::remove(Key_param_type key)
00283 {
00284 typedef Avl_tree_node A;
00285 typedef Bits::Bst_node N;
00286 N **q = &_head; /* search variable */
00287 N **s = &_head; /* last ('deepest') node on the search path to q
00288 * with balance 0, at this place the rebalancing
00289 * stops in any case */
00290 N **t = 0;
00291 Dir dir;
00292
00293 // find target node and rebalancing entry
00294 for (N *n; (n = *q); q = A::next_p(n, dir))
00295 {
00296 dir = Dir(this->greater(key, n));
00297 if (dir == Dir::L && !this->greater(k(n), key))
00298 /* found node */
00299 t = q;
00300
00301 if (!A::next(n, dir))
00302 break;
00303
00304 A const *a = static_cast<A const *>(n);
00305 if (a->balanced() || (a->balance() == !dir && A::next<A>(n, !dir)->balanced()))
00306 s = q;
00307 }
00308
00309 // nothing found
00310 if (!t)
00311 return 0;
00312
00313 A *i = static_cast<A*>(*t);
00314
00315 for (N *n; (n = *s); s = A::next_p(n, dir))
00316 {
00317 dir = Dir(this->greater(key, n));
00318
00319 if (!A::next(n, dir))
00320 break;
00321
00322 A *a = static_cast<A*>(n);

```

```

00323 // got one out of balance
00324 if (a->balanced())
00325 a->balance(!dir);
00326 else if (a->balance() == dir)
00327 a->balance(Bal::N);
00328 else
00329 {
00330 // we need rotations to get in balance
00331 Bal b = A::next<A>(n, !dir)->balance();
00332 if (b == dir)
00333 A::rotate2(s, !dir, A::next<A>(A::next(n, !dir), dir)->balance());
00334 else
00335 {
00336 A::rotate(s, !dir);
00337 if (b != Bal::N)
00338 {
00339 a->balance(Bal::N);
00340 static_cast<A*>(*s)->balance(Bal::N);
00341 }
00342 else
00343 {
00344 a->balance(!dir);
00345 static_cast<A*>(*s)->balance(dir);
00346 }
00347 }
00348 if (n == i)
00349 t = A::next_p(*s, dir);
00350 }
00351 }
00352
00353 A *n = static_cast<A*>(*q);
00354 *t = n;
00355 *q = A::next(n, !dir);
00356 *n = *i;
00357
00358 return static_cast<Node*>(i);
00359 }
00360
00361 #ifdef __DEBUG_L4_AVL
00362 template< typename Node, typename Get_key, class Compare>
00363 bool Avl_tree<Node, Get_key, Compare>::rec_dump(Avl_tree_node *n, int depth, int *dp, bool print, char
pfx)
00364 {
00365 typedef Avl_tree_node A;
00366
00367 if (!n)
00368 return true;
00369
00370 int dpx[2] = {depth, depth};
00371 bool res = true;
00372
00373 res = rec_dump(A::next<A>(n, Dir::R), depth + 1, dpx + 1, print, '/');
00374
00375 if (print)
00376 {
00377 fprintf(stderr, "%2d: [%8p] b=%1d: ", depth, n, (int)n->balance().d);
00378
00379 for (int i = 0; i < depth; ++i)
00380 std::cerr << " ";
00381
00382 std::cerr << pfx << (static_cast<Node*>(n)->item) << std::endl;
00383 }
00384
00385 res = res & rec_dump(A::next<A>(n, Dir::L), depth + 1, dpx, print, '\\');
00386
00387 int b = dpx[1] - dpx[0];
00388
00389 if (b < 0)
00390 *dp = dpx[0];
00391 else
00392 *dp = dpx[1];
00393
00394 Bal x = n->balance();
00395 if ((b < -1 || b > 1) ||
00396 (b == 0 && x != Bal::N) ||
00397 (b == -1 && x != Bal::L) ||
00398 (b == 1 && x != Bal::R))
00399 {
00400 if (print)
00401 fprintf(stderr, "%2d: [%8p] b=%1d: balance error %d\n", depth, n, (int)n->balance().d, b);
00402 return false;
00403 }
00404 return res;
00405 }
00406 #endif
00407
00408 }

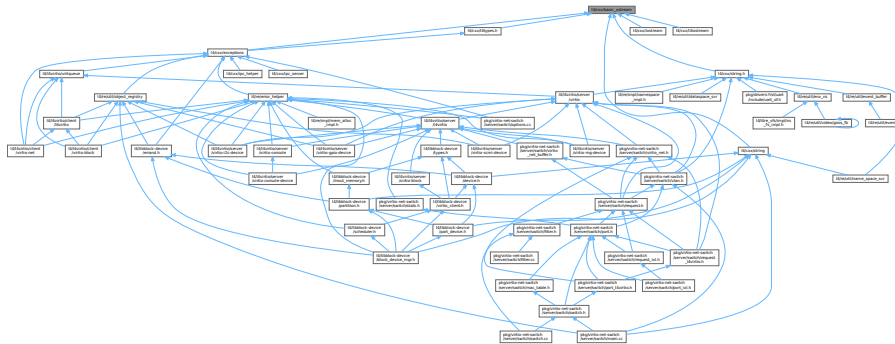
```

00409

## 17.166 l4/cxx/basic\_ostream File Reference

Basic IO stream.

This graph shows which files directly or indirectly include this file:



### Data Structures

- class [L4::IOModifier](#)  
*Modifier class for the IO stream.*

### Namespaces

- namespace [L4](#)  
*L4 low-level kernel interface.*

### Variables

- [IOModifier](#) const [L4::hex](#)  
*Modifies the stream to print numbers as hexadecimal values.*
- [IOModifier](#) const [L4::dec](#)  
*Modifies the stream to print numbers as decimal values.*

### 17.166.1 Detailed Description

Basic IO stream.

Definition in file [basic\\_ostream](#).

## 17.167 basic\_ostream

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2009 Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00014 namespace L4 {
00015
00022 class IOModifier
00023 {
00024 public:
00025 IOModifier(int x) : mod(x) {}
00026 bool operator == (IOModifier o) { return mod == o.mod; }
00027 bool operator != (IOModifier o) { return mod != o.mod; }
00028 int mod;
00029 };
00030
00035 class IOBackend
00036 {
00037 public:
00038 typedef int Mode;
00039
00040 protected:
00041 friend class BasicOStream;
00042
00043 IOBackend()
00044 : int_mode(10)
00045 {}
00046
00047 virtual ~IOBackend() {}
00048
00049 virtual void write(char const *str, unsigned len) = 0;
00050
00051 private:
00052 void write(IOModifier m);
00053 void write(long long int c, int len);
00054 void write(long long unsigned c, int len);
00055 void write(long long unsigned c, unsigned char base = 10,
00056 unsigned char len = 0, char pad = ' ');
00057
00058 Mode mode() const
00059 { return int_mode; }
00060
00061 void mode(Mode m)
00062 { int_mode = m; }
00063
00064 int int_mode;
00065 };
00066
00071 class BasicOStream
00072 {
00073 public:
00074 BasicOStream(IOBackend *b)
00075 : iob(b)
00076 {}
00077
00078 void write(char const *str, unsigned len)
00079 {
00080 if (iob)
00081 iob->write(str, len);
00082 }
00083
00084 void write(long long int c, int len)
00085 {
00086 if (iob)
00087 iob->write(c, len);
00088 }
00089
00090 void write(long long unsigned c, unsigned char base = 10,
00091 unsigned char len = 0, char pad = ' ')
00092 {
00093 if (iob)
00094 iob->write(c, base, len, pad);
00095 }
00096
00097 void write(long long unsigned c, int len)
00098 {
00099 if (iob)
00100 iob->write(c, len);

```

```

00101 }
00102
00103 void write(IOModifier m)
00104 {
00105 if (iob)
00106 iob->write(m);
00107 }
00108
00109 IOBackend::Mode be_mode() const
00110 {
00111 if (iob)
00112 return iob->mode();
00113 return 0;
00114 }
00115
00116 void be_mode(IOBackend::Mode m)
00117 {
00118 if (iob)
00119 iob->mode(m);
00120 }
00121
00122 private:
00123 IOBackend *iob;
00124 };
00125
00130 class IONumFmt
00131 {
00132 public:
00133 IONumFmt(unsigned long long n, unsigned char base = 10,
00134 unsigned char len = 0, char pad = ' ')
00135 : n(n), base(base), len(len), pad(pad)
00136 {}
00137
00138 BasicOStream &print(BasicOStream &o) const;
00139
00140 private:
00141 unsigned long long n;
00142 unsigned char base, len;
00143 char pad;
00144 };
00145
00146 inline IONumFmt n_hex(unsigned long long n) { return IONumFmt(n, 16); }
00147
00151 extern IOModifier const hex;
00152
00156 extern IOModifier const dec;
00157
00158 inline
00159 BasicOStream &IONumFmt::print(BasicOStream &o) const
00160 {
00161 o.write(n, base, len, pad);
00162 return o;
00163 }
00164 }
00165
00166 // Implementation
00167
00169 inline
00170 L4::BasicOStream &
00171 operator « (L4::BasicOStream &s, char const * const str)
00172 {
00173 if (!str)
00174 {
00175 s.write("(NULL)", 6);
00176 return s;
00177 }
00178
00179 unsigned l = 0;
00180 for (; str[l] != 0; l++)
00181 ;
00182 s.write(str, l);
00183 return s;
00184 }
00185
00186 inline
00187 L4::BasicOStream &
00188 operator « (L4::BasicOStream &s, signed short u)
00189 {
00190 s.write(static_cast<long long signed>(u), -1);
00191 return s;
00192 }
00193
00194 inline
00195 L4::BasicOStream &
00196 operator « (L4::BasicOStream &s, signed u)
00197 {

```

```

00198 s.write(static_cast<long long signed>(u), -1);
00199 return s;
00200 }
00201
00202 inline
00203 L4::BasicOStream &
00204 operator « (L4::BasicOStream &s, signed long u)
00205 {
00206 s.write(static_cast<long long signed>(u), -1);
00207 return s;
00208 }
00209
00210 inline
00211 L4::BasicOStream &
00212 operator « (L4::BasicOStream &s, signed long long u)
00213 {
00214 s.write(u, -1);
00215 return s;
00216 }
00217
00218 inline
00219 L4::BasicOStream &
00220 operator « (L4::BasicOStream &s, unsigned short u)
00221 {
00222 s.write(static_cast<long long unsigned>(u), -1);
00223 return s;
00224 }
00225
00226 inline
00227 L4::BasicOStream &
00228 operator « (L4::BasicOStream &s, unsigned u)
00229 {
00230 s.write(static_cast<long long unsigned>(u), -1);
00231 return s;
00232 }
00233
00234 inline
00235 L4::BasicOStream &
00236 operator « (L4::BasicOStream &s, unsigned long u)
00237 {
00238 s.write(static_cast<long long unsigned>(u), -1);
00239 return s;
00240 }
00241
00242 inline
00243 L4::BasicOStream &
00244 operator « (L4::BasicOStream &s, unsigned long long u)
00245 {
00246 s.write(u, -1);
00247 return s;
00248 }
00249
00250 inline
00251 L4::BasicOStream &
00252 operator « (L4::BasicOStream &s, void const *u)
00253 {
00254 long unsigned x = reinterpret_cast<long unsigned>(u);
00255 L4::IOBackend::Mode mode = s.be_mode();
00256 s.write(L4::hex);
00257 s.write(static_cast<long long unsigned>(x), -1);
00258 s.be_mode(mode);
00259 return s;
00260 }
00261
00262 inline
00263 L4::BasicOStream &
00264 operator « (L4::BasicOStream &s, L4::IOModifier m)
00265 {
00266 s.write(m);
00267 return s;
00268 }
00269
00270 inline
00271 L4::BasicOStream &
00272 operator « (L4::BasicOStream &s, char c)
00273 {
00274 s.write(&c, 1);
00275 return s;
00276 }
00277
00278 inline
00279 L4::BasicOStream &
00280 operator « (L4::BasicOStream &o, L4::IONumFmt const &n)
00281 { return n.print(o); }

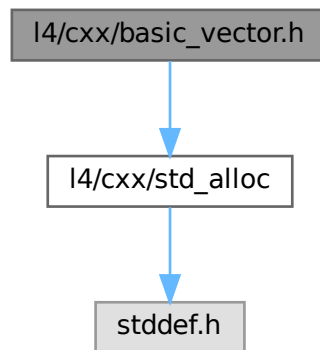
```

## 17.168 l4/cxx/basic\_vector.h File Reference

Basic vector.

```
#include <l4/cxx/std_alloc>
```

Include dependency graph for basic\_vector.h:



### Namespaces

- namespace `cxx`  
*Our C++ library.*

### 17.168.1 Detailed Description

Basic vector.

Definition in file [basic\\_vector.h](#).

## 17.169 basic\_vector.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #pragma once
00012
00013 #include <l4/cxx/std_alloc>
00014
00015 namespace cxx {
00016
00017 template< typename T >
00018 class Basic_vector
00019 {

```



```

00020 public:
00021 Basic_vector(T *array, unsigned long capacity)
00022 : _array(array), _capacity(capacity)
00023 {
00024 for (unsigned long i = 0; i < capacity; ++i)
00025 new (&_array[i]) T();
00026 }
00027
00028 private:
00029 T *_array;
00030 unsigned long _capacity;
00031 };
00032
00033 };

```

## 17.170 bitfield

```

00001 // vi:set ft=c++: -- Mode: C++ --
00002 /*
00003 * (c) 2012 Alexander Warg <warg@os.inf.tu-dresden.de>,
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00009 #pragma once
00010
00011 #include "type_list"
00012
00013 namespace cxx {
00014
00015 template<typename T, unsigned LSB, unsigned MSB>
00016 class Bitfield
00017 {
00018 private:
00019 typedef remove_reference_t<T> Base_type;
00020
00021 static_assert(MSB >= LSB, "boundary mismatch in bit-field definition");
00022 static_assert(MSB < sizeof(Base_type) * 8, "MSB outside of bit-field type");
00023 static_assert(LSB < sizeof(Base_type) * 8, "LSB outside of bit-field type");
00024
00025 template<unsigned BITS> struct Best_type
00026 {
00027 template< typename TY > struct Cmp { enum { value = (BITS <= sizeof(TY)*8) }; };
00028 typedef cxx::type_list<
00029 unsigned char,
00030 unsigned short,
00031 unsigned int,
00032 unsigned long,
00033 unsigned long long
00034 > Unsigned_types;
00035 typedef cxx::find_type_t<Unsigned_types, Cmp> Type;
00036 };
00037
00038 public:
00039 enum
00040 {
00041 Bits = MSB + 1 - LSB,
00042 Lsb = LSB,
00043 Msb = MSB,
00044 };
00045
00046 enum Masks : Base_type
00047 {
00048 Low_mask = static_cast<Base_type>(~0ULL) >> (sizeof(Base_type)*8 - Bits),
00049 Mask = Low_mask << Lsb,
00050 };
00051
00052 typedef typename Best_type<Bits>::Type Bits_type;
00053
00054 typedef typename Best_type<Bits + Lsb>::Type Shift_type;
00055
00056 private:
00057 static_assert(sizeof(Bits_type)*8 >= Bits, "error finding the type to store the bits");
00058 static_assert(sizeof(Shift_type)*8 >= Bits + Lsb, "error finding the type to keep the shifted bits");
00059 static_assert(sizeof(Bits_type) <= sizeof(Base_type), "size mismatch for Bits_type");
00060 static_assert(sizeof(Shift_type) <= sizeof(Base_type), "size mismatch for Shift_type");
00061 static_assert(sizeof(Bits_type) <= sizeof(Shift_type), "size mismatch for Shift_type and Bits_type");
00062
00063 public:
00064 static constexpr Bits_type get(Shift_type val)

```

```

00100 { return (val >> Lsb) & Low_mask; }
00101
00112 static constexpr Base_type get_unshifted(Shift_type val)
00113 { return val & Mask; }
00114
00127 static constexpr Base_type set_dirty(Base_type dest, Shift_type val)
00128 {
00129 //assert (!(val & ~Low_mask));
00130 return (dest & ~Mask) | (val << Lsb);
00131 }
00132
00147 static constexpr Base_type set_unshifted_dirty(Base_type dest, Shift_type val)
00148 {
00149 //assert (!(val & ~Mask));
00150 return (dest & ~Mask) | val;
00151 }
00152
00161 static Base_type set(Base_type dest, Bits_type val)
00162 { return set_dirty(dest, val & Low_mask); }
00163
00173 static Base_type set_unshifted(Base_type dest, Shift_type val)
00174 { return set_unshifted_dirty(dest, val & Mask); }
00175
00187 static constexpr Base_type val_dirty(Shift_type val) { return val << Lsb; }
00188
00196 static constexpr Base_type val(Bits_type val) { return val_dirty(val & Low_mask); }
00197
00206 static constexpr Base_type val_unshifted(Shift_type val) { return val & Mask; }
00207
00209 template< typename TT >
00210 class Value_base
00211 {
00212 private:
00213 TT v;
00214
00215 public:
00216 constexpr Value_base(TT t) : v(t) {}
00217 constexpr Bits_type get() const { return Bitfield::get(v); }
00218 constexpr Base_type get_unshifted() const { return Bitfield::get_unshifted(v); }
00219
00220 void set(Bits_type val) { v = Bitfield::set(v, val); }
00221 void set_dirty(Bits_type val) { v = Bitfield::set_dirty(v, val); }
00222 void set_unshifted(Shift_type val) { v = Bitfield::set_unshifted(v, val); }
00223 void set_unshifted_dirty(Shift_type val) { v = Bitfield::set_unshifted_dirty(v, val); }
00224 };
00225
00227 template< typename TT >
00228 class Value : public Value_base<TT>
00229 {
00230 public:
00231 constexpr Value(TT t) : Value_base<TT>(t) {}
00232 constexpr operator Bits_type () const { return this->get(); }
00233 constexpr Value &operator = (Bits_type val) { this->set(val); return *this; }
00234 constexpr Value &operator = (Value const &val)
00235 { this->set(val.get()); return *this; }
00236 Value(Value const &) = default;
00237 };
00238
00240 template< typename TT >
00241 class Value_unshifted : public Value_base<TT>
00242 {
00243 public:
00244 constexpr Value_unshifted(TT t) : Value_base<TT>(t) {}
00245 constexpr operator Shift_type () const { return this->get_unshifted(); }
00246 constexpr Value_unshifted &operator = (Shift_type val) { this->set_unshifted(val); return *this; }
00247 constexpr Value_unshifted &operator = (Value_unshifted const &val)
00248 { this->set_unshifted(val.get_unshifted()); return *this; }
00249 Value_unshifted(Value_unshifted const &) = default;
00250 };
00251
00253 typedef Value<Base_type &> Ref;
00255 typedef Value<Base_type volatile &> Ref_volatile;
00257 typedef Value<Base_type const> Val;
00258
00260 typedef Value_unshifted<Base_type &> Ref_unshifted;
00262 typedef Value_unshifted<Base_type volatile &> Ref_unshifted_volatile;
00264 typedef Value_unshifted<Base_type const> Val_unshifted;
00265 };
00266
00267 #define CXX_BITFIELD_MEMBER(LSB, MSB, name, data_member) \
00268 \
00269 \
00270 typedef cxx::Bitfield<decltype(data_member), LSB, MSB> name ## _bfm_t; \
00271 \
00272 constexpr typename name ## _bfm_t::Val name() const { return data_member; } \
00273 typename name ## _bfm_t::Val name() const volatile { return data_member; } \
00274 \

```

```

00275 constexpr typename name##_bfm_t::Ref name() { return data_member; } \
00276 typename name##_bfm_t::Ref_volatile name() volatile { return data_member; } \
00277
00278
00279 #define CXX_BITFIELD_MEMBER_RO(LSB, MSB, name, data_member) \
00280 \
00281 \
00282 typedef cxx::Bitfield<decltype(data_member), LSB, MSB> name##_bfm_t; \
00283 \
00284 constexpr typename name##_bfm_t::Val name() const { return data_member; } \
00285 typename name##_bfm_t::Val name() const volatile { return data_member; } \
00286
00287
00288 #define CXX_BITFIELD_MEMBER_UNSHIFTED(LSB, MSB, name, data_member) \
00289 \
00290 \
00291 typedef cxx::Bitfield<decltype(data_member), LSB, MSB> name##_bfm_t; \
00292 \
00293 constexpr typename name##_bfm_t::Val_unshifted name() const { return data_member; } \
00294 typename name##_bfm_t::Val_unshifted name() const volatile { return data_member; } \
00295 \
00296 constexpr typename name##_bfm_t::Ref_unshifted name() { return data_member; } \
00297 typename name##_bfm_t::Ref_unshifted_volatile name() volatile { return data_member; } \
00298
00299
00300 #define CXX_BITFIELD_MEMBER_UNSHIFTED_RO(LSB, MSB, name, data_member) \
00301 \
00302 \
00303 typedef cxx::Bitfield<decltype(data_member), LSB, MSB> name##_bfm_t; \
00304 \
00305 constexpr typename name##_bfm_t::Val_unshifted name() const { return data_member; } \
00306 typename name##_bfm_t::Val_unshifted name() const volatile { return data_member; } \
00307
00308 }

```

## 17.171 bitmap

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2014 Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00009 #pragma once
00010
00011 namespace cxx {
00012
00013 class Bitmap_base
00014 {
00015 protected:
00016 typedef unsigned long word_type;
00017
00018 enum
00019 {
00020 W_bits = sizeof(word_type) * 8,
00021 C_bits = 8,
00022 };
00023
00024 word_type *_bits;
00025
00026 static unsigned word_index(unsigned bit) { return bit / W_bits; }
00027
00028 static unsigned bit_index(unsigned bit) { return bit % W_bits; }
00029
00030 class Bit
00031 {
00032 {
00033 Bitmap_base *_bm;
00034 long _bit;
00035
00036 public:
00037 Bit(Bitmap_base *bm, long bit) : _bm(bm), _bit(bit) {}
00038 Bit &operator = (bool val) { _bm->bit(_bit, val); return *this; }
00039 operator bool () const { return _bm->bit(_bit); }
00040 };
00041
00042 public:
00043 explicit Bitmap_base(void *bits) noexcept : _bits(reinterpret_cast<word_type *>(bits)) {}
00044
00045 static long words(long bits) noexcept { return (bits + W_bits - 1) / W_bits; }
00046 static long bit_buffer_bytes(long bits) noexcept
00047 { return words(bits) * W_bits / 8; }

```

```

00076
00077 template< long BITS >
00078 class Word
00079 {
00080 public:
00081 typedef unsigned long Type;
00082 enum
00083 {
00084 Size = (BITS + W_bits - 1) / W_bits
00085 };
00086 };
00087
00088
00089 static long chars(long bits) noexcept
00090 { return (bits + C_bits - 1) / C_bits; }
00091
00092
00093 template< long BITS >
00094 class Char
00095 {
00096 public:
00097 typedef unsigned char Type;
00098 enum
00099 {
00100 Size = (BITS + C_bits - 1) / C_bits
00101 };
00102 };
00103
00104
00105 void bit(long bit, bool on) noexcept;
00106
00107 void clear_bit(long bit) noexcept;
00108
00109 void atomic_clear_bit(long bit) noexcept;
00110
00111 word_type atomic_get_and_clear(long bit) noexcept;
00112
00113 void set_bit(long bit) noexcept;
00114
00115 void atomic_set_bit(long bit) noexcept;
00116
00117 word_type atomic_get_and_set(long bit) noexcept;
00118
00119 word_type bit(long bit) const noexcept;
00120
00121 word_type operator [] (long bit) const noexcept
00122 { return this->bit(bit); }
00123
00124 Bit operator [] (long bit) noexcept
00125 { return Bit(this, bit); }
00126
00127 long scan_zero(long max_bit, long start_bit = 0) const noexcept;
00128
00129 void *bit_buffer() const noexcept { return _bits; }
00130
00131 protected:
00132 static int _bzl(unsigned long w) noexcept;
00133 };
00134
00135 template<int BITS>
00136 class Bitmap : public Bitmap_base
00137 {
00138 private:
00139 char _bits[Bitmap_base::Char<BITS>::Size];
00140
00141 public:
00142 Bitmap() noexcept : Bitmap_base(_bits) {}
00143 Bitmap(Bitmap<BITS> const &o) noexcept : Bitmap_base(_bits)
00144 { __builtin_memcpy(_bits, o._bits, sizeof(_bits)); }
00145 long scan_zero(long start_bit = 0) const noexcept;
00146
00147 void clear_all()
00148 { __builtin_memset(_bits, 0, sizeof(_bits)); }
00149 };
00150
00151 inline
00152 void
00153 Bitmap_base::bit(long bit, bool on) noexcept
00154 {
00155 long idx = word_index(bit);
00156 long b = bit_index(bit);
00157 _bits[idx] = (_bits[idx] & ~(1UL << b)) | (static_cast<unsigned long>(on) << b);
00158 }
00159
00160 inline
00161 void
00162 Bitmap_base::clear_bit(long bit) noexcept
00163 {

```

```

00262 long idx = word_index(bit);
00263 long b = bit_index(bit);
00264 _bits[idx] &= ~(1UL << b);
00265 }
00266
00267 inline
00268 void
00269 Bitmap_base::atomic_clear_bit(long bit) noexcept
00270 {
00271 long idx = word_index(bit);
00272 long b = bit_index(bit);
00273 word_type mask = 1UL << b;
00274 __atomic_and_fetch(&_bits[idx], ~mask, __ATOMIC_RELAXED);
00275 }
00276
00277 inline
00278 Bitmap_base::word_type
00279 Bitmap_base::atomic_get_and_clear(long bit) noexcept
00280 {
00281 long idx = word_index(bit);
00282 long b = bit_index(bit);
00283 word_type mask = 1UL << b;
00284 return __atomic_fetch_and(&_bits[idx], ~mask, __ATOMIC_RELAXED) & mask;
00285 }
00286
00287 inline
00288 void
00289 Bitmap_base::set_bit(long bit) noexcept
00290 {
00291 long idx = word_index(bit);
00292 long b = bit_index(bit);
00293 _bits[idx] |= (1UL << b);
00294 }
00295
00296 inline
00297 void
00298 Bitmap_base::atomic_set_bit(long bit) noexcept
00299 {
00300 long idx = word_index(bit);
00301 long b = bit_index(bit);
00302 word_type mask = 1UL << b;
00303 __atomic_or_fetch(&_bits[idx], mask, __ATOMIC_RELAXED);
00304 }
00305
00306 inline
00307 Bitmap_base::word_type
00308 Bitmap_base::atomic_get_and_set(long bit) noexcept
00309 {
00310 long idx = word_index(bit);
00311 long b = bit_index(bit);
00312 word_type mask = 1UL << b;
00313 return __atomic_fetch_or(&_bits[idx], mask, __ATOMIC_RELAXED) & mask;
00314 }
00315
00316 inline
00317 Bitmap_base::word_type
00318 Bitmap_base::bit(long bit) const noexcept
00319 {
00320 long idx = word_index(bit);
00321 long b = bit_index(bit);
00322 return _bits[idx] & (1UL << b);
00323 }
00324
00325 inline
00326 int
00327 Bitmap_base::_bzl(unsigned long w) noexcept
00328 {
00329 for (int i = 0; i < W_bits; ++i, w >>= 1)
00330 {
00331 if ((w & 1) == 0)
00332 return i;
00333 }
00334 return -1;
00335 }
00336
00337 inline
00338 long
00339 Bitmap_base::scan_zero(long max_bit, long start_bit) const noexcept
00340 {
00341 if (!(operator [] (start_bit)))
00342 return start_bit;
00343
00344 long idx = word_index(start_bit);
00345 max_bit -= start_bit & ~(W_bits - 1);
00346
00347 for (; max_bit > 0; max_bit -= W_bits, ++idx)

```

```

00349 {
00350 if (_bits[idx] == 0)
00351 return idx * W_bits;
00352
00353 if (_bits[idx] != ~OUL)
00354 {
00355 long zbit = _bz1(_bits[idx]);
00356 return zbit < max_bit ? idx * W_bits + zbit : -1;
00357 }
00358 }
00359
00360 return -1;
00361 }
00362
00363 template<int BITS> inline
00364 long
00365 Bitmap<BITS>::scan_zero(long start_bit) const noexcept
00366 {
00367 return Bitmap_base::scan_zero(BITS, start_bit);
00368 }
00369
00370 };

```

## 17.172 I4/cxx/bits/bst.h File Reference

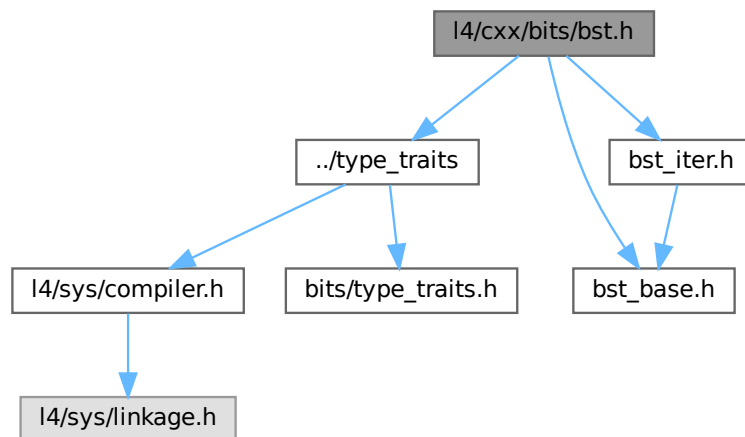
AVL tree.

```

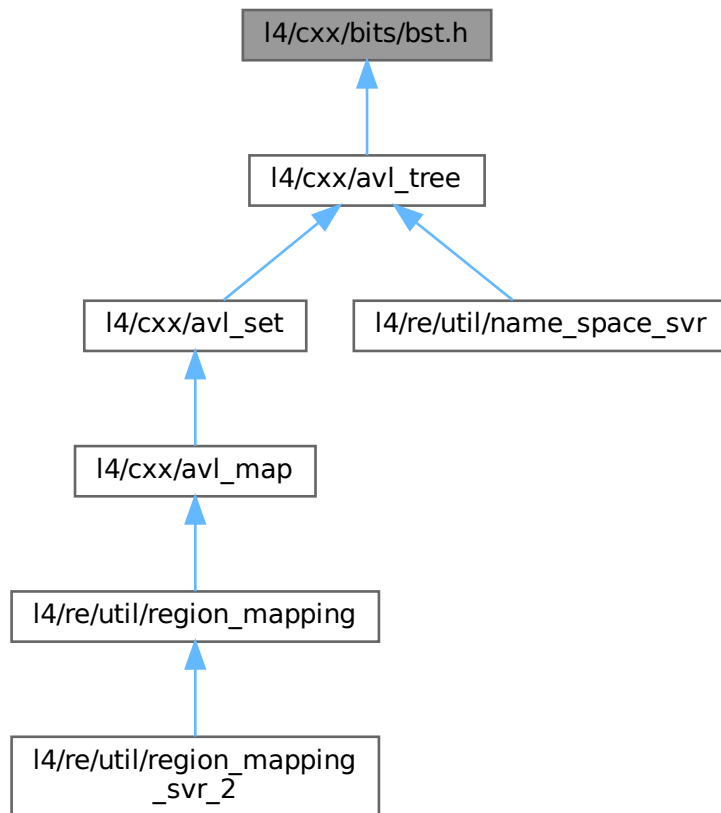
#include "../type_traits"
#include "bst_base.h"
#include "bst_iter.h"

```

Include dependency graph for bst.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class [cxx::Bits::Bst< Node, Get\\_key, Compare >](#)  
*Basic binary search tree (BST).*

## Namespaces

- namespace [cxx](#)  
*Our C++ library.*
- namespace [cxx::Bits](#)  
*Internal helpers for the cxx package.*

## 17.172.1 Detailed Description

AVL tree.

Definition in file [bst.h](#).

## 17.173 bst.h

[Go to the documentation of this file.](#)

```

00001 // vi:ft=cpp
00006 /*
00007 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>,
00008 * Carsten Weinhold <weinhold@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #pragma once
00014
00015 #include "../type_traits"
00016 #include "bst_base.h"
00017 #include "bst_iter.h"
00018
00019 struct Avl_set_tester;
00020
00021 namespace cxx { namespace Bits {
00022
00023 template< typename Node, typename Get_key, typename Compare >
00031 class Bst
00032 {
00033 friend struct ::Avl_set_tester;
00034
00035 private:
00036 typedef Direction Dir;
00037 struct Fwd
00038 {
00039 static Node *child(Node const *n, Direction d)
00040 { return Bst_node::next<Node>(n, d); }
00041
00042 static bool cmp(Node const *l, Node const *r)
00043 { return Compare() (Get_key::key_of(l), Get_key::key_of(r)); }
00044 };
00045
00046 struct Rev
00047 {
00048 static Node *child(Node const *n, Direction d)
00049 { return Bst_node::next<Node>(n, !d); }
00050
00051 static bool cmp(Node const *l, Node const *r)
00052 { return Compare() (Get_key::key_of(r), Get_key::key_of(l)); }
00053 };
00054
00055 public:
00056 typedef typename Get_key::Key_type Key_type;
00057 typedef typename Type_traits<Key_type>::Param_type Key_param_type;
00058
00059 typedef Fwd Fwd_iter_ops;
00060 typedef Rev Rev_iter_ops;
00061
00062 typedef __Bst_iter<Node, Node, Fwd> Iterator;
00063 typedef __Bst_iter<Node, Node const, Fwd> Const_iterator;
00064 typedef __Bst_iter<Node, Node, Rev> Rev_iterator;
00065 typedef __Bst_iter<Node, Node const, Rev> Const_rev_iterator;
00066
00067 protected:
00068
00069 Bst_node *_head;
00070
00071 Bst() : _head(0) {}
00072
00073 Node *head() const { return static_cast<Node*>(_head); }
00074
00075 static Key_type k(Bst_node const *n)
00076 { return Get_key::key_of(static_cast<Node const *>(n)); }
00077
00078 static Dir dir(Key_param_type l, Key_param_type r)
00079 {
00080 Compare cmp;
00081 Dir d(cmp(r, l));
00082 if (d == Direction::L && !cmp(l, r))
00083 return Direction::N;
00084 return d;
00085 }
00086
00087 static Dir dir(Key_param_type l, Bst_node const *r)
00088 { return dir(l, k(r)); }
00089
00090 static bool greater(Key_param_type l, Key_param_type r)
00091 { return Compare() (r, l); }
00092 };
00093
00094 } }

```



```

00136 static bool greater(Key_param_type l, Bst_node const *r)
00137 { return greater(l, k(r)); }
00138
00140 static bool greater(Bst_node const *l, Bst_node const *r)
00141 { return greater(k(l), k(r)); }
00143
00150 template<typename FUNC>
00151 static void remove_tree(Bst_node *head, FUNC &&callback)
00152 {
00153 if (Bst_node *n = Bst_node::next(head, Dir::L))
00154 remove_tree(n, callback);
00155
00156 if (Bst_node *n = Bst_node::next(head, Dir::R))
00157 remove_tree(n, callback);
00158
00159 callback(static_cast<Node *>(head));
00160 }
00161
00162 public:
00163
00172 Const_iterator begin() const { return Const_iterator(head()); }
00177 Const_iterator end() const { return Const_iterator(); }
00178
00183 Iterator begin() { return Iterator(head()); }
00188 Iterator end() { return Iterator(); }
00189
00194 Const_rev_iterator rbegin() const { return Const_rev_iterator(head()); }
00199 Const_rev_iterator rend() const { return Const_rev_iterator(); }
00200
00205 Rev_iterator rbegin() { return Rev_iterator(head()); }
00210 Rev_iterator rend() { return Rev_iterator(); }
00212
00213
00218
00224 Node *find_node(Key_param_type key) const;
00225
00232 Node *lower_bound_node(Key_param_type key) const;
00233
00240 Const_iterator find(Key_param_type key) const;
00241
00250 template<typename FUNC>
00251 void remove_all(FUNC &&callback)
00252 {
00253 if (!_head)
00254 return;
00255
00256 Bst_node *head = _head;
00257 _head = 0;
00258 remove_tree(head, cxx::forward<FUNC>(callback));
00259 }
00260
00261
00263 };
00264
00265 /* find an element */
00266 template< typename Node, typename Get_key, class Compare>
00267 inline
00268 Node *
00269 Bst<Node, Get_key, Compare>::find_node(Key_param_type key) const
00270 {
00271 Dir d;
00272
00273 for (Bst_node *q = _head; q; q = Bst_node::next(q, d))
00274 {
00275 d = dir(key, q);
00276 if (d == Dir::N)
00277 return static_cast<Node*>(q);
00278 }
00279 return 0;
00280 }
00281
00282 template< typename Node, typename Get_key, class Compare>
00283 inline
00284 Node *
00285 Bst<Node, Get_key, Compare>::lower_bound_node(Key_param_type key) const
00286 {
00287 Dir d;
00288 Bst_node *r = 0;
00289
00290 for (Bst_node *q = _head; q; q = Bst_node::next(q, d))
00291 {
00292 d = dir(key, q);
00293 if (d == Dir::L)
00294 r = q; // found a node greater than key
00295 else if (d == Dir::N)
00296 return static_cast<Node*>(q);
00297 }

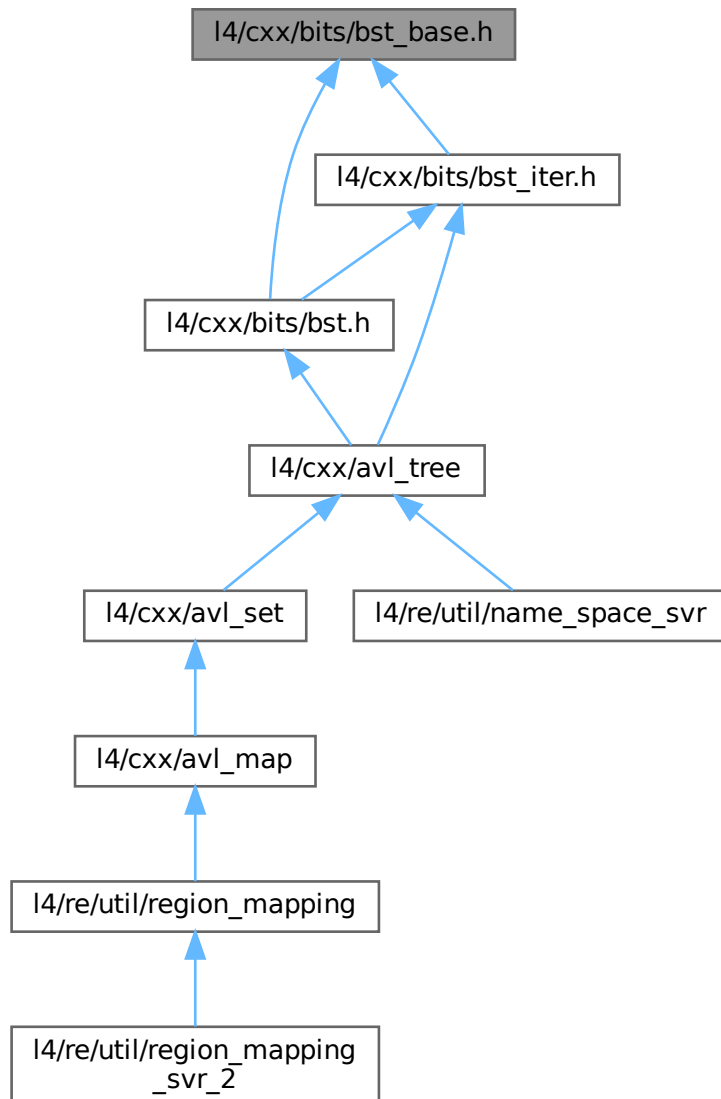
```

```
00298 return static_cast<Node*>(r);
00299 }
00300
00301 /* find an element */
00302 template< typename Node, typename Get_key, class Compare>
00303 inline
00304 typename Bst<Node, Get_key, Compare>::Const_iterator
00305 Bst<Node, Get_key, Compare>::find(Key_param_type key) const
00306 {
00307 Bst_node *q = _head;
00308 Bst_node *r = q;
00309
00310 for (Dir d; q; q = Bst_node::next(q, d))
00311 {
00312 d = dir(key, q);
00313 if (d == Dir::N)
00314 return Iterator(static_cast<Node*>(q), static_cast<Node *>(r));
00315
00316 if (d != Dir::L && q == r)
00317 r = Bst_node::next(q, d);
00318 }
00319 return Iterator();
00320 }
00321
00322 }
```

## 17.174 I4/cxx/bits/bst\_base.h File Reference

AVL tree.

This graph shows which files directly or indirectly include this file:



## Data Structures

- struct `cxx::Bits::Direction`  
*The direction to go in a binary search tree.*
- class `cxx::Bits::Bst_node`  
*Basic type of a node in a binary search tree (BST).*

## Namespaces

- namespace `cxx`  
*Our C++ library.*
- namespace `cxx::Bits`  
*Internal helpers for the cxx package.*

## 17.174.1 Detailed Description

AVL tree.

Definition in file [bst\\_base.h](#).

## 17.175 bst\_base.h

[Go to the documentation of this file.](#)

```

00001 // vi:ft=cpp
00006 /*
00007 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>,
00008 * Carsten Weinhold <weinhold@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013
00014 #pragma once
00015
00016 /*
00017 * This file contains very basic bits for implementing binary search trees
00018 */
00019 namespace cxx {
00023 namespace Bits {
00024
00028 struct Direction
00029 {
00031 enum Direction_e
00032 {
00033 L = 0,
00034 R = 1,
00035 N = 2
00036 };
00037 unsigned char d;
00038
00040 Direction() = default;
00041
00043 Direction(Direction_e d) : d(d) {}
00044
00046 explicit Direction(bool b) : d(Direction_e(b)) /*d(b ? R : L)*/ {}
00047
00052 Direction operator ! () const { return Direction(!d); }
00053
00055
00057 bool operator == (Direction_e o) const { return d == o; }
00059 bool operator != (Direction_e o) const { return d != o; }
00061 bool operator == (Direction o) const { return d == o.d; }
00063 bool operator != (Direction o) const { return d != o.d; }
00065 };
00066
00070 class Bst_node
00071 {
00072 // all BSTs are friends
00073 template< typename Node, typename Get_key, typename Compare >
00074 friend class Bst;
00075
00076 protected:
00085
00087 static Bst_node *next(Bst_node const *p, Direction d)
00088 { return p->_c[d.d]; }
00089
00091 static void next(Bst_node *p, Direction d, Bst_node *n)
00092 { p->_c[d.d] = n; }
00093
00095 static Bst_node **next_p(Bst_node *p, Direction d)
00096 { return &p->_c[d.d]; }
00097
00099 template< typename Node > static
00100 Node *next(Bst_node const *p, Direction d)
00101 { return static_cast<Node *>(p->_c[d.d]); }
00102
00104 static void rotate(Bst_node **t, Direction idir);
00106
00107 private:
00108 Bst_node *_c[2];
00109
00110 protected:

```

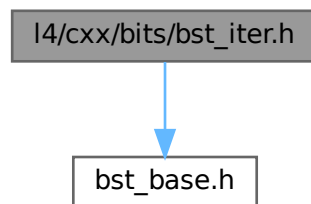
```
00112 Bst_node() {}
00113
00115 explicit Bst_node(bool) { _c[0] = _c[1] = 0; }
00116 };
00117
00118 inline
00119 void
00120 Bst_node::rotate(Bst_node **t, Direction idir)
00121 {
00122 Bst_node *tmp = *t;
00123 *t = next(tmp, idir);
00124 next(tmp, idir, next(*t, !idir));
00125 next(*t, !idir, tmp);
00126 }
00127
00128 }
```

## 17.176 I4/cxx/bits/bst\_iter.h File Reference

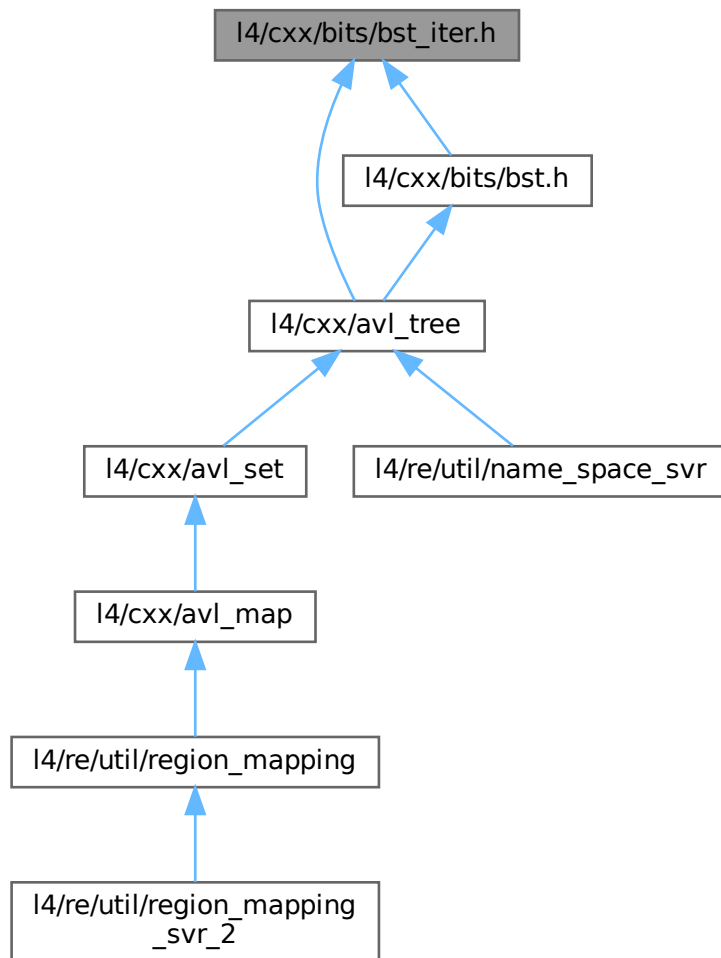
AVL tree.

```
#include "bst_base.h"
```

Include dependency graph for bst\_iter.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace `cxx`  
*Our C++ library.*
- namespace `cxx::Bits`  
*Internal helpers for the cxx package.*

### 17.176.1 Detailed Description

AVL tree.

Definition in file [bst\\_iter.h](#).

## 17.177 bst\_iter.h

[Go to the documentation of this file.](#)

```

00001 // vi:ft=cpp
00006 /*
00007 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>,
00008 * Carsten Weinhold <weinhold@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013
00014 #pragma once
00015
00016 #include "bst_base.h"
00017
00018 namespace cxx { namespace Bits {
00019
00020 template< typename Node, typename Node_op >
00021 class __Bst_iter_b
00022 {
00023 protected:
00024 typedef Direction Dir;
00025 Node const *_n;
00026 Node const *_r;
00027
00028 __Bst_iter_b() : _n(0), _r(0) {}
00029
00030 __Bst_iter_b(Node const *t)
00031 : _n(t), _r(_n)
00032 { _downmost(); }
00033
00034 __Bst_iter_b(Node const *t, Node const *r)
00035 : _n(t), _r(r)
00036 {}
00037
00038 inline void _downmost();
00039
00040 inline void inc();
00041
00042 public:
00043 bool operator == (__Bst_iter_b const &o) const { return _n == o._n; }
00044 bool operator != (__Bst_iter_b const &o) const { return _n != o._n; }
00045 };
00046
00047 template< typename Node, typename Node_type, typename Node_op >
00048 class __Bst_iter : public __Bst_iter_b<Node, Node_op>
00049 {
00050 private:
00051 typedef __Bst_iter_b<Node, Node_op> Base;
00052 using Base::_n;
00053 using Base::_r;
00054 using Base::inc;
00055
00056 public:
00057 __Bst_iter() {}
00058
00059 __Bst_iter(Node const *t) : Base(t) {}
00060 __Bst_iter(Node const *t, Node const *r) : Base(t, r) {}
00061
00062 // template<typename Key2>
00063 __Bst_iter(Base const &o) : Base(o) {}
00064
00065 Node_type &operator * () const { return *const_cast<Node *>(_n); }
00066 Node_type *operator -> () const { return const_cast<Node *>(_n); }
00067 __Bst_iter &operator ++ () { inc(); return *this; }
00068 __Bst_iter &operator ++ (int)
00069 { __Bst_iter tmp = *this; inc(); return tmp; }
00070 };
00071
00072 //-----
00073 /* IMPLEMENTATION: __Bst_iter_b */
00074
00075 template< typename Node, typename Node_op>
00076 inline
00077 void __Bst_iter_b<Node, Node_op>::_downmost()
00078 {
00079 while (_n)
00080 {
00081 Node *n = Node_op::child(_n, Dir::L);
00082 if (n)
00083 _n = n;
00084 else

```

```

00134 return;
00135 }
00136 }
00137
00138 template< typename Node, typename Node_op>
00139 void __Bst_iter_b<Node, Node_op>::inc()
00140 {
00141 if (!_n)
00142 return;
00143
00144 if (_n == _r)
00145 {
00146 _r = _n = Node_op::child(_r, Dir::R);
00147 _downmost();
00148 return;
00149 }
00150
00151 if (Node_op::child(_n, Dir::R))
00152 {
00153 _n = Node_op::child(_n, Dir::R);
00154 _downmost();
00155 return;
00156 }
00157
00158 Node const *q = _r;
00159 Node const *p = _r;
00160 while (1)
00161 {
00162 if (Node_op::cmp(_n, q))
00163 {
00164 p = q;
00165 q = Node_op::child(q, Dir::L);
00166 }
00167 else if (_n == q || Node_op::child(q, Dir::R) == _n)
00168 {
00169 _n = p;
00170 return;
00171 }
00172 else
00173 q = Node_op::child(q, Dir::R);
00174 }
00175 }
00176
00177 }}

```

## 17.178 list\_basics.h

```

00001 /*
00002 * (c) 2011 Alexander Warg <warg@os.inf.tu-dresden.de>
00003 * economic rights: Technische Universität Dresden (Germany)
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #pragma once
00009
00010 namespace cxx { namespace Bits {
00011
00012 template< typename T >
00013 class List_iterator_end_ptr
00014 {
00015 private:
00016 template< typename U > friend class Basic_list;
00017 static void *_end;
00018 };
00019
00020 template< typename T >
00021 void *List_iterator_end_ptr<T>::_end;
00022
00023 template< typename VALUE_T, typename TYPE >
00024 struct Basic_list_policy
00025 {
00026 typedef VALUE_T *Value_type;
00027 typedef VALUE_T const *Const_value_type;
00028 typedef TYPE **Type;
00029 typedef TYPE *Const_type;
00030 typedef TYPE *Head_type;
00031 typedef TYPE Item_type;
00032
00033 static Type next(Type c) { return &(*c)->_n; }
00034 static Const_type next(Const_type c) { return c->_n; }
00035 };
00036

```



```

00038 template< typename POLICY >
00039 class Basic_list
00040 {
00041 Basic_list(Basic_list const &) = delete;
00042 void operator = (Basic_list const &) = delete;
00043
00044 public:
00045 typedef typename POLICY::Value_type Value_type;
00046 typedef typename POLICY::Const_value_type Const_value_type;
00047
00048 class Iterator
00049 {
00050 private:
00051 typedef typename POLICY::Type Internal_type;
00052
00053 public:
00054 typedef typename POLICY::Value_type value_type;
00055 typedef typename POLICY::Value_type Value_type;
00056
00057 Value_type operator * () const { return static_cast<Value_type>(*_c); }
00058 Value_type operator -> () const { return static_cast<Value_type>(*_c); }
00059 Iterator operator ++ () { _c = POLICY::next(_c); return *this; }
00060
00061 bool operator == (Iterator const &o) const { return *_c == *o._c; }
00062 bool operator != (Iterator const &o) const { return !operator == (o); }
00063
00064 Iterator() : _c(__end()) {}
00065
00066 private:
00067 friend class Basic_list;
00068 static Internal_type __end()
00069 {
00070 union X { Internal_type l; void **v; } z;
00071 z.v = &Bits::List_iterator_end_ptr<void>::_end;
00072 return z.l;
00073 }
00074
00075 explicit Iterator(Internal_type i) : _c(i) {}
00076
00077 Internal_type _c;
00078 };
00079
00080 class Const_iterator
00081 {
00082 private:
00083 typedef typename POLICY::Const_type Internal_type;
00084
00085 public:
00086 typedef typename POLICY::Value_type value_type;
00087 typedef typename POLICY::Value_type Value_type;
00088
00089 Value_type operator * () const { return static_cast<Value_type>(_c); }
00090 Value_type operator -> () const { return static_cast<Value_type>(_c); }
00091 Const_iterator operator ++ () { _c = POLICY::next(_c); return *this; }
00092
00093 friend bool operator == (Const_iterator const &lhs, Const_iterator const &rhs)
00094 { return lhs._c == rhs._c; }
00095 friend bool operator != (Const_iterator const &lhs, Const_iterator const &rhs)
00096 { return lhs._c != rhs._c; }
00097
00098 Const_iterator() {}
00099 Const_iterator(Iterator const &o) : _c(*o) {}
00100
00101 private:
00102 friend class Basic_list;
00103
00104 explicit Const_iterator(Internal_type i) : _c(i) {}
00105
00106 Internal_type _c;
00107 };
00108
00109 // BSS allocation
00110 explicit Basic_list(bool) {}
00111 Basic_list() : _f(0) {}
00112
00113 Basic_list(Basic_list &o) : _f(o._f)
00114 {
00115 o.clear();
00116 }
00117
00118 Basic_list &operator = (Basic_list &o)
00119 {
00120 _f = o._f;
00121 o.clear();
00122 return *this;
00123 }
00124

```

```

00126 bool empty() const { return !_f; }
00128 Value_type front() const { return static_cast<Value_type>(_f); }
00129
00135 void clear() { _f = 0; }
00136
00138 Iterator begin() { return Iterator(&_f); }
00140 Const_iterator begin() const { return Const_iterator(_f); }
00148 static Const_iterator iter(Const_value_type c) { return Const_iterator(c); }
00150 Const_iterator end() const { return Const_iterator(nullptr); }
00152 Iterator end() { return Iterator(); }
00153
00154 protected:
00155 static typename POLICY::Type __get_internal(Iterator const &i) { return i._c; }
00156 static Iterator __iter(typename POLICY::Type c) { return Iterator(c); }
00157
00159 typename POLICY::Head_type _f;
00160 };
00161
00162 }}
00163

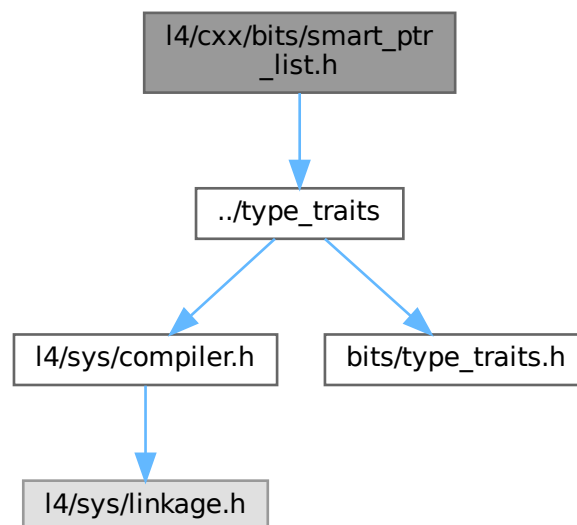
```

## 17.179 I4/cxx/bits/smart\_ptr\_list.h File Reference

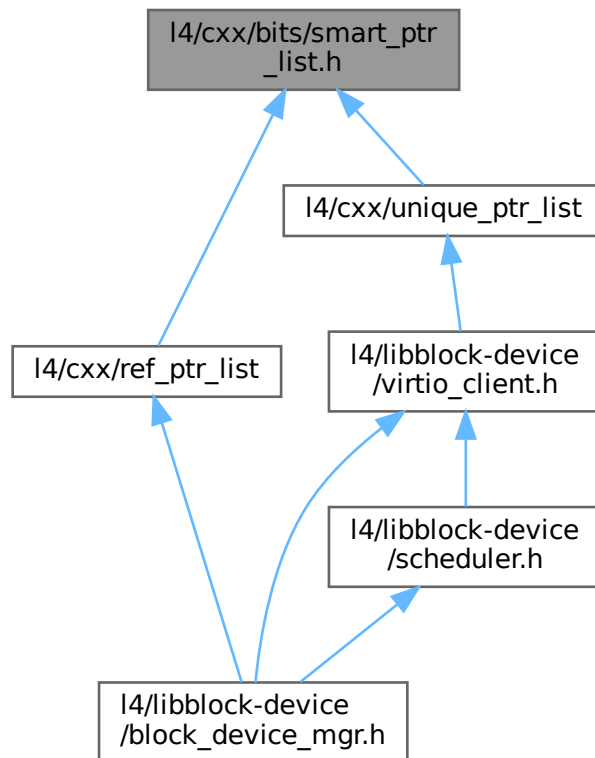
Implementation of a list of smart-pointer-managed objects.

```
#include "../type_traits"
```

Include dependency graph for smart\_ptr\_list.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class [cxx::Bits::Smart\\_ptr\\_list\\_item< T, STORE\\_T >](#)  
*List item for an arbitrary item in a [Smart\\_ptr\\_list](#).*
- class [cxx::Bits::Smart\\_ptr\\_list< ITEM >](#)  
*List of smart-pointer-managed objects.*

## Namespaces

- namespace [cxx](#)  
*Our C++ library.*
- namespace [cxx::Bits](#)  
*Internal helpers for the cxx package.*

### 17.179.1 Detailed Description

Implementation of a list of smart-pointer-managed objects.

Definition in file [smart\\_ptr\\_list.h](#).

## 17.180 smart\_ptr\_list.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * Copyright (C) 2018, 2022, 2024 Kernkonzept GmbH.
00007 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #pragma once
00012
00013 #include "../type_traits"
00014
00015 namespace cxx { namespace Bits {
00016
00026 template <typename T, typename STORE_T>
00027 class Smart_ptr_list_item
00028 {
00029 using Value_type = T;
00030 using Storage_type = STORE_T;
00031
00032 template<typename U> friend class Smart_ptr_list;
00033 Storage_type _n;
00034 };
00035
00045 template <typename ITEM>
00046 class Smart_ptr_list
00047 {
00048 using Value_type = typename ITEM::Value_type;
00049 using Next_type = typename ITEM::Storage_type;
00050
00051 public:
00052 class Iterator
00053 {
00054 public:
00055 Iterator() : _c(nullptr) {}
00056
00057 Value_type *operator * () const { return _c; }
00058 Value_type *operator -> () const { return _c; }
00059
00060 Iterator operator ++ ()
00061 {
00062 _c = _c->_n.get();
00063 return *this;
00064 }
00065
00066 bool operator == (Iterator const &o) const { return _c == o._c; }
00067 bool operator != (Iterator const &o) const { return !operator == (o); }
00068
00069 private:
00070 friend class Smart_ptr_list;
00071
00072 explicit Iterator(Value_type *i) : _c(i) {}
00073
00074 Value_type *_c;
00075 };
00076
00077 class Const_iterator
00078 {
00079 public:
00080 Const_iterator() : _c(nullptr) {}
00081
00082 Value_type const *operator * () const { return _c; }
00083 Value_type const *operator -> () const { return _c; }
00084
00085 Const_iterator operator ++ ()
00086 {
00087 _c = _c->_n.get();
00088 return *this;
00089 }
00090
00091 bool operator == (Const_iterator const &o) const { return _c == o._c; }
00092 bool operator != (Const_iterator const &o) const { return !operator == (o); }
00093
00094 private:
00095 friend class Smart_ptr_list;
00096
00097 explicit Const_iterator(Value_type const *i) : _c(i) {}
00098
00099 Value_type const *_c;
00100 };
00101
00102 Smart_ptr_list() : _b(&_f) {}
00103

```

```

00105 void push_front(Next_type &&e)
00106 {
00107 e->_n = cxx::move(this->_f);
00108 this->_f = cxx::move(e);
00109
00110 if (_b == &_f)
00111 _b = &(_f->_n);
00112 }
00113
00115 void push_front(Next_type const &e)
00116 {
00117 e->_n = cxx::move(this->_f);
00118 this->_f = e;
00119
00120 if (_b == &_f)
00121 _b = &(_f->_n);
00122 }
00123
00125 void push_back(Next_type &&e)
00126 {
00127 *_b = cxx::move(e);
00128 _b = &(*_b->_n);
00129 }
00130
00132 void push_back(Next_type const &e)
00133 {
00134 *_b = e;
00135 _b = &(*_b->_n);
00136 }
00137
00139 Value_type *front() const
00140 { return _f.get(); }
00141
00149 Next_type pop_front()
00150 {
00151 Next_type ret = cxx::move(_f);
00152
00153 if (ret)
00154 _f = cxx::move(ret->_n);
00155
00156 if (!_f)
00157 _b = &_f;
00158
00159 return ret;
00160 }
00161
00163 bool empty() const
00164 { return !_f; }
00165
00166 Iterator begin() { return Iterator(_f.get()); }
00167 Iterator end() { return Iterator(); }
00168
00169 Const_iterator begin() const { return Const_iterator(_f.get()); }
00170 Const_iterator end() const { return Const_iterator(); }
00171
00172 Const_iterator cbegin() const { return const_iterator(_f.get()); }
00173 Const_iterator cend() const { return Const_iterator(); }
00174
00175 private:
00176 Next_type _f;
00177 Next_type *_b;
00178 };
00179
00180 }

```

## 17.181 type\_traits.h

```

00001 // vi:ft=cpp
00002 /*
00003 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>,
00004 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009
00010 #pragma once
00011
00012 namespace cxx {
00013
00014 class Null_type;
00015
00016 template< bool flag, typename T, typename F >

```

```

00017 class Select
00018 {
00019 public:
00020 typedef T Type;
00021 };
00022
00023 template< typename T, typename F >
00024 class Select< false, T, F >
00025 {
00026 public:
00027 typedef F Type;
00028 };
00029
00030
00031
00032 template< typename T, typename U >
00033 class Conversion
00034 {
00035 typedef char S;
00036 class B { char dummy[2]; };
00037 static S test(U);
00038 static B test(...);
00039 static T make_T();
00040 public:
00041 enum
00042 {
00043 exists = sizeof(test(make_T())) == sizeof(S),
00044 two_way = exists && Conversion<U,T>::exists,
00045 exists_2_way = two_way,
00046 same_type = false
00047 };
00048 };
00049
00050 template< >
00051 class Conversion<void, void>
00052 {
00053 public:
00054 enum { exists = 1, two_way = 1, exists_2_way = two_way, same_type = 1 };
00055 };
00056
00057 template< typename T >
00058 class Conversion<T, T>
00059 {
00060 public:
00061 enum { exists = 1, two_way = 1, exists_2_way = two_way, same_type = 1 };
00062 };
00063
00064 template< typename T >
00065 class Conversion<void, T>
00066 {
00067 public:
00068 enum { exists = 0, two_way = 0, exists_2_way = two_way, same_type = 0 };
00069 };
00070
00071 template< typename T >
00072 class Conversion<T, void>
00073 {
00074 public:
00075 enum { exists = 0, two_way = 0, exists_2_way = two_way, same_type = 0 };
00076 };
00077
00078 template< int I >
00079 class Int_to_type
00080 {
00081 public:
00082 enum { i = I };
00083 };
00084
00085 namespace TT
00086 {
00087 template< typename U > class Pointer_traits
00088 {
00089 public:
00090 typedef Null_type Pointee;
00091 enum { value = false };
00092 };
00093
00094 template< typename U > class Pointer_traits< U* >
00095 {
00096 public:
00097 typedef U Pointee;
00098 enum { value = true };
00099 };
00100
00101 template< typename U > struct Ref_traits
00102 {
00103 enum { value = false };

```

```

00104 typedef U Referee;
00105 };
00106
00107 template< typename U > struct Ref_traits<U&>
00108 {
00109 enum { value = true };
00110 typedef U Referee;
00111 };
00112
00113
00114 template< typename U > struct Add_ref { typedef U &Type; };
00115 template< typename U > struct Add_ref<U&> { typedef U Type; };
00116
00117 template< typename U > struct PMF_traits { enum { value = false }; };
00118 template< typename U, typename F > struct PMF_traits<U F::*>
00119 { enum { value = true }; };
00120
00121
00122 template< typename U > class Is_unsigned { public: enum { value = false }; };
00123 template<> class Is_unsigned<unsigned> { public: enum { value = true }; };
00124 template<> class Is_unsigned<unsigned char> {
00125 public: enum { value = true };
00126 };
00127 template<> class Is_unsigned<unsigned short> {
00128 public: enum { value = true };
00129 };
00130 template<> class Is_unsigned<unsigned long> {
00131 public: enum { value = true };
00132 };
00133 template<> class Is_unsigned<unsigned long long> {
00134 public: enum { value = true };
00135 };
00136
00137 template< typename U > class Is_signed { public: enum { value = false }; };
00138 template<> class Is_signed<signed char> { public: enum { value = true }; };
00139 template<> class Is_signed<signed short> { public: enum { value = true }; };
00140 template<> class Is_signed<signed> { public: enum { value = true }; };
00141 template<> class Is_signed<signed long> { public: enum { value = true }; };
00142 template<> class Is_signed<signed long long> {
00143 public: enum { value = true };
00144 };
00145
00146 template< typename U > class Is_int { public: enum { value = false }; };
00147 template<> class Is_int< char > { public: enum { value = true }; };
00148 template<> class Is_int< bool > { public: enum { value = true }; };
00149 template<> class Is_int< wchar_t > { public: enum { value = true }; };
00150
00151 template< typename U > class Is_float { public: enum { value = false }; };
00152 template<> class Is_float< float > { public: enum { value = true }; };
00153 template<> class Is_float< double > { public: enum { value = true }; };
00154 template<> class Is_float< long double > { public: enum { value = true }; };
00155
00156 template<typename T> class Const_traits
00157 {
00158 public:
00159 enum { value = false };
00160 typedef T Type;
00161 typedef const T Const_type;
00162 };
00163
00164 template<typename T> class Const_traits<const T>
00165 {
00166 public:
00167 enum { value = true };
00168 typedef T Type;
00169 typedef const T Const_type;
00170 };
00171 };
00172
00173 template< typename T >
00174 class Type_traits
00175 {
00176 public:
00177
00178 enum
00179 {
00180 is_unsigned = TT::Is_unsigned<T>::value,
00181 is_signed = TT::Is_signed<T>::value,
00182 is_int = TT::Is_int<T>::value,
00183 is_float = TT::Is_float<T>::value,
00184 is_pointer = TT::Pointer_traits<T>::value,
00185 is_pointer_to_member = TT::PMF_traits<T>::value,
00186 is_reference = TT::Ref_traits<T>::value,
00187 is_scalar = is_unsigned || is_signed || is_int || is_pointer
00188 || is_pointer_to_member || is_reference,
00189 is_fundamental = is_unsigned || is_signed || is_float
00190 || Conversion<T, void>::same_type,

```

```

00191 is_const = TT::Const_traits<T>::value,
00192
00193 alignment =
00194 (sizeof(T) >= sizeof(unsigned long)
00195 ? sizeof(unsigned long)
00196 : (sizeof(T) >= sizeof(unsigned)
00197 ? sizeof(unsigned)
00198 : (sizeof(T) >= sizeof(short)
00199 ? sizeof(short)
00200 : 1)))
00201 };
00202
00203 typedef typename Select<is_scalar, T, typename TT::Add_ref<typename
TT::Const_traits<T>::Const_type>::Type>::Type Param_type;
00204 typedef typename TT::Pointer_traits<T>::Pointee Pointee_type;
00205 typedef typename TT::Ref_traits<T>::Referee Referee_type;
00206 typedef typename TT::Const_traits<T>::Type Non_const_type;
00207 typedef typename TT::Const_traits<T>::Const_type Const_type;
00208
00209 static unsigned long align(unsigned long a)
00210 {
00211 return (a + static_cast<unsigned long>(alignment) - 1UL)
00212 & ~(static_cast<unsigned long>(alignment) - 1UL);
00213 }
00214 };
00215
00216
00217 };
00218
00219
00220

```

## 17.182 dlist

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2011 Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00009 #pragma once
00010
00011 namespace cxx {
00012
00013 class D_list_item
00014 {
00015 public:
00016 constexpr D_list_item() : _dli_next(nullptr), _dli_prev(nullptr) {}
00017
00018 D_list_item(D_list_item const &) = delete;
00019 void operator = (D_list_item const &) = delete;
00020
00021 private:
00022 friend struct D_list_item_policy;
00023
00024 D_list_item *_dli_next, *_dli_prev;
00025 };
00026
00027 struct D_list_item_policy
00028 {
00029 typedef D_list_item Item;
00030 static D_list_item *&prev(D_list_item *e) { return e->_dli_prev; }
00031 static D_list_item *&next(D_list_item *e) { return e->_dli_next; }
00032 static D_list_item *prev(D_list_item const *e) { return e->_dli_prev; }
00033 static D_list_item *next(D_list_item const *e) { return e->_dli_next; }
00034 };
00035
00036 template< typename T >
00037 struct Sd_list_head_policy
00038 {
00039 typedef T *Head_type;
00040 static T *head(Head_type h) { return h; }
00041 static void set_head(Head_type &h, T *v) { h = v; }
00042 };
00043
00044 template<
00045 typename T,
00046 typename C = D_list_item_policy
00047 >
00048 class D_list_cyclic
00049 {

```



```

00050 protected:
00051 template< typename VALUE, typename ITEM >
00052 class __Iterator
00053 {
00054 public:
00055 typedef VALUE *Value_type;
00056 typedef VALUE *value_type;
00057
00058 __Iterator() {}
00059
00060 bool operator == (__Iterator const &o) const
00061 { return _c == o._c; }
00062
00063 bool operator != (__Iterator const &o) const
00064 { return _c != o._c; }
00065
00066 __Iterator &operator ++ ()
00067 {
00068 _c = C::next(_c);
00069 return *this;
00070 }
00071
00072 __Iterator &operator -- ()
00073 {
00074 _c = C::prev(_c);
00075 return *this;
00076 }
00077
00078 Value_type operator * () const { return static_cast<Value_type>(_c); }
00079 Value_type operator -> () const { return static_cast<Value_type>(_c); }
00080
00081 protected:
00082 friend class D_list_cyclic;
00083
00084 explicit __Iterator(ITEM *s) : _c(s) {}
00085
00086 ITEM *_c;
00087 };
00088
00089 public:
00090 typedef T *Value_type;
00091 typedef T *value_type;
00092 typedef __Iterator<T, typename C::Item> Iterator;
00093 typedef Iterator Const_iterator;
00094
00095 static void remove(T *e)
00096 {
00097 C::next(C::prev(e)) = C::next(e);
00098 C::prev(C::next(e)) = C::prev(e);
00099 C::next(e) = nullptr;
00100 }
00101
00102 static Iterator erase(Iterator const &e)
00103 {
00104 typename C::Item *n = C::next(*e);
00105 remove(*e);
00106 return __iter(n);
00107 }
00108
00109 static Iterator iter(T const *e) { return Iterator(const_cast<T*>(e)); }
00110
00111 static bool in_list(T const *e) { return C::next(const_cast<T*>(e)); }
00112 static bool has_sibling(T const *e) { return C::next(const_cast<T*>(e)) != e; }
00113
00114 static Iterator insert_after(T *e, Iterator const &pos)
00115 {
00116 C::prev(e) = pos._c;
00117 C::next(e) = C::next(pos._c);
00118 C::prev(C::next(pos._c)) = e;
00119 C::next(pos._c) = e;
00120 return pos;
00121 }
00122
00123 static Iterator insert_before(T *e, Iterator const &pos)
00124 {
00125 C::next(e) = pos._c;
00126 C::prev(e) = C::prev(pos._c);
00127 C::next(C::prev(pos._c)) = e;
00128 C::prev(pos._c) = e;
00129 return pos;
00130 }
00131
00132 protected:
00133 static void self_insert(typename C::Item *e)
00134 { C::next(e) = C::prev(e) = e; }
00135
00136 static void remove_last(T *e)

```

```

00137 { C::next(e) = nullptr; }
00138
00145 static void splice_heads(Const_iterator pos, typename C::Item *other_list)
00146 {
00147 typename C::Item *ins_next = pos._C;
00148 typename C::Item *ins_prev = C::prev(pos._C);
00149 typename C::Item *other_head = C::next(other_list);
00150 typename C::Item *other_tail = C::prev(other_list);
00151
00152 C::next(ins_prev) = other_head;
00153 C::prev(other_head) = ins_prev;
00154 C::prev(ins_next) = other_tail;
00155 C::next(other_tail) = ins_next;
00156 }
00157
00158 static Iterator __iter(typename C::Item *e) { return Iterator(e); }
00159 };
00160
00161 template<
00162 typename T,
00163 typename C = D_list_item_policy,
00164 typename H = Sd_list_head_policy<T>,
00165 bool BSS = false
00166 >
00167 class Sd_list : public D_list_cyclic<T, C>
00168 {
00169 private:
00170 typedef D_list_cyclic<T, C> Base;
00171
00172 public:
00173 class Iterator : public Base::Iterator
00174 {
00175 public:
00176 Iterator &operator ++ ()
00177 {
00178 if (this->_c)
00179 Base::Iterator::operator ++ ();
00180
00181 if (this->_c == _h)
00182 this->_c = nullptr;
00183
00184 return *this;
00185 }
00186
00187 Iterator &operator -- () = delete;
00188
00189 private:
00190 friend class Sd_list;
00191
00192 explicit Iterator(T *h) : Base::Iterator(h), _h(h) {}
00193 typename C::Item *_h;
00194 };
00195
00196 class R_iterator : public Base::Iterator
00197 {
00198 public:
00199 R_iterator &operator ++ ()
00200 {
00201 if (this->_c)
00202 Base::Iterator::operator -- ();
00203
00204 if (this->_c == _h)
00205 this->_c = nullptr;
00206
00207 return *this;
00208 }
00209
00210 R_iterator &operator -- () = delete;
00211
00212 private:
00213 friend class Sd_list;
00214
00215 explicit R_iterator(T *h) : Base::Iterator(h), _h(h) {}
00216 typename C::Item *_h;
00217 };
00218
00219 //typedef typename Base::Iterator Iterator;
00220 enum Pos
00221 { Back, Front };
00222
00223 Sd_list()
00224 {
00225 if (!BSS)
00226 H::set_head(_f, nullptr);
00227 }
00228
00229 bool empty() const { return !H::head(_f); }

```

```

00230 T *front() const { return H::head(_f); }
00231
00232 void remove(T *e)
00233 {
00234 T *h = H::head(_f);
00235 if (e == C::next(e)) // must be the last
00236 {
00237 Base::remove_last(e);
00238 H::set_head(_f, nullptr);
00239 return;
00240 }
00241
00242 if (e == H::head(_f))
00243 H::set_head(_f, static_cast<T*>(C::next(h)));
00244
00245 Base::remove(e);
00246 }
00247
00248 Iterator erase(Iterator const &e)
00249 {
00250 Iterator next = e;
00251 ++next;
00252
00253 remove(*e);
00254 return next;
00255 }
00256
00257 void push(T *e, Pos pos)
00258 {
00259 T *h = H::head(_f);
00260 if (!h)
00261 {
00262 Base::self_insert(e);
00263 H::set_head(_f, e);
00264 }
00265 else
00266 {
00267 Base::insert_before(e, this->iter(h));
00268 if (pos == Front)
00269 H::set_head(_f, e);
00270 }
00271 }
00272
00273 void push_back(T *e) { push(e, Back); }
00274 void push_front(T *e) { push(e, Front); }
00275 void rotate_to(T *h) { H::set_head(_f, h); }
00276
00277 typename H::Head_type const &head() const { return _f; }
00278 typename H::Head_type &head() { return _f; }
00279
00280 Iterator begin() { return Iterator(H::head(_f)); }
00281 Iterator end() { return Iterator(nullptr); }
00282
00283 R_iterator rbegin()
00284 {
00285 if (head())
00286 return R_iterator(static_cast<T*>(C::prev(H::head(_f))));
00287 return R_iterator(nullptr);
00288 }
00289 R_iterator rend() { return R_iterator(nullptr); }
00290
00291 private:
00292 Sd_list(Sd_list const &);
00293 void operator = (Sd_list const &);
00294
00295 typename H::Head_type _f;
00296 };
00297
00298 template<
00299 typename T,
00300 typename C = D_list_item_policy,
00301 bool BSS = false
00302 >
00303 class D_list : public D_list_cyclic<T, C>
00304 {
00305 private:
00306 typedef D_list_cyclic<T, C> Base;
00307 typedef typename C::Item Internal_type;
00308
00309 public:
00310 enum Pos
00311 { Back, Front };
00312
00313 typedef typename Base::Iterator Iterator;
00314 typedef typename Base::Const_iterator Const_iterator;
00315 typedef T* value_type;
00316 typedef T* Value_type;

```

```

00317
00318 D_list() { this->self_insert(&_h); }
00319 ~D_list() { clear(); }
00320
00321 D_list(D_list &o)
00322 {
00323 if (o.empty())
00324 {
00325 this->self_insert(&_h);
00326 }
00327 else
00328 {
00329 Internal_type *p = C::prev(&o._h);
00330 Internal_type *n = C::next(&o._h);
00331 C::prev(&_h) = p;
00332 C::next(&_h) = n;
00333 C::next(p) = &_h;
00334 C::prev(n) = &_h;
00335 o.self_insert(&o._h);
00336 }
00337 }
00338
00339 D_list &operator=(D_list &o)
00340 {
00341 if (&o == this)
00342 return *this;
00343
00344 clear();
00345
00346 if (!o.empty())
00347 {
00348 Internal_type *p = C::prev(&o._h);
00349 Internal_type *n = C::next(&o._h);
00350 C::prev(&_h) = p;
00351 C::next(&_h) = n;
00352 C::next(p) = &_h;
00353 C::prev(n) = &_h;
00354 o.self_insert(&o._h);
00355 }
00356 }
00357
00358 D_list(D_list const &) = delete;
00359 void operator = (D_list const &) = delete;
00360
00361 void splice(Const_iterator pos, D_list &&other)
00362 {
00363 if (other.empty())
00364 return;
00365
00366 Base::splice_heads(pos, &other._h);
00367 other.self_insert(&other._h);
00368 }
00369
00370 bool empty() const { return C::next(&_h) == &_h; }
00371
00372 static void remove(T *e) { Base::remove(e); }
00373 Iterator erase(Iterator const &e) { return Base::erase(e); }
00374
00375 void clear()
00376 {
00377 // Just clear the _dli_next pointers of all elements. It is the indicator
00378 // that an element is not on a list.
00379 Internal_type *i = C::next(&_h);
00380 while (i != &_h)
00381 {
00382 Internal_type *d = i;
00383 i = C::next(i);
00384 C::next(d) = nullptr;
00385 }
00386
00387 this->self_insert(&_h);
00388 }
00389
00390 void push(T *e, Pos pos)
00391 {
00392 if (pos == Front)
00393 Base::insert_after(e, end());
00394 else
00395 Base::insert_before(e, end());
00396 }
00397
00398 void push_back(T *e) { push(e, Back); }
00399 void push_front(T *e) { push(e, Front); }
00400
00401 T *pop_back()
00402 {
00403 T *ret = *(end()--);

```

```

00409 remove(ret);
00410 return ret;
00411 }
00412
00418 T *pop_front()
00419 {
00420 T *ret = *begin();
00421 remove(ret);
00422 return ret;
00423 }
00424
00425 Iterator begin() const { return this->__iter(C::next(const_cast<Internal_type *>(&_h))); }
00426 Iterator end() const { return this->__iter(const_cast<Internal_type *>(&_h)); }
00427
00428 bool has_sibling(T const *e)
00429 {
00430 return C::next(const_cast<T*>(e)) != &_h
00431 || C::prev(const_cast<T*>(e)) != &_h;
00432 }
00433
00434 private:
00435 Internal_type _h;
00436 };
00437
00438 }
00439

```

## 17.183 l4/cxx/exceptions File Reference

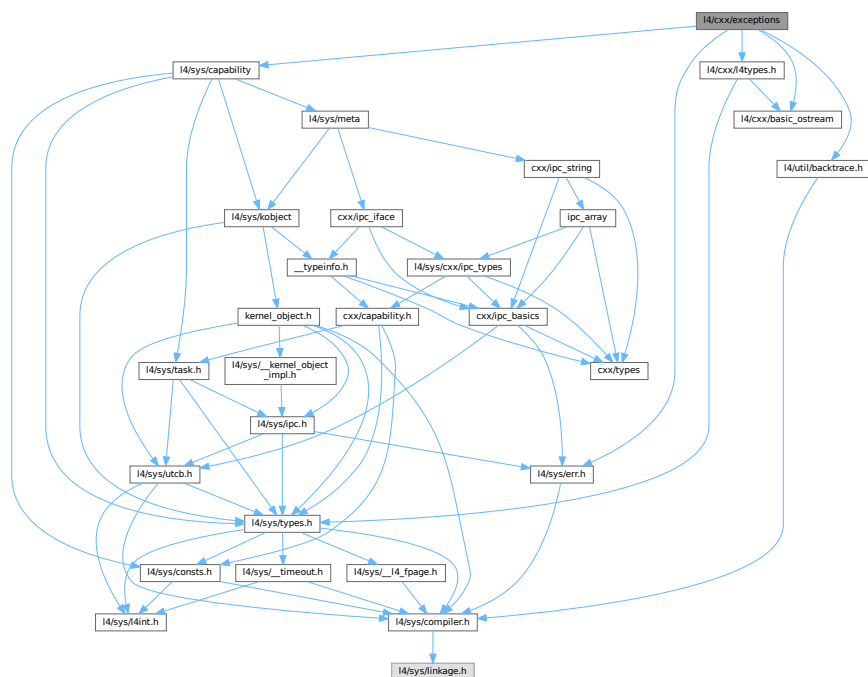
Base exceptions.

```

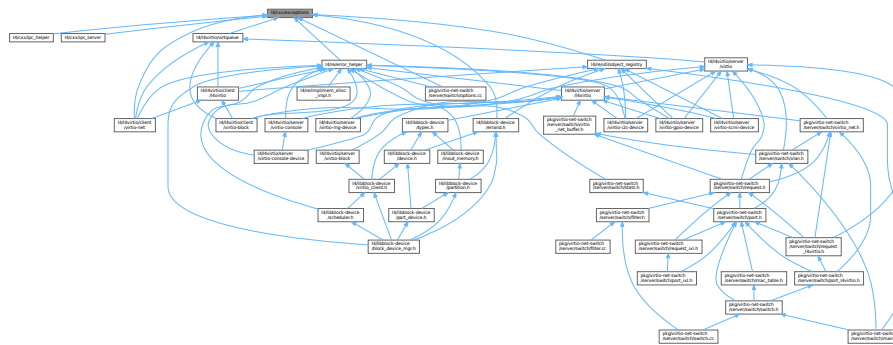
#include <l4/cxx/l4types.h>
#include <l4/cxx/basic_ostream>
#include <l4/sys/err.h>
#include <l4/sys/capability>
#include <l4/util/backtrace.h>

```

Include dependency graph for exceptions:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class [L4::Exception\\_tracer](#)  
*Back-trace support for exceptions.*
- class [L4::Base\\_exception](#)  
*Base class for all exceptions, thrown by the [L4Re](#) framework.*
- class [L4::Runtime\\_error](#)  
*Exception for an abstract runtime error.*
- class [L4::Out\\_of\\_memory](#)  
*Exception signalling insufficient memory.*
- class [L4::Element\\_already\\_exists](#)  
*Exception for duplicate element insertions.*
- class [L4::Unknown\\_error](#)  
*Exception for an unknown condition.*
- class [L4::Element\\_not\\_found](#)  
*Exception for a failed lookup (element not found).*
- class [L4::Invalid\\_capability](#)  
*Indicates that an invalid object was invoked.*
- class [L4::Com\\_error](#)  
*Error conditions during IPC.*
- class [L4::Bounds\\_error](#)  
*Access out of bounds.*

## Namespaces

- namespace [L4](#)  
*[L4](#) low-level kernel interface.*

## Macros

- `#define L4_CXX_EXCEPTION_BACKTRACE 20`  
*Number of instruction pointers in backtrace.*

## 17.183.1 Detailed Description

Base exceptions.

Definition in file [exceptions](#).

## 17.184 exceptions

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00007 /*
00008 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00009 * Alexander Warg <warg@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #pragma once
00015
00016 #include <l4/cxx/l4types.h>
00017 #include <l4/cxx/basic_ostream>
00018 #include <l4/sys/err.h>
00019 #include <l4/sys/capability>
00020
00021
00027
00028 #ifndef L4_CXX_NO_EXCEPTION_BACKTRACE
00029 # define L4_CXX_EXCEPTION_BACKTRACE 20
00030 #endif
00031
00032 #if defined(L4_CXX_EXCEPTION_BACKTRACE)
00033 #include <l4/util/backtrace.h>
00034 #endif
00035
00037 namespace L4
00038 {
00051 class Exception_tracer
00052 {
00053 #if defined(L4_CXX_EXCEPTION_BACKTRACE)
00054 private:
00055 void *_pc_array[L4_CXX_EXCEPTION_BACKTRACE];
00056 int _frame_cnt;
00057
00058 protected:
00062 #if defined(__PIC__)
00063 Exception_tracer() noexcept : _frame_cnt(0) {}
00064 #else
00065 Exception_tracer() noexcept
00066 : _frame_cnt(l4util_backtrace(_pc_array, L4_CXX_EXCEPTION_BACKTRACE)) {}
00067 #endif
00068
00069 public:
00073 void const *const *pc_array() const noexcept { return _pc_array; }
00077 int frame_count() const noexcept { return _frame_cnt; }
00078 #else
00079 protected:
00083 Exception_tracer() noexcept {}
00084
00085 public:
00089 void const *const *pc_array() const noexcept { return 0; }
00093 int frame_count() const noexcept { return 0; }
00094 #endif
00095 };
00096
00105 class Base_exception : public Exception_tracer
00106 {
00107 protected:
00109 Base_exception() noexcept {}
00110
00111 public:
00115 virtual char const *str() const noexcept = 0;
00116
00118 virtual ~Base_exception() noexcept {}
00119 };
00120
00128 class Runtime_error : public Base_exception
00129 {
00130 private:

```

```

00131 long _errno;
00132 char _extra[80];
00133
00134 public:
00141 explicit Runtime_error(long err_no, char const *extra = 0) noexcept
00142 : _errno(err_no)
00143 {
00144 if (!extra)
00145 _extra[0] = 0;
00146 else
00147 {
00148 unsigned i = 0;
00149 for (; i < sizeof(_extra) && extra[i]; ++i)
00150 _extra[i] = extra[i];
00151 _extra[i < sizeof(_extra) ? i : sizeof(_extra) - 1] = 0;
00152 }
00153 }
00154 char const *str() const noexcept override
00155 { return l4sys_errtostr(_errno); }
00156
00162 char const *extra_str() const { return _extra; }
00163 ~Runtime_error() noexcept {}
00164
00170 long err_no() const noexcept { return _errno; }
00171 };
00172
00177 class Out_of_memory : public Runtime_error
00178 {
00179 public:
00181 explicit Out_of_memory(char const *extra = "") noexcept
00182 : Runtime_error(-L4_ENOMEM, extra) {}
00184 ~Out_of_memory() noexcept {}
00185 };
00186
00187
00192 class Element_already_exists : public Runtime_error
00193 {
00194 public:
00195 explicit Element_already_exists(char const *e = "") noexcept
00196 : Runtime_error(-L4_EEXIST, e) {}
00197 ~Element_already_exists() noexcept {}
00198 };
00199
00208 class Unknown_error : public Base_exception
00209 {
00210 public:
00211 Unknown_error() noexcept {}
00212 char const *str() const noexcept override { return "unknown error"; }
00213 ~Unknown_error() noexcept {}
00214 };
00215
00220 class Element_not_found : public Runtime_error
00221 {
00222 public:
00223 explicit Element_not_found(char const *e = "") noexcept
00224 : Runtime_error(-L4_ENOENT, e) {}
00225 };
00226
00234 class Invalid_capability : public Base_exception
00235 {
00236 private:
00237 Cap<void> const _o;
00238
00239 public:
00244 explicit Invalid_capability(Cap<void> const &o) noexcept : _o(o) {}
00245 template< typename T>
00246 explicit Invalid_capability(Cap<T> const &o) noexcept : _o(o.cap()) {}
00247 char const *str() const noexcept override { return "invalid object"; }
00248
00253 Cap<void> const &cap() const noexcept { return _o; }
00254 ~Invalid_capability() noexcept {}
00255 };
00256
00263 class Com_error : public Runtime_error
00264 {
00265 public:
00270 explicit Com_error(long err) noexcept : Runtime_error(err) {}
00271
00272 ~Com_error() noexcept {}
00273 };
00274
00278 class Bounds_error : public Runtime_error
00279 {
00280 public:
00281 explicit Bounds_error(char const *e = "") noexcept
00282 : Runtime_error(-L4_ERANGE, e) {}
00283 ~Bounds_error() noexcept {}

```



```

00284 };
00286 };
00287
00288 inline
00289 L4::BasicOStream &
00290 operator « (L4::BasicOStream &o, L4::Base_exception const &e)
00291 {
00292 o « "Exception: " « e.str() « ".\n";
00293 o « "Backtrace:\n";
00294 for (int i = 0; i < e.frame_count(); ++i)
00295 o « " " « L4::n_hex(l4_addr_t(e.pc_array()[i])) « '\n';
00296
00297 return o;
00298 }
00299
00300 inline
00301 L4::BasicOStream &
00302 operator « (L4::BasicOStream &o, L4::Runtime_error const &e)
00303 {
00304 o « "Exception: " « e.str();
00305 if (e.extra_str())
00306 o « ": " « e.extra_str();
00307 o « "\n" « "Backtrace:\n";
00308 for (int i = 0; i < e.frame_count(); ++i)
00309 o « " " « L4::n_hex(l4_addr_t(e.pc_array()[i])) « '\n';
00310
00311 return o;
00312 }

```

## 17.185 hlist

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2011 Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00009 #pragma once
00010
00011 #include "bits/list_basics.h"
00012 #include "type_traits"
00013
00014 namespace cxx {
00015
00021 template<typename ELEM_TYPE>
00022 class H_list_item_t
00023 {
00024 public:
00030 H_list_item_t() : _n(0), _pn(0) {}
00037 ~H_list_item_t() noexcept { l_remove(); }
00038
00039 private:
00040 H_list_item_t(H_list_item_t const &) = delete;
00041
00042 template<typename T, typename P> friend class H_list;
00043 template<typename T, typename X> friend struct Bits::Basic_list_policy;
00044
00045 void l_remove() noexcept
00046 {
00047 if (!_pn)
00048 return;
00049
00050 *_pn = _n;
00051 if (_n)
00052 _n->_pn = _pn;
00053
00054 _pn = 0;
00055 }
00056
00057 H_list_item_t *_n, **_pn;
00058 };
00059
00061 typedef H_list_item_t<void> H_list_item;
00062
00068 template< typename T, typename POLICY = Bits::Basic_list_policy< T, H_list_item> >
00069 class H_list : public Bits::Basic_list<POLICY>
00070 {
00071 private:
00072 typedef typename POLICY::Item_type Item;
00073 typedef Bits::Basic_list<POLICY> Base;
00074 H_list(H_list const &);

```

```

00075 void operator = (H_list const &);
00076
00077 public:
00078 typedef typename Base::Iterator Iterator;
00079
00080 // BSS allocation
00081 explicit H_list(bool x) : Base(x) {}
00082 H_list() : Base() {}
00083
00093 static Iterator iter(T *c) { return Base::__iter(c->Item::_pn); }
00094
00096 static bool in_list(T const *e) { return e->Item::_pn; }
00097
00099 void add(T *e)
00100 {
00101 if (this->_f)
00102 this->_f->_pn = &e->Item::_n;
00103 e->Item::_n = this->_f;
00104 e->Item::_pn = &this->_f;
00105 this->_f = static_cast<Item*>(e);
00106 }
00107
00109 void push_front(T *e) { add(e); }
00110
00116 T *pop_front()
00117 {
00118 T *r = this->front();
00119 remove(r);
00120 return r;
00121 }
00122
00133 Iterator insert(T *e, Iterator const &pred)
00134 {
00135 Item **x = &this->_f;
00136 if (pred != Base::end())
00137 x = &(*pred)->_n;
00138
00139 e->Item::_n = *x;
00140
00141 if (*x)
00142 (*x)->_pn = &(e->Item::_n);
00143
00144 e->Item::_pn = x;
00145 *x = static_cast<Item*>(e);
00146 return iter(e);
00147 }
00148
00160 static Iterator insert_after(T *e, Iterator const &pred)
00161 {
00162 Item **x = &(*pred)->_n;
00163 e->Item::_n = *x;
00164
00165 if (*x)
00166 (*x)->_pn = &(e->Item::_n);
00167
00168 e->Item::_pn = x;
00169 *x = static_cast<Item*>(e);
00170 return iter(e);
00171 }
00172
00180 static void insert_before(T *e, Iterator const &succ)
00181 {
00182 Item **x = Base::__get_internal(succ);
00183
00184 e->Item::_n = *x;
00185 e->Item::_pn = x;
00186
00187 if (*x)
00188 (*x)->_pn = &e->Item::_n;
00189
00190 *x = static_cast<Item*>(e);
00191 }
00192
00204 static void replace(T *p, T *e)
00205 {
00206 e->Item::_n = p->Item::_n;
00207 e->Item::_pn = p->Item::_pn;
00208 *(p->Item::_pn) = static_cast<Item*>(e);
00209 if (e->Item::_n)
00210 e->Item::_n->_pn = &(e->Item::_n);
00211
00212 p->Item::_pn = 0;
00213 }
00214
00220 static void remove(T *e)
00221 { e->Item::_l_remove(); }
00222

```

```

00236 static Iterator erase(Iterator const &e)
00237 { e->Item::l_remove(); return e; }
00238 };
00239
00247 template< typename T >
00248 struct H_list_t : H_list<T, Bits::Basic_list_policy< T, H_list_item_t<T> > >
00249 {
00250 H_list_t() = default;
00251 H_list_t(bool b)
00252 : H_list<T, Bits::Basic_list_policy< T, H_list_item_t<T> > >(b)
00253 {};
00254 };
00255
00256 template< typename T >
00257 class H_list_bss : public H_list<T>
00258 {
00259 public:
00260 H_list_bss() : H_list<T>(true) {}
00261 };
00262
00263 template< typename T >
00264 class H_list_t_bss : public H_list_t<T>
00265 {
00266 public:
00267 H_list_t_bss() : H_list_t<T>(true) {}
00268 };
00269
00270
00271 }

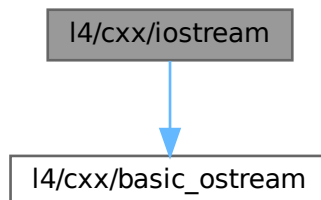
```

## 17.186 l4/cxx/iostream File Reference

IO Stream.

```
#include <l4/cxx/basic_ostream>
```

Include dependency graph for iostream:



### Namespaces

- namespace [L4](#)  
*L4 low-level kernel interface.*

### Variables

- BasicOStream **L4::cout**  
*Standard output stream.*
- BasicOStream **L4::cerr**  
*Standard error stream.*

## 17.186.1 Detailed Description

IO Stream.

Definition in file [iostream](#).

## 17.187 iostream

[Go to the documentation of this file.](#)

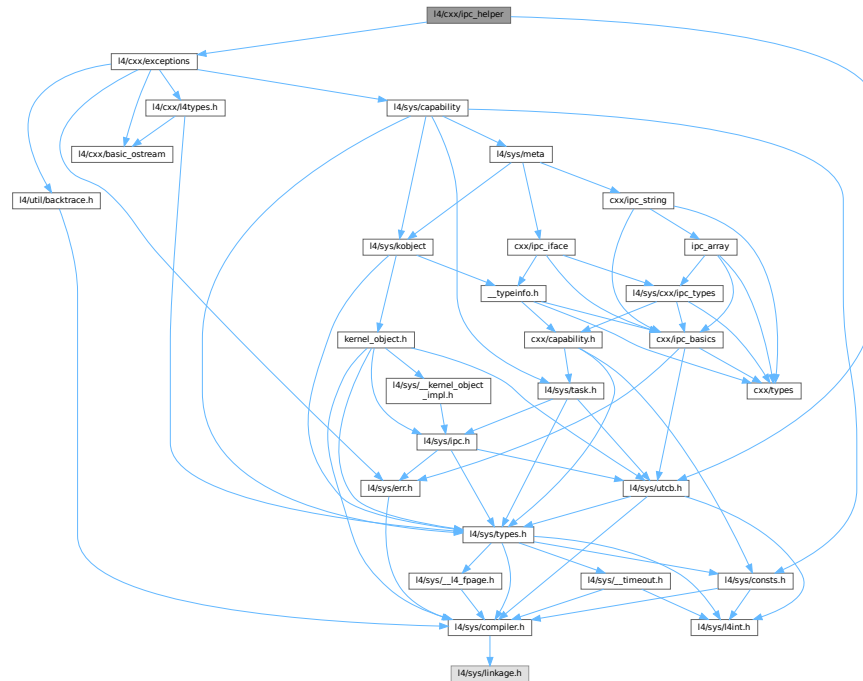
```
00001 // -*- Mode: C++ -*-
00002 // vim:ft=cpp
00007 /*
00008 * (c) 2004-2009 Alexander Warg <warg@os.inf.tu-dresden.de>,
00009 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #pragma once
00015
00016 #include <l4/cxx/basic_ostream>
00017
00018 namespace L4 {
00019
00023 extern BasicOStream cout;
00024
00028 extern BasicOStream cerr;
00029
00030 extern void iostream_init();
00031
00032 static void __attribute__((used, constructor)) __iostream_init()
00033 { iostream_init(); }
00034 };
```

## 17.188 l4/cxx/ipc\_helper File Reference

IPC helper.

```
#include <l4/cxx/exceptions>
#include <l4/sys/utcb.h>
```

Include dependency graph for ipc\_helper:



## Namespaces

- namespace [L4](#)  
*[L4](#) low-level kernel interface.*

## Functions

- void [L4::throw\\_ipc\\_exception](#) ([L4::Cap](#)< void > const &o, [l4\\_msgtag\\_t](#) const &err, [l4\\_utcb\\_t](#) \*utcb)  
*Throw an [L4](#) IPC error as exception.*
- void [L4::throw\\_ipc\\_exception](#) (void const \*o, [l4\\_msgtag\\_t](#) const &err, [l4\\_utcb\\_t](#) \*utcb)  
*Throw an [L4](#) IPC error as exception.*

### 17.188.1 Detailed Description

IPC helper.

Definition in file [ipc\\_helper](#).

## 17.189 ipc\_helper

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 #pragma once
00010
00011 #include <l4/cxx/exceptions>
00012 #include <l4/sys/utcb.h>
00013
00014 namespace L4
00015 {
00016 #ifdef __EXCEPTIONS
00017 inline void
00018 throw_ipc_exception([[maybe_unused]] L4::Cap<void> const &o,
00019 l4_msgtag_t const &err, l4_utcb_t *utcb)
00020 {
00021 if (err.has_error())
00022 throw (L4::Com_error(l4_error_u(err, utcb)));
00023 }
00024
00025 inline void
00026 throw_ipc_exception(void const *o, l4_msgtag_t const &err,
00027 l4_utcb_t *utcb)
00028 {
00029 throw_ipc_exception(L4::Cap<void>(o), err, utcb);
00030 }
00031 #endif
00032 }

```

## 17.190 l4/cxx/ipc\_stream File Reference

IPC stream.

```

#include <l4/sys/ipc.h>
#include <l4/sys/capability>
#include <l4/sys/cxx/ipc_types>
#include <l4/sys/cxx/ipc_varg>
#include <l4/cxx/type_traits>
#include <l4/cxx/minmax>

```



## Namespaces

- namespace [L4](#)  
*L4 low-level kernel interface.*
- namespace [L4::lpc](#)  
*IPC related functionality.*

## Functions

- `template<typename T>`  
`Internal::Buf_cp_out< T > L4::lpc::buf\_cp\_out (T const *v, unsigned long size)`  
*Insert an array into an [lpc::Ostream](#).*
- `template<typename T>`  
`Internal::Buf_cp_in< T > L4::lpc::buf\_cp\_in (T *v, unsigned long &size)`  
*Extract an array from an [lpc::Istream](#).*
- `template<typename T>`  
`Str_cp_in< T > L4::lpc::str\_cp\_in (T *v, unsigned long &size)`  
*Create a [Str\\_cp\\_in](#) for the given values.*
- `template<typename T>`  
`Msg_ptr< T > L4::lpc::msg\_ptr (T *&p)`  
*Create an [Msg\\_ptr](#) to adjust the given pointer.*
- `template<typename T>`  
`Internal::Buf_in< T > L4::lpc::buf\_in (T *&v, unsigned long &size)`  
*Return a pointer to stream array data.*
- `L4::lpc::Istream & operator>> (L4::lpc::Istream &s, bool &v)`  
*Extract one element of type *T* from the stream *s*.*
- `L4::lpc::Istream & operator>> (L4::lpc::Istream &s, l4\_msgtag\_t &v)`  
*Extract the *L4* message tag from the stream *s*.*
- `template<typename T>`  
`L4::lpc::Istream & operator>> (L4::lpc::Istream &s, L4::lpc::Internal::Buf\_in< T > const &v)`  
*Extract an array of *T* elements from the stream *s*.*
- `template<typename T>`  
`L4::lpc::Istream & operator>> (L4::lpc::Istream &s, L4::lpc::Msg\_ptr< T > const &v)`  
*Extract an element of type *T* from the stream *s*.*
- `template<typename T>`  
`L4::lpc::Istream & operator>> (L4::lpc::Istream &s, L4::lpc::Str\_cp\_in< T > const &v)`  
*Extract a zero-terminated string from the stream.*
- `L4::lpc::Ostream & operator<< (L4::lpc::Ostream &s, bool v)`  
*Insert an element to type *T* into the stream *s*.*
- `L4::lpc::Ostream & operator<< (L4::lpc::Ostream &s, l4\_msgtag\_t const &v)`  
*Insert the *L4* message tag into the stream *s*.*
- `template<typename T>`  
`L4::lpc::Ostream & operator<< (L4::lpc::Ostream &s, L4::lpc::Internal::Buf\_cp\_out< T > const &v)`  
*Insert an array with elements of type *T* into the stream *s*.*
- `L4::lpc::Ostream & operator<< (L4::lpc::Ostream &s, char const *v)`  
*Insert a zero terminated character string into the stream *s*.*
- `template<typename T>`  
`T L4::lpc::read (Istream &s)`  
*Read a value out of a stream.*



## 17.190.1 Detailed Description

IPC stream.

Definition in file [ipc\\_stream](#).

## 17.190.2 Function Documentation

### 17.190.2.1 `operator<<()` [1/4]

```
L4::Ipc::Ostream & operator<< (
 L4::Ipc::Ostream & s,
 bool v) [inline]
```

Insert an element to type `T` into the stream `s`.

#### Parameters

|                |                                                   |
|----------------|---------------------------------------------------|
| <code>s</code> | The stream to insert the element <code>v</code> . |
| <code>v</code> | The element to insert.                            |

#### Returns

The stream `s`.

Definition at line 1197 of file [ipc\\_stream](#).

References [L4::Ipc::Ostream::put\(\)](#).

Here is the call graph for this function:



### 17.190.2.2 `operator<<()` [2/4]

```
L4::Ipc::Ostream & operator<< (
 L4::Ipc::Ostream & s,
 char const * v) [inline]
```

Insert a zero terminated character string into the stream `s`.

#### Parameters

|          |                                            |
|----------|--------------------------------------------|
| <i>s</i> | The stream to insert the string <i>v</i> . |
| <i>v</i> | The string to insert.                      |

#### Returns

The stream *s*.

This operator produces basically the same content as the array insertion, however the length of the array is calculated using `strlen(v) + 1`. The string is copied into the message including the trailing zero.

Definition at line 1269 of file `ipc_stream`.

References `L4::Ipc::Ostream::put()`.

Here is the call graph for this function:



#### 17.190.2.3 operator<<() [3/4]

```

template<typename T>
L4::Ipc::Ostream & operator<< (
 L4::Ipc::Ostream & s,
 L4::Ipc::Internal::Buf_cp_out< T > const & v) [inline]

```

Insert an array with elements of type *T* into the stream *s*.

#### Parameters

|          |                                                            |
|----------|------------------------------------------------------------|
| <i>s</i> | The stream to insert the array <i>v</i> .                  |
| <i>v</i> | The array to insert (see <code>Ipc::Buf_cp_out()</code> ). |

**Returns**

The stream *s*.

Definition at line 1248 of file [ipc\\_stream](#).

References [L4::Ipc::Ostream::put\(\)](#).

Here is the call graph for this function:

**17.190.2.4 operator<<() [4/4]**

```
L4::Ipc::Ostream & operator<< (
 L4::Ipc::Ostream & s,
 l4_msgtag_t const & v) [inline]
```

Insert the [L4](#) message tag into the stream *s*.

**Parameters**

|          |                                               |
|----------|-----------------------------------------------|
| <i>s</i> | The stream to insert the tag <i>v</i> .       |
| <i>v</i> | The <a href="#">L4</a> message tag to insert. |

**Returns**

The stream *s*.

**Note**

Only one message tag can be inserted into a stream. Multiple insertions simply overwrite previous insertions.

Definition at line 1232 of file [ipc\\_stream](#).

References [L4::Ipc::Ostream::tag\(\)](#).

Here is the call graph for this function:



**17.190.2.5 operator>>() [1/6]**

```
L4::Ipc::Istream & operator>> (
 L4::Ipc::Istream & s,
 bool & v) [inline]
```

Extract one element of type T from the stream s.

**Parameters**

|     |   |                             |
|-----|---|-----------------------------|
|     | s | The stream to extract from. |
| out | v | Extracted value.            |

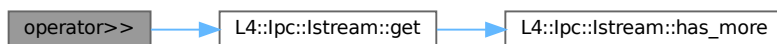
**Returns**

The stream s.

Definition at line 1044 of file [ipc\\_stream](#).

References [L4::Ipc::Istream::get\(\)](#).

Here is the call graph for this function:

**17.190.2.6 operator>>() [2/6]**

```
template<typename T>
L4::Ipc::Istream & operator>> (
 L4::Ipc::Istream & s,
 L4::Ipc::Internal::Buf_cp_in< T > const & v) [inline]
```

Extract an array of T elements from the stream s.

**Parameters**

|     |   |                                                                                                |
|-----|---|------------------------------------------------------------------------------------------------|
|     | s | The stream to extract from.                                                                    |
| out | v | <a href="#">Buffer</a> description to copy the array to ( <a href="#">Ipc::Buf_cp_out()</a> ). |

**Returns**

The stream `s`.

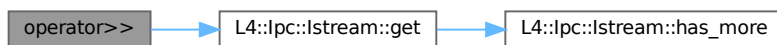
This operator does a copy out of the data into the given buffer.

See `lpc::Buf_in`, `lpc::Buf_cp_in`, and `lpc::Buf_cp_out`.

Definition at line 1151 of file `ipc_stream`.

References [L4::lpc::Istream::get\(\)](#).

Here is the call graph for this function:

**17.190.2.7 operator>>() [3/6]**

```

template<typename T>
L4::Ipc::Istream & operator>> (
 L4::Ipc::Istream & s,
 L4::Ipc::Internal::Buf_in< T > const & v) [inline]

```

Extract an array of `T` elements from the stream `s`.

**Parameters**

|                  |                |                                                               |
|------------------|----------------|---------------------------------------------------------------|
|                  | <code>s</code> | The stream to extract from.                                   |
| <code>out</code> | <code>v</code> | Pointer to the extracted array ( <code>ipc_buf_in()</code> ). |

**Returns**

The stream `s`.

This operator actually does not copy out the data in the array, but returns a pointer into the message buffer itself. This means that the data is only valid as long as there is no new data inserted into the stream.

**Note**

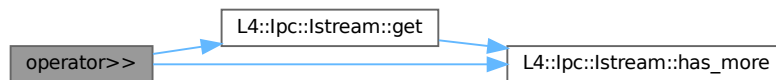
If array does not fit into transmitted words size will be set to zero. Client has to implement check against zero.

See `lpc::Buf_in`, `lpc::Buf_cp_in`, and `lpc::Buf_cp_out`.

Definition at line 1103 of file `ipc_stream`.

References `L4::lpc::Istream::get()`, and `L4::lpc::Istream::has_more()`.

Here is the call graph for this function:

**17.190.2.8 operator>>() [4/6]**

```

template<typename T>
L4::Ipc::Istream & operator>> (
 L4::Ipc::Istream & s,
 L4::Ipc::Msg_ptr< T > const & v) [inline]

```

Extract an element of type `T` from the stream `s`.

**Parameters**

|                  |                |                                   |
|------------------|----------------|-----------------------------------|
|                  | <code>s</code> | The stream to extract from.       |
| <code>out</code> | <code>v</code> | Pointer to the extracted element. |

**Returns**

The stream `s`.

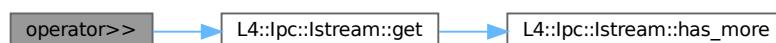
This operator actually does not copy out the data, but returns a pointer into the message buffer itself. This means that the data is only valid as long as there is no new data inserted into the stream.

See `Msg_ptr`.

Definition at line 1130 of file `ipc_stream`.

References `L4::lpc::Istream::get()`.

Here is the call graph for this function:



**17.190.2.9 operator>>() [5/6]**

```
template<typename T>
L4::Ipc::Istream & operator>> (
 L4::Ipc::Istream & s,
 L4::Ipc::Str_cp_in< T > const & v) [inline]
```

Extract a zero-terminated string from the stream.

**Parameters**

|     |          |                                                              |
|-----|----------|--------------------------------------------------------------|
|     | <i>s</i> | The stream to extract from.                                  |
| out | <i>v</i> | Buffer description to copy the array to (lpc::Str_cp_out()). |

**Returns**

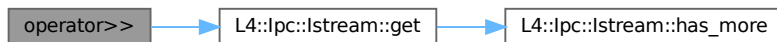
The stream *s*.

This operator does a copy out of the data into the given buffer.

Definition at line 1172 of file `lpc_stream`.

References `L4::lpc::Istream::get()`.

Here is the call graph for this function:

**17.190.2.10 operator>>() [6/6]**

```
L4::Ipc::Istream & operator>> (
 L4::Ipc::Istream & s,
 l4_msgtag_t & v) [inline]
```

Extract the `L4` message tag from the stream *s*.

**Parameters**

|     |          |                             |
|-----|----------|-----------------------------|
|     | <i>s</i> | The stream to extract from. |
| out | <i>v</i> | The extracted tag.          |

**Returns**

The stream `s`.

Definition at line 1078 of file `ipc_stream`.

References `L4::ipc::Istream::tag()`.

Here is the call graph for this function:



## 17.191 ipc\_stream

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00009 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #pragma once
00015
00016 #include <l4/sys/ipc.h>
00017 #include <l4/sys/capability>
00018 #include <l4/sys/cxx/ipc_types>
00019 #include <l4/sys/cxx/ipc_varg>
00020 #include <l4/cxx/type_traits>
00021 #include <l4/cxx/minmax>
00022
00023 namespace L4 {
00024 namespace Ipc {
00025
00026 class Ostream;
00027 class Istream;
00028
00029 namespace Internal {
00047 template< typename T >
00048 class Buf_cp_out
00049 {
00050 public:
00057 Buf_cp_out(T const *v, unsigned long size) : _v(v), _s(size) {}
00058
00066 unsigned long size() const { return _s; }
00067
00075 T const *buf() const { return _v; }
00076
00077 private:
00078 friend class Ostream;
00079 T const *_v;
00080 unsigned long _s;
00081 };
00082 }
00083
00099 template< typename T >
00100 Internal::Buf_cp_out<T> buf_cp_out(T const *v, unsigned long size)
00101 { return Internal::Buf_cp_out<T>(v, size); }
00102
00103
00104 namespace Internal {

```



```

00117 template< typename T >
00118 class Buf_cp_in
00119 {
00120 public:
00129 Buf_cp_in(T *v, unsigned long &size) : _v(v), _s(&size) {}
00130
00131 unsigned long &size() const { return *_s; }
00132 T *buf() const { return _v; }
00133
00134 private:
00135 friend class Istream;
00136 T *_v;
00137 unsigned long *_s;
00138 };
00139 }
00140
00158 template< typename T >
00159 Internal::Buf_cp_in<T> buf_cp_in(T *v, unsigned long &size)
00160 { return Internal::Buf_cp_in<T>(v, size); }
00161
00177 template< typename T >
00178 class Str_cp_in
00179 {
00180 public:
00189 Str_cp_in(T *v, unsigned long &size) : _v(v), _s(&size) {}
00190
00191 unsigned long &size() const { return *_s; }
00192 T *buf() const { return _v; }
00193
00194 private:
00195 friend class Istream;
00196 T *_v;
00197 unsigned long *_s;
00198 };
00199
00212 template< typename T >
00213 Str_cp_in<T> str_cp_in(T *v, unsigned long &size)
00214 { return Str_cp_in<T>(v, size); }
00215
00228 template< typename T >
00229 class Msg_ptr
00230 {
00231 private:
00232 T **_p;
00233 public:
00240 explicit Msg_ptr(T *&p) : _p(&p) {}
00241 void set(T *p) const { *_p = p; }
00242 };
00243
00251 template< typename T >
00252 Msg_ptr<T> msg_ptr(T *&p)
00253 { return Msg_ptr<T>(p); }
00254
00255 namespace Internal {
00270 template< typename T >
00271 class Buf_in
00272 {
00273 public:
00280 Buf_in(T *&v, unsigned long &size) : _v(&v), _s(&size) {}
00281
00282 void set_size(unsigned long s) const { *_s = s; }
00283 T *&buf() const { return *_v; }
00284
00285 private:
00286 friend class Istream;
00287 T **_v;
00288 unsigned long *_s;
00289 };
00290 }
00291
00309 template< typename T >
00310 Internal::Buf_in<T> buf_in(T *&v, unsigned long &size)
00311 { return Internal::Buf_in<T>(v, size); }
00312
00313 namespace Utc_b_stream_check
00314 {
00315 static bool check_utcb_data_offset(unsigned sz)
00316 { return sz > sizeof(l4_umword_t) * L4_UTCB_GENERIC_DATA_SIZE; }
00317 }
00318
00319
00334 class Istream
00335 {
00336 public:
00348 Istream(l4_utcb_t *utcb)
00349 : _tag(), _utcb(utcb),

```

```

00350 _current_msg(reinterpret_cast<char*>(l4_utcb_mr_u(utcb)->mr)),
00351 _pos(0), _current_buf(0)
00352 {}
00353
00358 void reset()
00359 {
00360 _pos = 0;
00361 _current_buf = 0;
00362 _current_msg = reinterpret_cast<char*>(l4_utcb_mr_u(utcb)->mr);
00363 }
00364
00368 template< typename T >
00369 bool has_more(unsigned long count = 1)
00370 {
00371 auto const max_bytes = L4_UTCB_GENERIC_DATA_SIZE * sizeof(l4_umword_t);
00372 unsigned apos = cxx::Type_traits<T>::align(_pos);
00373 return (count <= max_bytes / sizeof(T))
00374 && (apos + (sizeof(T) * count)
00375 <= _tag.words() * sizeof(l4_umword_t));
00376 }
00377
00382
00393 template< typename T >
00394 unsigned long get(T *buf, unsigned long elems)
00395 {
00396 if (L4_UNLIKELY(!has_more<T>(elems)))
00397 return 0;
00398
00399 unsigned long size = elems * sizeof(T);
00400 _pos = cxx::Type_traits<T>::align(_pos);
00401
00402 __builtin_memcpy(buf, _current_msg + _pos, size);
00403 _pos += size;
00404 return elems;
00405 }
00406
00407
00413 template< typename T >
00414 void skip(unsigned long elems)
00415 {
00416 if (L4_UNLIKELY(!has_more<T>(elems)))
00417 return;
00418
00419 unsigned long size = elems * sizeof(T);
00420 _pos = cxx::Type_traits<T>::align(_pos);
00421 _pos += size;
00422 }
00423
00438 template< typename T >
00439 unsigned long get(Msg_ptr<T> const &buf, unsigned long elems = 1)
00440 {
00441 if (L4_UNLIKELY(!has_more<T>(elems)))
00442 return 0;
00443
00444 unsigned long size = elems * sizeof(T);
00445 _pos = cxx::Type_traits<T>::align(_pos);
00446
00447 buf.set(reinterpret_cast<T*>(_current_msg + _pos));
00448 _pos += size;
00449 return elems;
00450 }
00451
00452
00463 template< typename T >
00464 bool get(T &v)
00465 {
00466 if (L4_UNLIKELY(!has_more<T>()))
00467 {
00468 v = T();
00469 return false;
00470 }
00471
00472 _pos = cxx::Type_traits<T>::align(_pos);
00473 v = *(reinterpret_cast<T*>(_current_msg + _pos));
00474 _pos += sizeof(T);
00475 return true;
00476 }
00477
00478
00479 bool get(Ipc::Varg *va)
00480 {
00481 Ipc::Varg::Tag t;
00482 if (!has_more<Ipc::Varg::Tag>())
00483 {
00484 va->tag(0);
00485 return 0;
00486 }

```

```

00487 get(t);
00488 va->tag(t);
00489 char const *d;
00490 get(msg_ptr(d), va->length());
00491 va->data(d);
00492
00493 return 1;
00494 }
00495
00505 l4_msgtag_t tag() const { return _tag; }
00506
00507
00517 l4_msgtag_t &tag() { return _tag; }
00518
00520
00525 inline bool put(Rcv_fpage const &);
00526
00531 inline bool put(Small_buf const &);
00532
00533
00538
00548 inline l4_msgtag_t wait(l4_umword_t *src)
00549 { return wait(src, L4_IPC_NEVER); }
00550
00561 inline l4_msgtag_t wait(l4_umword_t *src, l4_timeout_t timeout);
00562
00572 inline l4_msgtag_t receive(l4_cap_idx_t src)
00573 { return receive(src, L4_IPC_NEVER); }
00574
00575 inline l4_msgtag_t receive(l4_cap_idx_t src, l4_timeout_t timeout);
00576
00577
00581 inline l4_utcb_t *utcb() const { return _utcb; }
00582
00583 protected:
00584 l4_msgtag_t _tag;
00585 l4_utcb_t *_utcb;
00586 char *_current_msg;
00587 unsigned _pos;
00588 unsigned char _current_buf;
00589 };
00590
00591 class Istream_copy : public Istream
00592 {
00593 private:
00594 l4_msg_regs_t _mrs;
00595
00596 public:
00597 Istream_copy(Istream const &o) : Istream(o), _mrs(*l4_utcb_mr_u(o.utcb()))
00598 {
00599 // do some reverse mr to utcb trickery
00600 _utcb = reinterpret_cast<l4_utcb_t *>
00601 (reinterpret_cast<l4_addr_t>(&_mrs)
00602 - reinterpret_cast<l4_addr_t>(l4_utcb_mr_u(nullptr)));
00603 _current_msg = reinterpret_cast<char*>(l4_utcb_mr_u(_utcb)->mr);
00604 }
00605
00606 };
00607
00623 class Ostream
00624 {
00625 public:
00629 Ostream(l4_utcb_t *utcb)
00630 : _tag(), _utcb(utcb),
00631 _current_msg(reinterpret_cast<char *>(l4_utcb_mr_u(_utcb)->mr)),
00632 _pos(0), _current_item(0)
00633 {}
00634
00638 void reset()
00639 {
00640 _pos = 0;
00641 _current_item = 0;
00642 _current_msg = reinterpret_cast<char*>(l4_utcb_mr_u(_utcb)->mr);
00643 }
00644
00652
00659 template< typename T >
00660 bool put(T *buf, unsigned long size)
00661 {
00662 size *= sizeof(T);
00663 _pos = cxx::Type_traits<T>::align(_pos);
00664 if (Utcb_stream_check::check_utcb_data_offset(_pos + size))
00665 return false;
00666
00667 __builtin_memcpy(_current_msg + _pos, buf, size);
00668 _pos += size;
00669 return true;
00670 }

```

```

00671
00672 template< typename T >
00673 bool put(T const &v)
00674 {
00675 _pos = cxx::Type_traits<T>::align(_pos);
00676 if (Utcstream_check::check_utcb_data_offset(_pos + sizeof(T)))
00677 return false;
00678
00679 *(reinterpret_cast<T*>(_current_msg + _pos)) = v;
00680 _pos += sizeof(T);
00681 return true;
00682 }
00683
00684 int put(Varg const &va)
00685 {
00686 put(va.tag());
00687 put(va.data(), va.length());
00688
00689 return 0;
00690 }
00691
00692 template< typename T >
00693 int put(Varg_t<T> const &va)
00694 { return put(static_cast<Varg const &>(va)); }
00695
00696 l4_msgtag_t tag() const { return _tag; }
00697
00698 l4_msgtag_t &tag() { return _tag; }
00699
00700 inline bool put_snd_item(Snd_fpage const &);
00701
00702 inline l4_msgtag_t send(l4_cap_idx_t dst, long proto = 0, unsigned flags = 0);
00703
00704 inline l4_utcb_t *utcb() const { return _utcb; }
00705 #if 0
00706 unsigned long tell() const
00707 {
00708 unsigned w = l4_bytes_to_mwords(_pos) - _current_item * 2;
00709 _tag = l4_msgtag(0, w, _current_item, 0);
00710 }
00711 #endif
00712 public:
00713 l4_msgtag_t prepare_ipc(long proto = 0, unsigned flags = 0)
00714 {
00715 unsigned w = l4_bytes_to_mwords(_pos) - _current_item * 2;
00716 return l4_msgtag(proto, w, _current_item, flags);
00717 }
00718
00719 // XXX: this is a hack for <l4/sys/cxx/ipc_server> adaption
00720 void set_ipc_params(l4_msgtag_t tag)
00721 {
00722 _pos = (tag.words() + tag.items() * 2) * sizeof(l4_umword_t);
00723 _current_item = tag.items();
00724 }
00725 protected:
00726 l4_msgtag_t _tag;
00727 l4_utcb_t *_utcb;
00728 char *_current_msg;
00729 unsigned _pos;
00730 unsigned char _current_item;
00731 };
00732
00733 class Iostream : public Istream, public Ostream
00734 {
00735 public:
00736 explicit Iostream(l4_utcb_t *utcb)
00737 : Istream(utcb), Ostream(utcb)
00738 {}
00739
00740 // disambiguate those functions
00741 l4_msgtag_t tag() const { return Istream::tag(); }
00742 l4_msgtag_t &tag() { return Istream::tag(); }
00743 l4_utcb_t *utcb() const { return Istream::utcb(); }
00744
00745 void reset()
00746 {
00747 Istream::reset();
00748 Ostream::reset();
00749 }
00750
00751
00752
00753

```

```

00829
00830 using Istream::get;
00831 using Istream::put;
00832 using Ostream::put;
00833
00835
00840
00856 inline l4_msgtag_t call(l4_cap_idx_t dst, l4_timeout_t timeout, long proto = 0);
00857 inline l4_msgtag_t call(l4_cap_idx_t dst, long proto = 0);
00858
00874 inline l4_msgtag_t reply_and_wait(l4_umword_t *src_dst, long proto = 0)
00875 { return reply_and_wait(src_dst, L4_IPC_SEND_TIMEOUT_0, proto); }
00876
00877 inline l4_msgtag_t send_and_wait(l4_cap_idx_t dest, l4_umword_t *src,
00878 long proto = 0)
00879 { return send_and_wait(dest, src, L4_IPC_SEND_TIMEOUT_0, proto); }
00880
00897 inline l4_msgtag_t reply_and_wait(l4_umword_t *src_dst,
00898 l4_timeout_t timeout, long proto = 0);
00899 inline l4_msgtag_t send_and_wait(l4_cap_idx_t dest, l4_umword_t *src,
00900 l4_timeout_t timeout, long proto = 0);
00901 inline l4_msgtag_t reply(l4_timeout_t timeout, long proto = 0);
00902 inline l4_msgtag_t reply(long proto = 0)
00903 { return reply(L4_IPC_SEND_TIMEOUT_0, proto); }
00904
00906 };
00907
00908
00909 inline bool
00910 Ostream::put_snd_item(Snd_fpage const &v)
00911 {
00912 typedef Snd_fpage T;
00913 _pos = cxx::Type_traits<Snd_fpage>::align(_pos);
00914 if (Utcb_stream_check::check_utcb_data_offset(_pos + sizeof(T))
00915 return false;
00916
00917 *(reinterpret_cast<T*>(_current_msg + _pos)) = v;
00918 _pos += sizeof(T);
00919 ++_current_item;
00920 return true;
00921 }
00922
00923
00924 inline bool
00925 Istream::put(Rcv_fpage const &item)
00926 {
00927 unsigned words = item.forward_mappings() ? 3 : 2;
00928 if (_current_buf >= L4_UTCB_GENERIC_BUFFERS_SIZE - words - 1)
00929 return false;
00930
00931 l4_utcb_br_u(_utcb)->bdr &= ~L4_BDR_OFFSET_MASK;
00932
00933 l4_umword_t *buf
00934 = reinterpret_cast<l4_umword_t *>(&l4_utcb_br_u(_utcb)->br[_current_buf]);
00935 *buf++ = item.base_x();
00936 *buf++ = item.data();
00937 if (item.forward_mappings())
00938 *buf++ = item.rcv_task();
00939 _current_buf += words;
00940 return true;
00941 }
00942
00943
00944 inline bool
00945 Istream::put(Small_buf const &item)
00946 {
00947 if (_current_buf >= L4_UTCB_GENERIC_BUFFERS_SIZE - 2)
00948 return false;
00949
00950 l4_utcb_br_u(_utcb)->bdr &= ~L4_BDR_OFFSET_MASK;
00951
00952 reinterpret_cast<Small_buf&>(l4_utcb_br_u(_utcb)->br[_current_buf]) = item;
00953 _current_buf += 1;
00954 return true;
00955 }
00956
00957
00958 inline l4_msgtag_t
00959 Ostream::send(l4_cap_idx_t dst, long proto, unsigned flags)
00960 {
00961 l4_msgtag_t tag = prepare_ipc(proto, L4_MSGTAG_FLAGS & flags);
00962 return l4_ipc_send(dst, _utcb, tag, L4_IPC_NEVER);
00963 }
00964
00965 inline l4_msgtag_t
00966 Iostream::call(l4_cap_idx_t dst, l4_timeout_t timeout, long label)
00967 {

```

```

00968 l4_msgtag_t tag = prepare_ipc(label);
00969 tag = l4_ipc_call(dst, Ostream::_utcb, tag, timeout);
00970 Istream::tag() = tag;
00971 Istream::_pos = 0;
00972 return tag;
00973 }
00974
00975 inline l4_msgtag_t
00976 Iostream::call(l4_cap_idx_t dst, long label)
00977 { return call(dst, L4_IPC_NEVER, label); }
00978
00979
00980 inline l4_msgtag_t
00981 Iostream::reply_and_wait(l4_umword_t *src_dst, l4_timeout_t timeout, long proto)
00982 {
00983 l4_msgtag_t tag = prepare_ipc(proto);
00984 tag = l4_ipc_reply_and_wait(Ostream::_utcb, tag, src_dst, timeout);
00985 Istream::tag() = tag;
00986 Istream::_pos = 0;
00987 return tag;
00988 }
00989
00990
00991 inline l4_msgtag_t
00992 Iostream::send_and_wait(l4_cap_idx_t dest, l4_umword_t *src,
00993 l4_timeout_t timeout, long proto)
00994 {
00995 l4_msgtag_t tag = prepare_ipc(proto);
00996 tag = l4_ipc_send_and_wait(dest, Ostream::_utcb, tag, src, timeout);
00997 Istream::tag() = tag;
00998 Istream::_pos = 0;
00999 return tag;
01000 }
01001
01002 inline l4_msgtag_t
01003 Iostream::reply(l4_timeout_t timeout, long proto)
01004 {
01005 l4_msgtag_t tag = prepare_ipc(proto);
01006 tag = l4_ipc_send(L4_INVALID_CAP | L4_SYSF_REPLY, Ostream::_utcb, tag, timeout);
01007 Istream::tag() = tag;
01008 Istream::_pos = 0;
01009 return tag;
01010 }
01011
01012 inline l4_msgtag_t
01013 Istream::wait(l4_umword_t *src, l4_timeout_t timeout)
01014 {
01015 l4_msgtag_t res;
01016 res = l4_ipc_wait(_utcb, src, timeout);
01017 tag() = res;
01018 _pos = 0;
01019 return res;
01020 }
01021
01022
01023 inline l4_msgtag_t
01024 Istream::receive(l4_cap_idx_t src, l4_timeout_t timeout)
01025 {
01026 l4_msgtag_t res;
01027 res = l4_ipc_receive(src, _utcb, timeout);
01028 tag() = res;
01029 _pos = 0;
01030 return res;
01031 }
01032
01033 } // namespace Ipc
01034 } // namespace L4
01035
01044 inline L4::Ipc::Istream &operator » (L4::Ipc::Istream &s, bool &v) { s.get(v); return s; }
01045 inline L4::Ipc::Istream &operator » (L4::Ipc::Istream &s, int &v) { s.get(v); return s; }
01046 inline L4::Ipc::Istream &operator » (L4::Ipc::Istream &s, long int &v) { s.get(v); return s; }
01047 inline L4::Ipc::Istream &operator » (L4::Ipc::Istream &s, long long int &v) { s.get(v); return s; }
01048 inline L4::Ipc::Istream &operator » (L4::Ipc::Istream &s, unsigned int &v) { s.get(v); return s; }
01049 inline L4::Ipc::Istream &operator » (L4::Ipc::Istream &s, unsigned long int &v) { s.get(v); return s; }
01050 inline L4::Ipc::Istream &operator » (L4::Ipc::Istream &s, unsigned long long int &v) { s.get(v);
 return s; }
01051 inline L4::Ipc::Istream &operator » (L4::Ipc::Istream &s, short int &v) { s.get(v); return s; }
01052 inline L4::Ipc::Istream &operator » (L4::Ipc::Istream &s, unsigned short int &v) { s.get(v); return s; }
01053 inline L4::Ipc::Istream &operator » (L4::Ipc::Istream &s, char &v) { s.get(v); return s; }
01054 inline L4::Ipc::Istream &operator » (L4::Ipc::Istream &s, unsigned char &v) { s.get(v); return s; }
01055 inline L4::Ipc::Istream &operator » (L4::Ipc::Istream &s, signed char &v) { s.get(v); return s; }
01056 inline L4::Ipc::Istream &operator « (L4::Ipc::Istream &s, L4::Ipc::Rcv_fpage const &v) { s.put(v);
 return s; }
01057 inline L4::Ipc::Istream &operator « (L4::Ipc::Istream &s, L4::Ipc::Small_buf const &v) { s.put(v);
 return s; }

```

```

01058 inline L4::Ipc::Istream &operator » (L4::Ipc::Istream &s, L4::Ipc::Snd_fpage &v)
01059 {
01060 l4_umword_t b, d;
01061 s » b » d;
01062 v = L4::Ipc::Snd_fpage(b, d);
01063 return s;
01064 }
01065 inline L4::Ipc::Istream &operator » (L4::Ipc::Istream &s, L4::Ipc::Varg &v)
01066 { s.get(&v); return s; }
01067
01068
01077 inline
01078 L4::Ipc::Istream &operator » (L4::Ipc::Istream &s, l4_msgtag_t &v)
01079 {
01080 v = s.tag();
01081 return s;
01082 }
01083
01101 template< typename T >
01102 inline
01103 L4::Ipc::Istream &operator » (L4::Ipc::Istream &s,
01104 L4::Ipc::Internal::Buf_in<T> const &v)
01105 {
01106 unsigned long si;
01107 if (s.get(si) && s.has_more<T>(si))
01108 v.set_size(s.get(L4::Ipc::Msg_ptr<T>(v.buf()), si));
01109 else
01110 v.set_size(0);
01111 return s;
01112 }
01113
01128 template< typename T >
01129 inline
01130 L4::Ipc::Istream &operator » (L4::Ipc::Istream &s,
01131 L4::Ipc::Msg_ptr<T> const &v)
01132 {
01133 s.get(v);
01134 return s;
01135 }
01136
01149 template< typename T >
01150 inline
01151 L4::Ipc::Istream &operator » (L4::Ipc::Istream &s,
01152 L4::Ipc::Internal::Buf_cp_in<T> const &v)
01153 {
01154 unsigned long sz;
01155 s.get(sz);
01156 v.size() = s.get(v.buf(), cxx::min(v.size(), sz));
01157 return s;
01158 }
01159
01170 template< typename T >
01171 inline
01172 L4::Ipc::Istream &operator » (L4::Ipc::Istream &s,
01173 L4::Ipc::Str_cp_in<T> const &v)
01174 {
01175 unsigned long sz;
01176 s.get(sz);
01177 unsigned long rsz = s.get(v.buf(), cxx::min(v.size(), sz));
01178 if (rsz < v.size() && v.buf()[rsz - 1])
01179 ++rsz; // add the zero termination behind the received data
01180
01181 if (rsz != 0)
01182 v.buf()[rsz - 1] = 0;
01183
01184 v.size() = rsz;
01185 return s;
01186 }
01187
01188
01197 inline L4::Ipc::Ostream &operator « (L4::Ipc::Ostream &s, bool v) { s.put(v); return s; }
01198 inline L4::Ipc::Ostream &operator « (L4::Ipc::Ostream &s, int v) { s.put(v); return s; }
01199 inline L4::Ipc::Ostream &operator « (L4::Ipc::Ostream &s, long int v) { s.put(v); return s; }
01200 inline L4::Ipc::Ostream &operator « (L4::Ipc::Ostream &s, long long int v) { s.put(v); return s; }
01201 inline L4::Ipc::Ostream &operator « (L4::Ipc::Ostream &s, unsigned int v) { s.put(v); return s; }
01202 inline L4::Ipc::Ostream &operator « (L4::Ipc::Ostream &s, unsigned long int v) { s.put(v); return s; }
01203 inline L4::Ipc::Ostream &operator « (L4::Ipc::Ostream &s, unsigned long long int v) { s.put(v); return
s; }
01204 inline L4::Ipc::Ostream &operator « (L4::Ipc::Ostream &s, short int v) { s.put(v); return s; }
01205 inline L4::Ipc::Ostream &operator « (L4::Ipc::Ostream &s, unsigned short int v) { s.put(v); return s;
}
01206 inline L4::Ipc::Ostream &operator « (L4::Ipc::Ostream &s, char v) { s.put(v); return s; }
01207 inline L4::Ipc::Ostream &operator « (L4::Ipc::Ostream &s, unsigned char v) { s.put(v); return s; }
01208 inline L4::Ipc::Ostream &operator « (L4::Ipc::Ostream &s, signed char v) { s.put(v); return s; }
01209 inline L4::Ipc::Ostream &operator « (L4::Ipc::Ostream &s, L4::Ipc::Snd_fpage const &v) {
s.put_snd_item(v); return s; }
01210 template< typename T >

```

```

01211 inline L4::Ipc::Ostream &operator << (L4::Ipc::Ostream &s, L4::Cap<T> const &v)
01212 { s << L4::Ipc::Snd_fpage(v.fpage()); return s; }
01213
01214 inline L4::Ipc::Ostream &operator << (L4::Ipc::Ostream &s, L4::Ipc::Varg const &v)
01215 { s.put(v); return s; }
01216 template< typename T >
01217 inline L4::Ipc::Ostream &operator << (L4::Ipc::Ostream &s, L4::Ipc::Varg_t<T> const &v)
01218 { s.put(v); return s; }
01219
01231 inline
01232 L4::Ipc::Ostream &operator << (L4::Ipc::Ostream &s, l4_msgtag_t const &v)
01233 {
01234 s.tag() = v;
01235 return s;
01236 }
01237
01246 template< typename T >
01247 inline
01248 L4::Ipc::Ostream &operator << (L4::Ipc::Ostream &s,
01249 L4::Ipc::Internal::Buf_cp_out<T> const &v)
01250 {
01251 s.put(v.size());
01252 s.put(v.buf(), v.size());
01253 return s;
01254 }
01255
01268 inline
01269 L4::Ipc::Ostream &operator << (L4::Ipc::Ostream &s, char const *v)
01270 {
01271 unsigned long l = __builtin_strlen(v) + 1;
01272 s.put(l);
01273 s.put(v, l);
01274 return s;
01275 }
01276
01277 namespace L4 { namespace Ipc {
01287 template< typename T >
01288 inline
01289 T read(Istream &s) { T t; s >> t; return t; }
01290
01291 } // namespace Ipc
01292 } // namespace L4

```

## 17.192 ipc\_timeout\_queue

```

00001 // vim:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2014 Steffen Liebergeld <steffen.liebergeld@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/cxx/hlist>
00010 #include <l4/sys/cxx/ipc_server_loop>
00011
00012 namespace L4 { namespace Ipc_svr {
00013
00018 class Timeout : public cxx::H_list_item
00019 {
00020 friend class Timeout_queue;
00021 public:
00023 Timeout() : _timeout(0) {}
00024
00026 virtual ~Timeout() = 0;
00027
00034 virtual void expired() = 0;
00035
00042 l4_kernel_clock_t timeout() const
00043 { return _timeout; }
00044
00045 private:
00046 l4_kernel_clock_t _timeout;
00047 };
00048
00049 inline Timeout::~~Timeout() {}
00050
00055 class Timeout_queue
00056 {
00057 public:
00059 typedef L4::Ipc_svr::Timeout Timeout;
00060
00065 l4_kernel_clock_t next_timeout() const

```



```

00066 {
00067 if (auto e = _timeouts.front())
00068 return e->timeout();
00069 return 0;
00070 }
00071
00072
00081 bool timeout_expired(l4_kernel_clock_t now) const
00082 {
00083 l4_kernel_clock_t next = next_timeout();
00084 return (next != 0) && (next <= now);
00085 }
00086
00091 void handle_expired_timeouts(l4_kernel_clock_t now)
00092 {
00093 while (!_timeouts.empty())
00094 {
00095 Queue::Iterator top = _timeouts.begin();
00096 if ((*top)->_timeout > now)
00097 return;
00098
00099 Timeout *t = *top;
00100 top = _timeouts.erase(top);
00101 t->expired();
00102 }
00103 }
00104
00111 void add(Timeout *timeout, l4_kernel_clock_t time)
00112 {
00113 timeout->_timeout = time;
00114 Queue::Iterator i = _timeouts.begin();
00115 while (i != _timeouts.end() && (*i)->timeout() < time)
00116 ++i;
00117 _timeouts.insert_before(timeout, i);
00118 }
00119
00126 void remove(Timeout *timeout)
00127 {
00128 _timeouts.remove(timeout);
00129 }
00130
00131 private:
00132 typedef cxx::H_list<Timeout> Queue;
00133 Queue _timeouts;
00134 };
00135
00149 template< typename HOOKS, typename BR_MAN = Br_manager_no_buffers >
00150 class Timeout_queue_hooks : public BR_MAN
00151 {
00152 l4_kernel_clock_t _now()
00153 { return static_cast<HOOKS*>(this)->now(); }
00154
00155 unsigned _timeout_br()
00156 { return this->first_free_br(); }
00157
00158 public:
00160 l4_timeout_t timeout()
00161 {
00162 l4_kernel_clock_t t = queue.next_timeout();
00163 if (t)
00164 return l4_timeout(L4_IPC_TIMEOUT_0, l4_timeout_abs(t, _timeout_br()));
00165 return L4_IPC_SEND_TIMEOUT_0;
00166 }
00167
00169 void setup_wait(l4_utcb_t *utcb, L4::Ipc_svr::Reply_mode mode)
00170 {
00171 // we must handle the timer only when called after a possible reply
00172 // otherwise we probably destroy the reply message.
00173 if (mode == L4::Ipc_svr::Reply_separate)
00174 {
00175 l4_kernel_clock_t now = _now();
00176 if (queue.timeout_expired(now))
00177 queue.handle_expired_timeouts(now);
00178 }
00179 BR_MAN::setup_wait(utcb, mode);
00180 }
00181 }
00182
00184 L4::Ipc_svr::Reply_mode before_reply(l4_msgtag_t, l4_utcb_t *)
00185 {
00186 // split up reply and wait when a timeout has expired
00187 if (queue.timeout_expired(_now()))
00188 return L4::Ipc_svr::Reply_separate;
00189 return L4::Ipc_svr::Reply_compound;
00190 }
00191

```

```

00202 int add_timeout(Timeout *timeout, l4_kernel_clock_t time) override
00203 {
00204 queue.add(timeout, time);
00205 return 0;
00206 }
00207
00215 int remove_timeout(Timeout *timeout) override
00216 {
00217 queue.remove(timeout);
00218 return 0;
00219 }
00220
00221 Timeout_queue queue;
00222 };
00223
00224 }}

```

## 17.193 l4/cxx/l4iostream File Reference

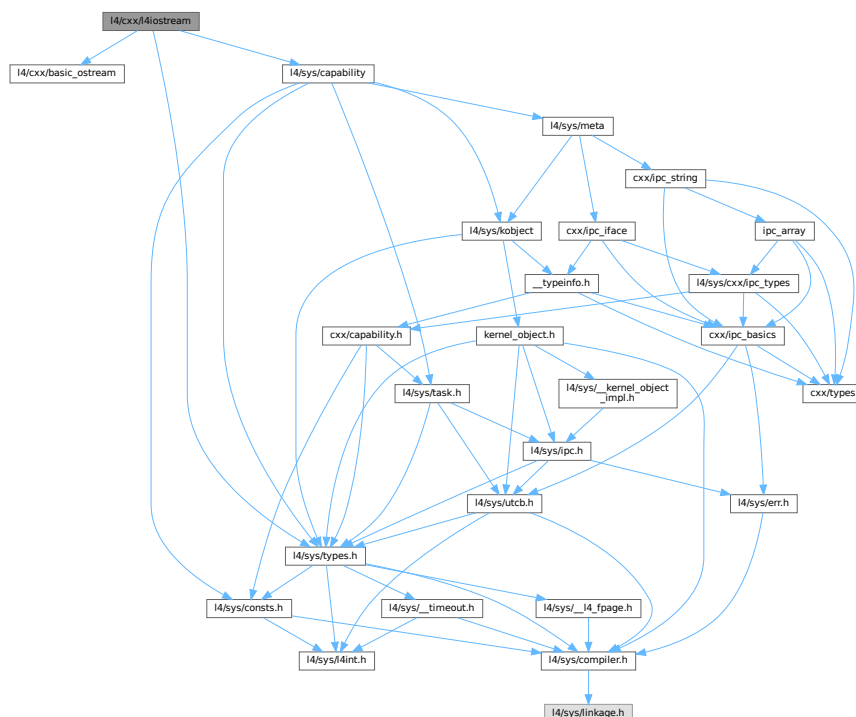
[L4](#) IO stream.

```

#include <l4/cxx/basic_ostream>
#include <l4/sys/types.h>
#include <l4/sys/capability>

```

Include dependency graph for l4iostream:



### 17.193.1 Detailed Description

[L4](#) IO stream.

Definition in file [l4iostream](#).

## 17.194 l4iostream

[Go to the documentation of this file.](#)

```

00001 // -*- Mode: C++ -*-
00002 // vim:ft=cpp
00003 /*
00004 * (c) 2004-2009 Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 #pragma once
00010
00011 #include <l4/cxx/basic_ostream>
00012 #include <l4/sys/types.h>
00013 #include <l4/sys/capability>
00014
00015 inline
00016 L4::BasicOStream &operator << (L4::BasicOStream &o, l4_msgtag_t const &tag)
00017 {
00018 L4::IOBackend::Mode m = o.be_mode();
00019 o << "[l=" << L4::dec << tag.label() << "; w=" << tag.words() << "; i="
00020 << tag.items() << "];";
00021 o.be_mode(m);
00022 return o;
00023 }
00024
00025 template<typename T>
00026 inline
00027 L4::BasicOStream &operator << (L4::BasicOStream &o, L4::Cap<T> const &cap)
00028 {
00029 o << "[C:" << L4::n_hex(cap.cap()) << "];";
00030 return o;
00031 }
00032
00033
00034
00035

```

## 17.195 l4/cxx/l4types.h File Reference

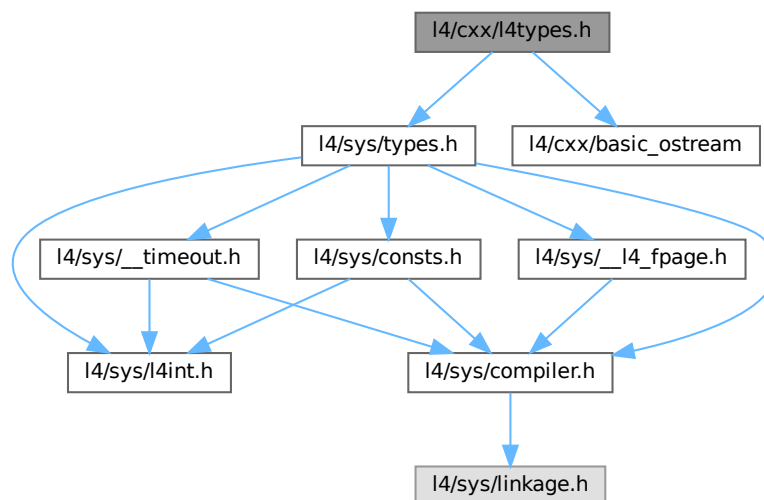
L4 Types.

```

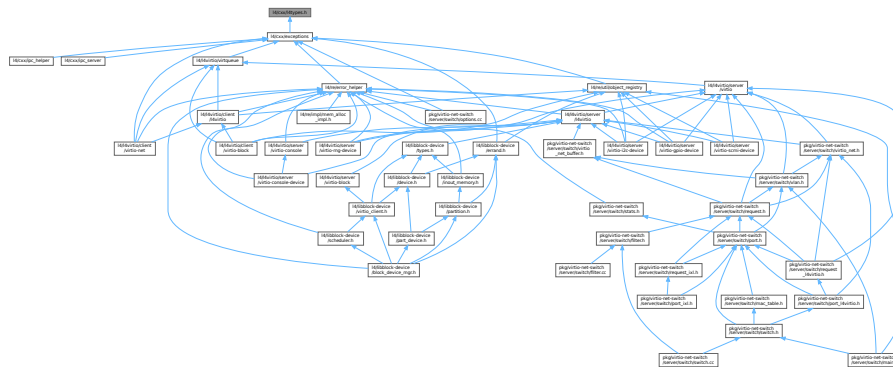
#include <l4/sys/types.h>
#include <l4/cxx/basic_ostream>

```

Include dependency graph for l4types.h:



This graph shows which files directly or indirectly include this file:



## 17.195.1 Detailed Description

[L4](#) Types.

Definition in file [l4types.h](#).

## 17.196 l4types.h

[Go to the documentation of this file.](#)

```
00001
00005 /*
00006 * (c) 2004-2009 Alexander Warg <warg@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #pragma once
00012
00013 #include <l4/sys/types.h>
00014 #include <l4/cxx/basic_ostream>
```

## 17.197 list

```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00009 #pragma once
00010
00011 #include <l4/cxx/type_traits>
00012 #include <l4/cxx/std_alloc>
00013 #include <l4/cxx/std_ops>
00014
00015 namespace cxx {
00016 /*
00017 * Classes: List_item, List<D, Alloc>
00018 */
00019
00020 class List_item
00021 {
00022 public:
00023 class Iter
00024 {
```

```

00036 public:
00037 Iter(List_item *c, List_item *f) noexcept : _c(c), _f(f) {}
00038 Iter(List_item *f = 0) noexcept : _c(f), _f(f) {}
00039
00040 List_item *operator * () const noexcept { return _c; }
00041 List_item *operator -> () const noexcept { return _c; }
00042 Iter &operator ++ () noexcept
00043 {
00044 if (!_f)
00045 _c = 0;
00046 else
00047 _c = _c->get_next_item();
00048
00049 if (_c == _f)
00050 _c = 0;
00051
00052 return *this;
00053 }
00054
00055 Iter operator ++ (int) noexcept
00056 { Iter o = *this; operator ++ (); return o; }
00057
00058 Iter &operator -- () noexcept
00059 {
00060 if (!_f)
00061 _c = 0;
00062 else
00063 _c = _c->get_prev_item();
00064
00065 if (_c == _f)
00066 _c = 0;
00067
00068 return *this;
00069 }
00070
00071 Iter operator -- (int) noexcept
00072 { Iter o = *this; operator -- (); return o; }
00073
00074 List_item *remove_me() noexcept
00075 {
00076 if (!_c)
00077 return 0;
00078
00079 List_item *l = _c;
00080 operator ++ ();
00081 l->remove_me();
00082
00083 if (_f == l)
00084 _f = _c;
00085
00086 return l;
00087 }
00088
00089 private:
00090 List_item *_c, *_f;
00091 };
00092
00093 template< typename T, bool Poly = false>
00094 class T_iter : public Iter
00095 {
00096 private:
00097 static bool const P = !Conversion<const T*, const List_item *>::exists
00098 || Poly;
00099
00100 static List_item *cast_to_li(T *i, Int_to_type<true>) noexcept
00101 { return dynamic_cast<List_item*>(i); }
00102
00103 static List_item *cast_to_li(T *i, Int_to_type<false>) noexcept
00104 { return i; }
00105
00106 static T *cast_to_type(List_item *i, Int_to_type<true>) noexcept
00107 { return dynamic_cast<T*>(i); }
00108
00109 static T *cast_to_type(List_item *i, Int_to_type<false>) noexcept
00110 { return static_cast<T*>(i); }
00111
00112 public:
00113 template< typename O >
00114 explicit T_iter(T_iter<O> const &o) noexcept
00115 : Iter(o) { dynamic_cast<T*>(&o); }
00116
00117 //T_iter(CListItem *f) : Iter(f) {}
00118 T_iter(T *f = 0) noexcept : Iter(cast_to_li(f, Int_to_type<P>())) {}
00119 T_iter(T *c, T *f) noexcept
00120 : Iter(cast_to_li(c, Int_to_type<P>()),
00121 cast_to_li(f, Int_to_type<P>()))

```

```

00137 {}
00138
00139 inline T *operator * () const noexcept
00140 { return cast_to_type(Iter::operator * (), Int_to_type<P>()); }
00141 inline T *operator -> () const noexcept
00142 { return operator * (); }
00143
00144 T_iter<T, Poly> operator ++ (int) noexcept
00145 { T_iter<T, Poly> o = *this; Iter::operator ++ (); return o; }
00146 T_iter<T, Poly> operator -- (int) noexcept
00147 { T_iter<T, Poly> o = *this; Iter::operator -- (); return o; }
00148 T_iter<T, Poly> &operator ++ () noexcept
00149 { Iter::operator ++ (); return *this; }
00150 T_iter<T, Poly> &operator -- () noexcept
00151 { Iter::operator -- (); return *this; }
00152 inline T *remove_me() noexcept;
00153 };
00154
00155 List_item() noexcept : _n(this), _p(this) {}
00156
00157 protected:
00158 List_item(List_item const &) noexcept : _n(this), _p(this) {}
00159
00160 public:
00162 List_item *get_prev_item() const noexcept { return _p; }
00163
00165 List_item *get_next_item() const noexcept { return _n; }
00166
00168 void insert_prev_item(List_item *p) noexcept
00169 {
00170 p->_p->_n = this;
00171 List_item *pr = p->_p;
00172 p->_p = _p;
00173 _p->_n = p;
00174 _p = pr;
00175 }
00176
00178 void insert_next_item(List_item *p) noexcept
00179 {
00180 p->_p->_n = _n;
00181 p->_p = this;
00182 _n->_p = p;
00183 _n = p;
00184 }
00185
00187 void remove_me() noexcept
00188 {
00189 if (_p != this)
00190 {
00191 _p->_n = _n;
00192 _n->_p = _p;
00193 }
00194 _p = _n = this;
00195 }
00196
00205 template< typename C, typename N >
00206 static inline C *push_back(C *head, N *p) noexcept;
00207
00216 template< typename C, typename N >
00217 static inline C *push_front(C *head, N *p) noexcept;
00218
00227 template< typename C, typename N >
00228 static inline C *remove(C *head, N *p) noexcept;
00229
00230 private:
00231 List_item *_n, *_p;
00232 };
00233
00234
00235 /* IMPLEMENTATION -----*/
00236 template< typename C, typename N >
00237 C *List_item::push_back(C *h, N *p) noexcept
00238 {
00239 if (!p)
00240 return h;
00241 if (!h)
00242 return p;
00243 h->insert_prev_item(p);
00244 return h;
00245 }
00246
00247 template< typename C, typename N >
00248 C *List_item::push_front(C *h, N *p) noexcept
00249 {
00250 if (!p)
00251 return h;
00252 if (h)

```

```

00253 h->insert_prev_item(p);
00254 return p;
00255 }
00256
00257 template< typename C, typename N >
00258 C *List_item::remove(C *h, N *p) noexcept
00259 {
00260 if (!p)
00261 return h;
00262 if (!h)
00263 return 0;
00264 if (h == p)
00265 {
00266 if (p == p->_n)
00267 h = 0;
00268 else
00269 h = static_cast<C*>(p->_n);
00270 }
00271 p->remove_me();
00272 return h;
00273 }
00274
00275 template< typename T, bool Poly >
00276 inline
00277 T *List_item::T_iter<T, Poly>::remove_me() noexcept
00278 { return cast_to_type(Iter::remove_me(), Int_to_type<P>()); }
00279
00280 template< typename T >
00281 class T_list_item : public List_item
00282 {
00283 public:
00284 T *next() const { return static_cast<T*>(List_item::get_next_item()); }
00285 T *prev() const { return static_cast<T*>(List_item::get_prev_item()); }
00286 };
00287
00288 template< typename LI >
00289 class L_list
00290 {
00291 private:
00292 LI *_h;
00293 public:
00294 L_list() : _h(0) {}
00295
00296 void push_front(LI *e) { _h = LI::push_front(_h, e); }
00297 void push_back(LI *e) { _h = LI::push_back(_h, e); }
00298 void insert_before(LI *e, LI *p)
00299 {
00300 p->insert_prev_item(e);
00301 if (_h == p)
00302 _h = e;
00303 }
00304 void insert_after(LI *e, LI *p) { p->insert_next_item(e); }
00305
00306 void remove(LI *e)
00307 { _h = LI::remove(_h, e); }
00308
00309 LI *head() const { return _h; }
00310 };
00311
00312 template< typename D, template<typename A> class Alloc = New_allocator >
00313 class List
00314 {
00315 private:
00316 class E : public List_item
00317 {
00318 public:
00319 E(D const &d) noexcept : data(d) {}
00320 D data;
00321 };
00322
00323 class Node : private E
00324 {};
00325
00326 typedef Alloc<Node> Node_alloc;
00327
00328 class Iter
00329 {
00330 private:
00331 List_item::T_iter<E> _i;
00332 public:

```

```

00349 Iter(E *e) noexcept : _i(e) {}
00350
00351 D &operator * () const noexcept { return (*_i)->data; }
00352 D &operator -> () const noexcept { return (*_i)->data; }
00353
00354 Iter operator ++ (int) noexcept
00355 { Iter o = *this; operator ++ (); return o; }
00356 Iter operator -- (int) noexcept
00357 { Iter o = *this; operator -- (); return o; }
00358 Iter &operator ++ () noexcept { ++_i; return *this; }
00359 Iter &operator -- () noexcept { --_i; return *this; }
00360
00362 operator E* () const noexcept { return *_i; }
00363 };
00364
00365 List(Alloc<Node> const &a = Alloc<Node>()) noexcept : _h(0), _l(0), _a(a) {}
00366
00368 void push_back(D const &d) noexcept
00369 {
00370 void *n = _a.alloc();
00371 if (!n) return;
00372 _h = E::push_back(_h, new (n) E(d));
00373 ++_l;
00374 }
00375
00377 void push_front(D const &d) noexcept
00378 {
00379 void *n = _a.alloc();
00380 if (!n) return;
00381 _h = E::push_front(_h, new (n) E(d));
00382 ++_l;
00383 }
00384
00386 void remove(Iter const &i) noexcept
00387 { E *e = i; _h = E::remove(_h, e); --_l; _a.free(e); }
00388
00390 unsigned long size() const noexcept { return _l; }
00391
00393 D const &operator [] (unsigned long idx) const noexcept
00394 { Iter i = _h; for (; idx && *i; ++i, --idx) { } return *i; }
00395
00397 D &operator [] (unsigned long idx) noexcept
00398 { Iter i = _h; for (; idx && *i; ++i, --idx) { } return *i; }
00399
00401 Iter items() noexcept { return Iter(_h); }
00402
00403 private:
00404 E *_h;
00405 unsigned _l;
00406 Alloc<Node> _a;
00407 };
00408
00409
00410 };
00411

```

## 17.198 list\_alloc

```

00001 // vim:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>,
00004 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009
00010 #pragma once
00011
00012 #include <l4/cxx/arith>
00013 #include <l4/cxx/minmax>
00014 #include <l4/sys/consts.h>
00015
00016 namespace cxx {
00017
00021 class List_alloc
00022 {
00023 private:
00024 friend class List_alloc_sanity_guard;
00025
00026 struct Mem_block
00027 {
00028 Mem_block *next;

```



```

00029 unsigned long size;
00030 };
00031
00032 Mem_block *_first;
00033
00034 inline void check_overlap(void *, unsigned long);
00035 inline void sanity_check_list(char const *, char const *);
00036 inline void merge();
00037
00038 public:
00039
00046 List_alloc() : _first(0) {}
00047
00060 inline void free(void *block, unsigned long size, bool initial_free = false);
00061
00076 inline void *alloc(unsigned long size, unsigned long align,
00077 unsigned long lower = 0, unsigned long upper = ~0UL);
00078
00099 inline void *alloc_max(unsigned long min, unsigned long *max,
00100 unsigned long align, unsigned granularity,
00101 unsigned long lower = 0, unsigned long upper = ~0UL);
00102
00108 inline unsigned long avail();
00109
00110 template <typename DBG>
00111 void dump_free_list(DBG &out);
00112 };
00113
00114 #if !defined (CXX_LIST_ALLOC_SANITY)
00115 class List_alloc_sanity_guard
00116 {
00117 public:
00118 List_alloc_sanity_guard(List_alloc *, char const *)
00119 {}
00120 };
00121
00122
00123 void
00124 List_alloc::check_overlap(void *, unsigned long)
00125 {}
00126
00127 void
00128 List_alloc::sanity_check_list(char const *, char const *)
00129 {}
00130
00131
00132 #else
00133
00134 class List_alloc_sanity_guard
00135 {
00136 private:
00137 List_alloc *a;
00138 char const *func;
00139
00140 public:
00141 List_alloc_sanity_guard(List_alloc *a, char const *func)
00142 : a(a), func(func)
00143 { a->sanity_check_list(func, "entry"); }
00144
00145 ~List_alloc_sanity_guard()
00146 { a->sanity_check_list(func, "exit"); }
00147 };
00148
00149 void
00150 List_alloc::check_overlap(void *b, unsigned long s)
00151 {
00152 unsigned long const mb_align = (1UL << arith::Ld<sizeof(Mem_block)>::value) - 1;
00153 if ((unsigned long)b & mb_align)
00154 {
00155 L4::cerr << "List_alloc(FATAL): trying to free unaligned memory: "
00156 << b << " align=" << arith::Ld<sizeof(Mem_block)>::value << "\n";
00157 }
00158
00159 Mem_block *c = _first;
00160 for (; c ; c = c->next)
00161 {
00162 unsigned long x_s = (unsigned long)b;
00163 unsigned long x_e = x_s + s;
00164 unsigned long b_s = (unsigned long)c;
00165 unsigned long b_e = b_s + c->size;
00166
00167 if ((x_s >= b_s && x_s < b_e)
00168 || (x_e > b_s && x_e <= b_e)
00169 || (b_s >= x_s && b_s < x_e)
00170 || (b_e > x_s && b_e <= x_e))
00171 {
00172 L4::cerr << "List_alloc(FATAL): trying to free memory that "

```

```

00173 "is already free: \n ["
00174 « (void*)x_s « '-' « (void*)x_e « ") overlaps ["
00175 « (void*)b_s « '-' « (void*)b_e « ")\n";
00176 }
00177 }
00178 }
00179
00180 void
00181 List_alloc::sanity_check_list(char const *func, char const *info)
00182 {
00183 Mem_block *c = _first;
00184 for (; c ; c = c->next)
00185 {
00186 if (c->next)
00187 {
00188 if (c >= c->next)
00189 {
00190 L4::cerr « "List_alloc(FATAL): " « func « '(' « info
00191 « "): list order violation\n";
00192 }
00193
00194 if (((unsigned long)c) + c->size > (unsigned long)c->next)
00195 {
00196 L4::cerr « "List_alloc(FATAL): " « func « '(' « info
00197 « "): list order violation\n";
00198 }
00199 }
00200 }
00201 }
00202
00203 #endif
00204
00205 void
00206 List_alloc::merge()
00207 {
00208 List_alloc_sanity_guard __attribute__((unused)) guard(this, __func__);
00209 Mem_block *c = _first;
00210 while (c && c->next)
00211 {
00212 unsigned long f_start = reinterpret_cast<unsigned long>(c);
00213 unsigned long f_end = f_start + c->size;
00214 unsigned long n_start = reinterpret_cast<unsigned long>(c->next);
00215
00216 if (f_end == n_start)
00217 {
00218 c->size += c->next->size;
00219 c->next = c->next->next;
00220 continue;
00221 }
00222
00223 c = c->next;
00224 }
00225 }
00226
00227 void
00228 List_alloc::free(void *block, unsigned long size, bool initial_free)
00229 {
00230 List_alloc_sanity_guard __attribute__((unused)) guard(this, __func__);
00231
00232 unsigned long const mb_align = (1UL « arith::Ld<sizeof(Mem_block)>::value) - 1;
00233
00234 if (initial_free)
00235 {
00236 // enforce alignment constraint on initial memory
00237 unsigned long nblock = (reinterpret_cast<unsigned long>(block) + mb_align)
00238 & ~mb_align;
00239 size = (size - (nblock - reinterpret_cast<unsigned long>(block)))
00240 & ~mb_align;
00241 block = reinterpret_cast<void*>(nblock);
00242 }
00243 else
00244 // blow up size to the minimum aligned size
00245 size = (size + mb_align) & ~mb_align;
00246
00247 check_overlap(block, size);
00248
00249 Mem_block **c = &_first;
00250 Mem_block *next = 0;
00251
00252 if (*c)
00253 {
00254 while (*c && *c < block)
00255 c = &(*c)->next;
00256
00257 next = *c;
00258 }
00259

```

```

00260 *c = reinterpret_cast<Mem_block*>(block);
00261
00262 (*c)->next = next;
00263 (*c)->size = size;
00264
00265 merge();
00266 }
00267
00268 void *
00269 List_alloc::alloc_max(unsigned long min, unsigned long *max, unsigned long align,
00270 unsigned granularity, unsigned long lower,
00271 unsigned long upper)
00272 {
00273 List_alloc_sanity_guard __attribute__((unused)) guard(this, __func__);
00274
00275 unsigned char const mb_bits = arith::Ld<sizeof(Mem_block)>::value;
00276 unsigned long const mb_align = (1UL << mb_bits) - 1;
00277
00278 // blow minimum up to at least the minimum aligned size of a Mem_block
00279 min = l4_round_size(min, mb_bits);
00280 // truncate maximum to at least the size of a Mem_block
00281 *max = l4_trunc_size(*max, mb_bits);
00282 // truncate maximum size according to granularity
00283 *max = *max & ~(granularity - 1UL);
00284
00285 if (min > *max)
00286 return 0;
00287
00288 unsigned long almask = align ? (align - 1UL) : 0;
00289
00290 // minimum alignment is given by the size of a Mem_block
00291 if (almask < mb_align)
00292 almask = mb_align;
00293
00294 Mem_block **c = &_first;
00295 Mem_block **fit = 0;
00296 unsigned long max_fit = 0;
00297 unsigned long a_lower = (lower + almask) & ~almask;
00298
00299 for (; *c; c = &(*c)->next)
00300 {
00301 // address of free memory block
00302 unsigned long n_start = reinterpret_cast<unsigned long>(*c);
00303
00304 // block too small, next
00305 // XXX: maybe we can skip this and just do the test below
00306 if ((*c)->size < min)
00307 continue;
00308
00309 // block outside region, next
00310 if (upper < n_start || a_lower > n_start + (*c)->size)
00311 continue;
00312
00313 // aligned start address within the free block
00314 unsigned long a_start = (n_start + almask) & ~almask;
00315
00316 // check if aligned start address is behind the block, next
00317 if (a_start - n_start >= (*c)->size)
00318 continue;
00319
00320 a_start = a_start < a_lower ? a_lower : a_start;
00321
00322 // end address would overflow, next
00323 if (min > ~0UL - a_start)
00324 continue;
00325
00326 // block outside region, next
00327 if (a_start + min - 1UL > upper)
00328 continue;
00329
00330 // remaining size after subtracting the padding for the alignment
00331 unsigned long r_size = (*c)->size - a_start + n_start;
00332
00333 // upper limit can limit maximum size
00334 if (a_start + r_size - 1UL > upper)
00335 r_size = upper - a_start + 1UL;
00336
00337 // round down according to granularity
00338 r_size &= ~(granularity - 1UL);
00339
00340 // block too small
00341 if (r_size < min)
00342 continue;
00343
00344 if (r_size >= *max)
00345 {
00346 fit = c;

```

```

00347 max_fit = *max;
00348 break;
00349 }
00350
00351 if (r_size > max_fit)
00352 {
00353 max_fit = r_size;
00354 fit = c;
00355 }
00356 }
00357
00358 if (fit)
00359 {
00360 unsigned long n_start = reinterpret_cast<unsigned long>(*fit);
00361 unsigned long a_lower = (lower + almask) & ~almask;
00362 unsigned long a_start = (n_start + almask) & ~almask;
00363 a_start = a_start < a_lower ? a_lower : a_start;
00364 unsigned long r_size = (*fit)->size - a_start + n_start;
00365
00366 if (a_start > n_start)
00367 {
00368 (*fit)->size -= r_size;
00369 fit = &(*fit)->next;
00370 }
00371 else
00372 *fit = (*fit)->next;
00373
00374 *max = max_fit;
00375 if (r_size == max_fit)
00376 return reinterpret_cast<void *>(a_start);
00377
00378 Mem_block *m = reinterpret_cast<Mem_block*>(a_start + max_fit);
00379 m->next = *fit;
00380 m->size = r_size - max_fit;
00381 *fit = m;
00382 return reinterpret_cast<void *>(a_start);
00383 }
00384
00385 return 0;
00386 }
00387
00388 void *
00389 List_alloc::alloc(unsigned long size, unsigned long align, unsigned long lower,
00390 unsigned long upper)
00391 {
00392 List_alloc_sanity_guard __attribute__((unused)) guard(this, __func__);
00393
00394 unsigned long const mb_align
00395 = (1UL << arith::Ld<sizeof(Mem_block)>::value) - 1;
00396
00397 // blow up size to the minimum aligned size
00398 size = (size + mb_align) & ~mb_align;
00399
00400 unsigned long almask = align ? (align - 1UL) : 0;
00401
00402 // minimum alignment is given by the size of a Mem_block
00403 if (almask < mb_align)
00404 almask = mb_align;
00405
00406 Mem_block **c = &_first;
00407 unsigned long a_lower = (lower + almask) & ~almask;
00408
00409 for (; *c; c=&(*c)->next)
00410 {
00411 // address of free memory block
00412 unsigned long n_start = reinterpret_cast<unsigned long>(*c);
00413
00414 // block too small, next
00415 // XXX: maybe we can skip this and just do the test below
00416 if ((*c)->size < size)
00417 continue;
00418
00419 // block outside region, next
00420 if (upper < n_start || a_lower > n_start + (*c)->size)
00421 continue;
00422
00423 // aligned start address within the free block
00424 unsigned long a_start = (n_start + almask) & ~almask;
00425
00426 // block too small after alignment, next
00427 if (a_start - n_start >= (*c)->size)
00428 continue;
00429
00430 a_start = a_start < a_lower ? a_lower : a_start;
00431
00432 // end address would overflow, next
00433 if (size > ~0UL - a_start)

```

```

00434 continue;
00435
00436 // block outside region, next
00437 if (a_start + size - 1UL > upper)
00438 continue;
00439
00440 // remaining size after subtracting the padding
00441 // for the alignment
00442 unsigned long r_size = (*c)->size - a_start + n_start;
00443
00444 // block too small
00445 if (r_size < size)
00446 continue;
00447
00448 if (a_start > n_start)
00449 {
00450 // have free space before the allocated block
00451 // shrink the block and set c to the next pointer of that
00452 // block
00453 (*c)->size -= r_size;
00454 c = &(*c)->next;
00455 }
00456 else
00457 // drop the block, c remains the next pointer of the
00458 // previous block
00459 *c = (*c)->next;
00460
00461 // allocated the whole remaining space
00462 if (r_size == size)
00463 return reinterpret_cast<void*>(a_start);
00464
00465 // add a new free block behind the allocated block
00466 Mem_block *m = reinterpret_cast<Mem_block*>(a_start + size);
00467 m->next = *c;
00468 m->size = r_size - size;
00469 *c = m;
00470 return reinterpret_cast<void *>(a_start);
00471 }
00472
00473 return 0;
00474 }
00475
00476 unsigned long
00477 List_alloc::avail()
00478 {
00479 List_alloc_sanity_guard __attribute__((unused)) guard(this, __FUNCTION__);
00480 Mem_block *c = _first;
00481 unsigned long a = 0;
00482 while (c)
00483 {
00484 a += c->size;
00485 c = c->next;
00486 }
00487
00488 return a;
00489 }
00490
00491 template <typename DBG>
00492 void
00493 List_alloc::dump_free_list(DBG &out)
00494 {
00495 Mem_block *c = _first;
00496 while (c)
00497 {
00498 unsigned sz;
00499 const char *unit;
00500
00501 if (c->size < 1024)
00502 {
00503 sz = c->size;
00504 unit = "Byte";
00505 }
00506 else if (c->size < 1 « 20)
00507 {
00508 sz = c->size » 10;
00509 unit = "KB";
00510 }
00511 else
00512 {
00513 sz = c->size » 20;
00514 unit = "MB";
00515 }
00516
00517 out.printf("%12p - %12p (%u %s)\n", c,
00518 reinterpret_cast<char *>(c) + c->size - 1, sz, unit);
00519
00520 c = c->next;

```

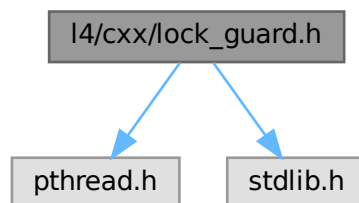
```
00521 }
00522 }
00523
00524 }
```

## 17.199 I4/cxx/lock\_guard.h File Reference

Lock guard implementation.

```
#include <pthread.h>
#include <stdlib.h>
```

Include dependency graph for lock\_guard.h:



### Data Structures

- class [L4::Lock\\_guard](#)

*Basic lock guard implementation that prevents forgotten unlocks on exit paths from a method or a block of code.*

### Namespaces

- namespace [L4](#)

[L4](#) *low-level kernel interface.*

### 17.199.1 Detailed Description

Lock guard implementation.

Definition in file [lock\\_guard.h](#).

## 17.200 lock\_guard.h

[Go to the documentation of this file.](#)

```

00001 // vim:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * Copyright (C) 2025 Kernkonzept GmbH.
00004 * Author(s): Martin Decky <martin.decky@kernkonzept.com>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00013
00014 #pragma once
00015
00016 #include <pthread.h>
00017 #include <stdlib.h>
00018
00019 namespace L4 {
00020
00044 class Lock_guard
00045 {
00046 public:
00047 Lock_guard() = delete;
00048 Lock_guard(const Lock_guard &) = delete;
00049 Lock_guard &operator=(const Lock_guard &) = delete;
00050
00059 explicit Lock_guard(pthread_mutex_t &lock) : _lock(&lock)
00060 {
00061 _status = pthread_mutex_lock(_lock);
00062 }
00063
00071 Lock_guard(Lock_guard &&guard) : _lock(guard._lock), _status(guard._status)
00072 {
00073 guard.release();
00074 }
00075
00090 Lock_guard &operator=(Lock_guard &&guard)
00091 {
00092 // Unlock the currently associated mutex (if any).
00093 reset();
00094
00095 // Move the state from the other guard.
00096 _lock = guard._lock;
00097 _status = guard._status;
00098
00099 // Release the mutex from the other guard.
00100 guard.release();
00101
00102 return *this;
00103 }
00104
00110 int status() const
00111 {
00112 return _status;
00113 }
00114
00126 ~Lock_guard()
00127 {
00128 reset();
00129 }
00130
00131 private:
00137 void release()
00138 {
00139 _lock = nullptr;
00140 }
00141
00150 void reset()
00151 {
00152 // No mutex might be associated with this lock guard only if the mutex has
00153 // been moved to a different lock guard.
00154 if (_lock)
00155 {
00156 _status = pthread_mutex_unlock(_lock);
00157 release();
00158 }
00159 }
00160
00161 pthread_mutex_t *_lock;
00162 int _status;
00163 };
00164
00165 } // namespace L4

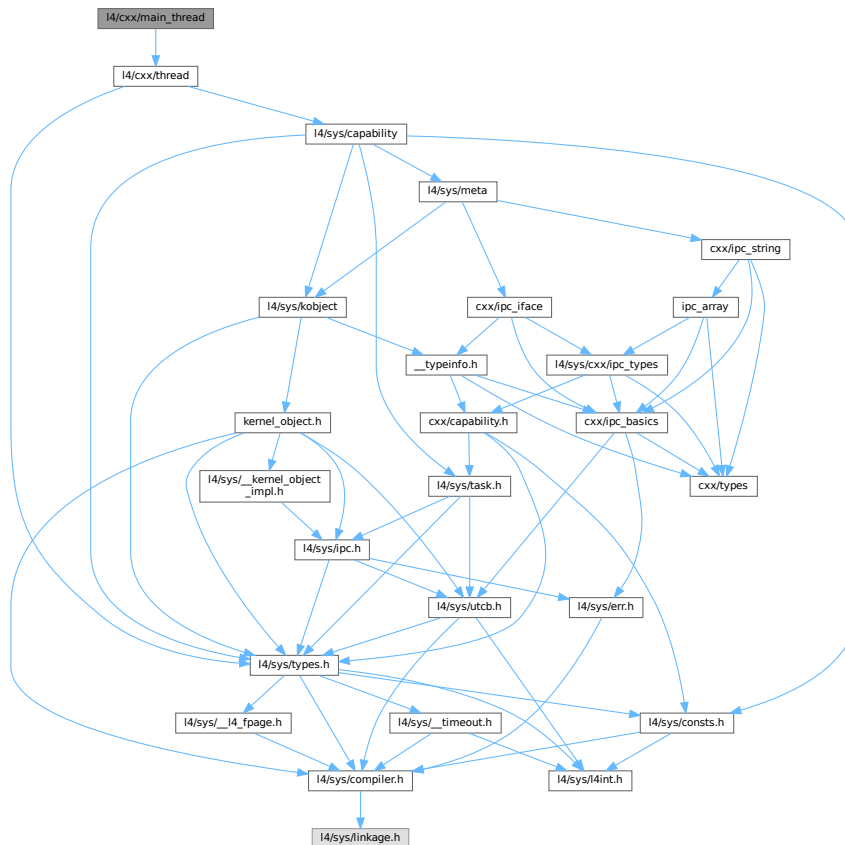
```

## 17.201 I4/cxx/main\_thread File Reference

Main thread.

```
#include <l4/cxx/thread>
```

Include dependency graph for main\_thread:



### Namespaces

- namespace `cxx`  
*Our C++ library.*

### 17.201.1 Detailed Description

Main thread.

Definition in file [main\\_thread](#).



## 17.202 main\_thread

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2004-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * This file is part of TUD:OS and distributed under the terms of the
00011 * GNU General Public License 2.
00012 * Please see the COPYING-GPL-2 file for details.
00013 *
00014 * As a special exception, you may use this file as part of a free software
00015 * library without restriction. Specifically, if other files instantiate
00016 * templates or use macros or inline functions from this file, or you compile
00017 * this file and link it with other files to produce an executable, this
00018 * file does not by itself cause the resulting executable to be covered by
00019 * the GNU General Public License. This exception does not however
00020 * invalidate any other reasons why the executable file might be covered by
00021 * the GNU General Public License.
00022 */
00023
00024 #ifndef L4_CXX_MAIN_THREAD_H__
00025 #define L4_CXX_MAIN_THREAD_H__
00026
00027 #include <l4/cxx/thread>
00028
00029 namespace cxx {
00030 class MainThread : public Thread
00031 {
00032 public:
00033 MainThread() : Thread(true)
00034 {}
00035 };
00036 };
00037
00038 #endif /* L4_CXX_MAIN_THREAD_H__ */

```

## 17.203 minmax

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include "type_traits"
00011
00012 namespace cxx {
00013 {
00014 // trivial, used to terminate the variadic recursion
00015 template<typename A>
00016 constexpr A const &
00017 min(A const &a)
00018 { return a; }
00019
00020 template<typename A, typename ...ARGS>
00021 constexpr A const &
00022 min(A const &a1, A const &a2, ARGS const &...a)
00023 {
00024 return min((a1 <= a2) ? a1 : a2, a...);
00025 }
00026
00027 template<typename A, typename ...ARGS>
00028 constexpr A const &
00029 min(cxx::identity_t<A> const &a1,
00030 cxx::identity_t<A> const &a2,
00031 ARGS const &...a)
00032 {
00033 return min<A>((a1 <= a2) ? a1 : a2, a...);
00034 }
00035
00036 // trivial, used to terminate the variadic recursion
00037 template<typename A>
00038 constexpr A const &
00039 max(A const &a)

```

```

00064 { return a; }
00065
00076 template<typename A, typename ...ARGS>
00077 constexpr A const &
00078 max(A const &a1, A const &a2, ARGS const &...a)
00079 { return max((a1 >= a2) ? a1 : a2, a...); }
00080
00091 template<typename A, typename ...ARGS>
00092 constexpr A const &
00093 max(cxx::identity_t<A> const &a1,
00094 cxx::identity_t<A> const &a2,
00095 ARGS const &...a)
00096 {
00097 return max<A>((a1 >= a2) ? a1 : a2, a...);
00098 }
00099
00107 template< typename T1 >
00108 inline
00109 T1 clamp(T1 v, T1 lo, T1 hi)
00110 { return min(hi, max(lo, v)); }
00111 };

```

## 17.204 numeric

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * Copyright (C) 2025 Kernkonzept GmbH.
00004 * Author(s): Martin Decky <martin.decky@kernkonzept.com>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00009 #pragma once
00010
00011 #include <l4/cxx/type_traits>
00012
00013 namespace cxx {
00014
00033 template <typename T>
00034 constexpr T gcd(T a, T b)
00035 {
00036 static_assert(Type_traits<T>::is_unsigned, "Type must be unsigned");
00037
00038 while (b != 0)
00039 {
00040 T remainder = a % b;
00041 a = b;
00042 b = remainder;
00043 }
00044
00045 return a;
00046 }
00047
00067 template <typename T>
00068 constexpr T lcm(T a, T b)
00069 {
00070 static_assert(Type_traits<T>::is_unsigned, "Type must be unsigned");
00071
00072 if (a == 0 || b == 0)
00073 return 0;
00074
00075 return (a / gcd(a, b)) * b;
00076 }
00077
00078 }

```

## 17.205 observer

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2010 Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include <l4/cxx/hlist>

```

```

00011
00012 namespace cxx {
00013
00014 class Observer : public H_list_item
00015 {
00016 public:
00017 virtual void notify() = 0;
00018 };
00019
00020 class Notifier : public H_list<Observer>
00021 {
00022 public:
00023 void notify()
00024 {
00025 for (Iterator i = begin(); i != end(); ++i)
00026 i->notify();
00027 }
00028 };
00029
00030 }
00031
00032

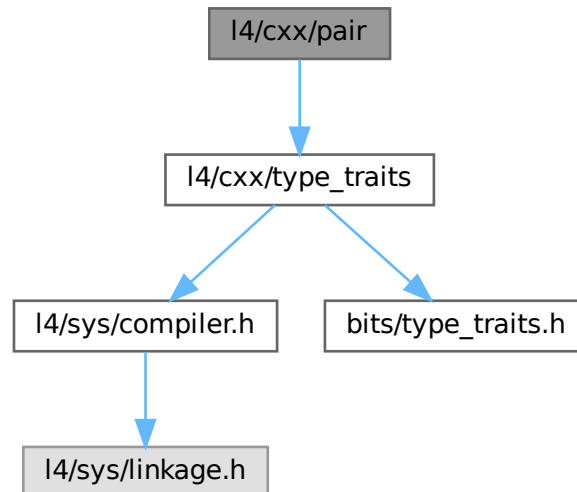
```

## 17.206 l4/cxx/pair File Reference

Pair implementation.

```
#include <l4/cxx/type_traits>
```

Include dependency graph for pair:





```

00045 Pair(A1 &&first, A2 &&second)
00046 : first(cxx::forward<A1>(first)), second(cxx::forward<A2>(second)) {}
00047
00052 template<typename A1>
00053 Pair(A1 &&first)
00054 : first(cxx::forward<A1>(first)), second() {}
00055
00057 Pair() = default;
00058 };
00059
00060 template< typename F, typename S >
00061 Pair<F,S> pair(F const &f, S const &s)
00062 { return cxx::Pair<F,S>(f,s); }
00063
00064
00073 template< typename Cmp, typename Typ >
00074 class Pair_first_compare
00075 {
00076 private:
00077 Cmp const &_cmp;
00078
00079 public:
00084 Pair_first_compare(Cmp const &cmp = Cmp()) : _cmp(cmp) {}
00085
00091 bool operator () (Typ const &l, Typ const &r) const
00092 { return _cmp(l.first,r.first); }
00093 };
00094
00095 }
00096
00097 template< typename OS, typename A, typename B >
00098 inline
00099 OS &operator « (OS &os, cxx::Pair<A,B> const &p)
00100 {
00101 os « p.first « ';<' « p.second;
00102 return os;
00103 }
00104

```

## 17.208 ref\_ptr

```

00001 // vim:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00009 #pragma once
00010
00011 #include "type_traits"
00012 #include <stddef.h>
00013 #include <14/sys/compiler.h>
00014
00015 namespace cxx {
00016
00017 template< typename T >
00018 struct Default_ref_counter
00019 {
00020 void h_drop_ref(T *p) noexcept
00021 {
00022 if (p->remove_ref() == 0)
00023 delete p;
00024 }
00025
00026 void h_take_ref(T *p) noexcept
00027 {
00028 p->add_ref();
00029 }
00030 };
00031
00032 struct Ref_ptr_base
00033 {
00034 enum Default_value
00035 { Nil = 0 };
00036 };
00037
00038 template<typename T, template< typename X > class CNT = Default_ref_counter>
00039 class Weak_ptr;
00040
00066 template <
00067 typename T = void,

```

```

00068 template< typename X > class CNT = Default_ref_counter
00069 >
00070 class Ref_ptr : public Ref_ptr_base, private CNT<T>
00071 {
00072 private:
00073 typedef decltype(nullptr) Null_type;
00074 typedef Weak_ptr<T, CNT> Wp;
00075
00076 public:
00077 Ref_ptr() noexcept : _p(0) {}
00078
00079 Ref_ptr(Ref_ptr_base::Default_value v)
00080 : _p(reinterpret_cast<T*>(static_cast<unsigned long>(v))) {}
00081
00082 Ref_ptr(Wp const &o) noexcept : _p(o.ptr())
00083 { __take_ref(); }
00084
00085 Ref_ptr(decltype(nullptr) n) noexcept : _p(n) {}
00086
00087 template<typename X>
00088 explicit Ref_ptr(X *o) noexcept : _p(o)
00089 { __take_ref(); }
00090
00091 Ref_ptr(T *o, [[maybe_unused]] bool d) noexcept : _p(o) { }
00092
00093 T *get() const noexcept
00094 {
00095 return _p;
00096 }
00097
00098 T *ptr() const noexcept
00099 {
00100 return _p;
00101 }
00102
00103 T *release() noexcept
00104 {
00105 T *p = _p;
00106 _p = 0;
00107 return p;
00108 }
00109
00110 ~Ref_ptr() noexcept
00111 { __drop_ref(); }
00112
00113 template<typename OT>
00114 Ref_ptr(Ref_ptr<OT, CNT> const &o) noexcept
00115 {
00116 _p = o.ptr();
00117 __take_ref();
00118 }
00119
00120 Ref_ptr(Ref_ptr<T> const &o) noexcept
00121 {
00122 _p = o._p;
00123 __take_ref();
00124 }
00125
00126 template< typename OT >
00127 void operator = (Ref_ptr<OT> const &o) noexcept
00128 {
00129 __drop_ref();
00130 _p = o.ptr();
00131 __take_ref();
00132 }
00133
00134 void operator = (Ref_ptr<T> const &o) noexcept
00135 {
00136 if (&o == this)
00137 return;
00138
00139 __drop_ref();
00140 _p = o._p;
00141 __take_ref();
00142 }
00143
00144 void operator = (Null_type) noexcept
00145 {
00146 __drop_ref();
00147 _p = 0;
00148 }
00149
00150 template<typename OT>
00151 Ref_ptr(Ref_ptr<OT, CNT> &&o) noexcept
00152 { _p = o.release(); }
00153
00154 Ref_ptr(Ref_ptr<T> &&o) noexcept

```

```

00190 { _p = o.release(); }
00191
00192 template< typename OT >
00193 void operator = (Ref_ptr<OT> &o) noexcept
00194 {
00195 __drop_ref();
00196 _p = o.release();
00197 }
00198
00199 void operator = (Ref_ptr<T> &o) noexcept
00200 {
00201 if (&o == this)
00202 return;
00203
00204 __drop_ref();
00205 _p = o.release();
00206 }
00207
00208 [[nodiscard]] explicit operator bool () const noexcept { return _p; }
00209
00210 T *operator -> () const noexcept
00211 { return _p; }
00212
00213 [[nodiscard]] bool operator == (Ref_ptr const &o) const noexcept
00214 { return _p == o._p; }
00215
00216 [[nodiscard]] bool operator != (Ref_ptr const &o) const noexcept
00217 { return _p != o._p; }
00218
00219 [[nodiscard]] bool operator < (Ref_ptr const &o) const noexcept
00220 { return _p < o._p; }
00221
00222 [[nodiscard]] bool operator <= (Ref_ptr const &o) const noexcept
00223 { return _p <= o._p; }
00224
00225 [[nodiscard]] bool operator > (Ref_ptr const &o) const noexcept
00226 { return _p > o._p; }
00227
00228 [[nodiscard]] bool operator >= (Ref_ptr const &o) const noexcept
00229 { return _p >= o._p; }
00230
00231 [[nodiscard]] bool operator == (T const *o) const noexcept
00232 { return _p == o; }
00233
00234 [[nodiscard]] bool operator < (T const *o) const noexcept
00235 { return _p < o; }
00236
00237 [[nodiscard]] bool operator <= (T const *o) const noexcept
00238 { return _p <= o; }
00239
00240 [[nodiscard]] bool operator > (T const *o) const noexcept
00241 { return _p > o; }
00242
00243 [[nodiscard]] bool operator >= (T const *o) const noexcept
00244 { return _p >= o; }
00245
00246 private:
00247 void __drop_ref() noexcept
00248 {
00249 if (_p)
00250 static_cast<CNT<T>*>(this)->h_drop_ref(_p);
00251 }
00252
00253 void __take_ref() noexcept
00254 {
00255 if (_p)
00256 static_cast<CNT<T>*>(this)->h_take_ref(_p);
00257 }
00258
00259 T *_p;
00260 };
00261
00262
00263 template<typename T, template< typename X > class CNT>
00264 class Weak_ptr
00265 {
00266 private:
00267 struct Null_type;
00268 typedef Ref_ptr<T, CNT> Rp;
00269
00270 public:
00271 Weak_ptr() = default;
00272 Weak_ptr(decltype(nullptr)) : _p(nullptr) {}
00273 // backwards 0 ctor
00274 explicit Weak_ptr(int x) noexcept
00275 L4_DEPRECATED("Use initialization from 'nullptr'")
00276 : _p(nullptr)

```

```

00277 { if (x != 0) __builtin_trap(); }
00278
00279 Weak_ptr(Rp const &o) noexcept : _p(o.ptr()) {}
00280 explicit Weak_ptr(T *o) noexcept : _p(o) {}
00281
00282 template<typename OT>
00283 Weak_ptr(Weak_ptr<OT, CNT> const &o) noexcept : _p(o.ptr()) {}
00284
00285 Weak_ptr(Weak_ptr<T, CNT> const &o) noexcept : _p(o._p) {}
00286
00287 Weak_ptr<T, CNT> &operator = (const Weak_ptr<T, CNT> &o) = default;
00288
00289 T *get() const noexcept { return _p; }
00290 T *ptr() const noexcept { return _p; }
00291
00292 T *operator -> () const noexcept { return _p; }
00293 operator Null_type const * () const noexcept
00294 { return reinterpret_cast<Null_type const*>(_p); }
00295
00296 private:
00297 T *_p;
00298 };
00299
00300 template<typename OT, typename T> inline
00301 Ref_ptr<OT> ref_ptr_static_cast(Ref_ptr<T> const &o)
00302 { return ref_ptr(static_cast<OT*>(o.ptr())); }
00303
00304 template< typename T >
00305 inline Ref_ptr<T> ref_ptr(T *t)
00306 { return Ref_ptr<T>(t); }
00307
00308 template< typename T >
00309 inline Weak_ptr<T> weak_ptr(T *t)
00310 { return Weak_ptr<T>(t); }
00311
00312
00313 class Ref_obj
00314 {
00315 private:
00316 mutable int _ref_cnt;
00317 public:
00318 Ref_obj() : _ref_cnt(0) {}
00319 void add_ref() const noexcept { ++_ref_cnt; }
00320 int remove_ref() const noexcept { return --_ref_cnt; }
00321 };
00322
00323
00324 template< typename T, typename... Args >
00325 Ref_ptr<T>
00326 make_ref_obj(Args &&... args)
00327 { return cxx::Ref_ptr<T>(new T(cxx::forward<Args>(args)...)); }
00328
00329 template<typename T, typename U>
00330 Ref_ptr<T>
00331 dynamic_pointer_cast(Ref_ptr<U> const &p) noexcept
00332 {
00333 // our constructor from a naked pointer increments the counter
00334 return Ref_ptr<T>(dynamic_cast<T *>(p.get()));
00335 }
00336
00337 template<typename T, typename U>
00338 Ref_ptr<T>
00339 static_pointer_cast(Ref_ptr<U> const &p) noexcept
00340 {
00341 // our constructor from a naked pointer increments the counter
00342 return Ref_ptr<T>(static_cast<T *>(p.get()));
00343 }
00344
00345 }

```

## 17.209 l4/cxx/ref\_ptr\_list File Reference

Implementation of a list of ref-ptr-managed objects.

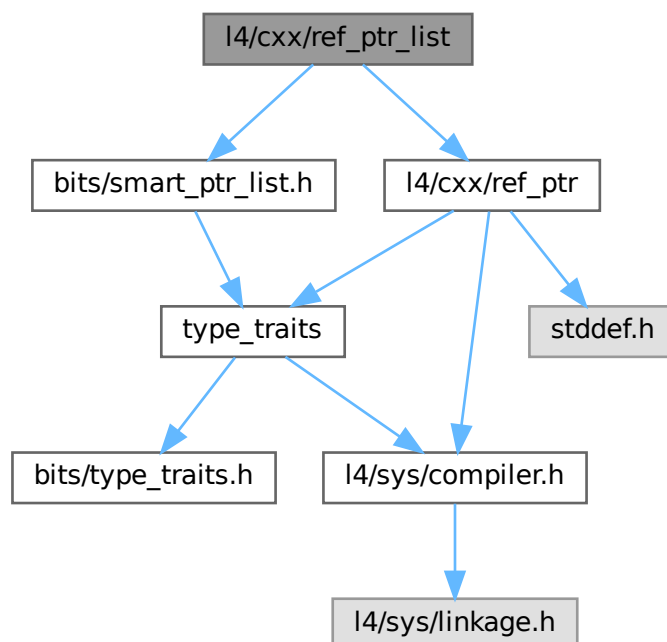
```

#include <l4/cxx/ref_ptr>
#include "bits/smart_ptr_list.h"

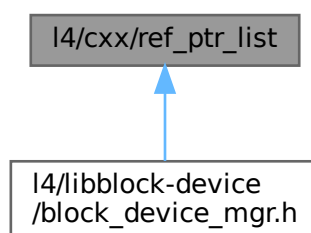
```



Include dependency graph for `ref_ptr_list`:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct `cxx::Ref_obj_list_item< T >`  
Item for list linked via `cxx::Ref_ptr` with default reference counting.

## Namespaces

- namespace `cxx`  
Our C++ library.

## Typedefs

- `template<typename T>`  
`using cxx::Ref_ptr_list_item = Bits::Smart_ptr_list_item<T, cxx::Ref_ptr<T> >`  
*Item for list linked with **cxx::Ref\_ptr**.*
- `template<typename T>`  
`using cxx::Ref_ptr_list = Bits::Smart_ptr_list<Ref_ptr_list_item<T> >`  
*Single-linked list where elements are connected via a **cxx::Ref\_ptr**.*

### 17.209.1 Detailed Description

Implementation of a list of ref-ptr-managed objects.

Definition in file [ref\\_ptr\\_list](#).

## 17.210 ref\_ptr\_list

[Go to the documentation of this file.](#)

```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * Copyright (C) 2018, 2022, 2024 Kernkonzept GmbH.
00008 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00014 #include <14/cxx/ref_ptr>
00015
00016 #include "bits/smart_ptr_list.h"
00017
00018 namespace cxx {
00019
00021 template <typename T>
00022 using Ref_ptr_list_item = Bits::Smart_ptr_list_item<T, cxx::Ref_ptr<T> >;
00023
00025 template <typename T>
00026 struct Ref_obj_list_item : public Ref_ptr_list_item<T>, public cxx::Ref_obj {};
00027
00031 template <typename T>
00032 using Ref_ptr_list = Bits::Smart_ptr_list<Ref_ptr_list_item<T> >;
00033
00034 }
```

## 17.211 slab\_alloc

```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009
00010 #pragma once
00011
00012 #include <14/cxx/std_alloc>
00013 #include <14/cxx/hlist>
00014 #include <14/sys/consts.h>
00015
00016 namespace cxx {
00017
00029 template< int Obj_size, int Slab_size = L4_PAGESIZE,
00030 int Max_free = 2, template<typename A> class Alloc = New_allocator >
00031 class Base_slab
```

```

00032 {
00033 private:
00034 struct Free_o
00035 {
00036 Free_o *next;
00037 };
00038
00039 protected:
00040 struct Slab_i;
00041
00042 private:
00043 struct Slab_head : public H_list_item
00044 {
00045 unsigned num_free;
00046 Free_o *free;
00047 Base_slab<Obj_size, Slab_size, Max_free, Alloc> *cache;
00048
00049 inline Slab_head() noexcept : num_free(0), free(0), cache(0)
00050 {}
00051 };
00052
00053 // In an empty or partially filled slab, each free object stores a pointer to
00054 // the next free object. Thus, the size of an object needs to be at least the
00055 // size of a pointer.
00056 static_assert(Obj_size >= sizeof(void *),
00057 "Object size must be at least the size of a pointer.");
00058 static_assert(Obj_size <= Slab_size - sizeof(Slab_head),
00059 "Object_size exceeds slab capability.");
00060
00061 public:
00062 enum
00063 {
00064 object_size = Obj_size,
00065 slab_size = Slab_size,
00066 objects_per_slab = (Slab_size - sizeof(Slab_head)) / object_size,
00067 max_free_slabs = Max_free,
00068 };
00069
00070 protected:
00071 struct Slab_store
00072 {
00073 char _o[slab_size - sizeof(Slab_head)];
00074 Free_o *object(unsigned obj) noexcept
00075 { return reinterpret_cast<Free_o*>(_o + object_size * obj); }
00076 };
00077
00078 struct Slab_i : public Slab_store, public Slab_head
00079 {};
00080
00081 public:
00082 typedef Alloc<Slab_i> Slab_alloc;
00083
00084 typedef void Obj_type;
00085
00086 private:
00087 Slab_alloc _alloc;
00088 unsigned _num_free;
00089 unsigned _num_slabs;
00090 H_list<Slab_i> _full_slabs;
00091 H_list<Slab_i> _partial_slabs;
00092 H_list<Slab_i> _empty_slabs;
00093
00094 void add_slab(Slab_i *s) noexcept
00095 {
00096 s->num_free = objects_per_slab;
00097 s->cache = this;
00098
00099 //L4::cerr << "Slab: " << this << "->add_slab(" << s << ", size="
00100 // << slab_size << "):" << " f=" << s->object(0) << '\n';
00101
00102 // initialize free list
00103 Free_o *f = s->free = s->object(0);
00104 for (unsigned i = 1; i < objects_per_slab; ++i)
00105 {
00106 f->next = s->object(i);
00107 f = f->next;
00108 }
00109 f->next = 0;
00110
00111 // insert slab into cache's list
00112 _empty_slabs.push_front(s);
00113 ++_num_slabs;
00114 ++_num_free;
00115 }
00116
00117 bool grow() noexcept
00118 {

```

```

00136 Slab_i *s = _alloc.alloc();
00137 if (!s)
00138 return false;
00139
00140 new (s, cxx::Nothrow()) Slab_i();
00141
00142 add_slab(s);
00143 return true;
00144 }
00145
00155 void shrink() noexcept
00156 {
00157 if (!_alloc.can_free)
00158 return;
00159
00160 while (!_empty_slabs.empty() && _num_free > max_free_slabs)
00161 {
00162 Slab_i *s = _empty_slabs.front();
00163 _empty_slabs.remove(s);
00164 --_num_free;
00165 --_num_slabs;
00166 _alloc.free(s);
00167 }
00168 }
00169
00170 public:
00171 Base_slab(Slab_alloc const &alloc = Slab_alloc()) noexcept
00172 : _alloc(alloc), _num_free(0), _num_slabs(0), _full_slabs(),
00173 _partial_slabs(), _empty_slabs()
00174 {}
00175
00176 ~Base_slab() noexcept
00177 {
00178 while (!_empty_slabs.empty())
00179 {
00180 Slab_i *o = _empty_slabs.front();
00181 _empty_slabs.remove(o);
00182 _alloc.free(o);
00183 }
00184 while (!_partial_slabs.empty())
00185 {
00186 Slab_i *o = _partial_slabs.front();
00187 _partial_slabs.remove(o);
00188 _alloc.free(o);
00189 }
00190 while (!_full_slabs.empty())
00191 {
00192 Slab_i *o = _full_slabs.front();
00193 _full_slabs.remove(o);
00194 _alloc.free(o);
00195 }
00196 }
00197
00207 void *alloc() noexcept
00208 {
00209 H_list<Slab_i> *free = &_partial_slabs;
00210 if (free->empty())
00211 free = &_empty_slabs;
00212
00213 if (free->empty() && !grow())
00214 return 0;
00215
00216 Slab_i *s = free->front();
00217 Free_o *o = s->free;
00218 s->free = o->next;
00219
00220 if (free == &_empty_slabs)
00221 {
00222 _empty_slabs.remove(s);
00223 --_num_free;
00224 }
00225
00226 --(s->num_free);
00227
00228 if (!s->free)
00229 {
00230 _partial_slabs.remove(s);
00231 _full_slabs.push_front(s);
00232 }
00233 else if (free == &_empty_slabs)
00234 _partial_slabs.push_front(s);
00235
00236 //L4::cerr << this << "->alloc(): " << o << ", of " << s << '\n';
00237
00238 return o;
00239 }
00240

```

```

00246 void free(void *_o) noexcept
00247 {
00248 if (!_o)
00249 return;
00250
00251 unsigned long addr = reinterpret_cast<unsigned long>(_o);
00252
00253 // find out the slab the object is in
00254 addr = (addr / slab_size) * slab_size;
00255 Slab_i *s = reinterpret_cast<Slab_i*>(addr);
00256
00257 if (s->cache != this)
00258 return;
00259
00260 Free_o *o = reinterpret_cast<Free_o*>(_o);
00261
00262 o->next = s->free;
00263 s->free = o;
00264
00265 bool was_full = false;
00266
00267 if (!s->num_free)
00268 {
00269 _full_slabs.remove(s);
00270 was_full = true;
00271 }
00272
00273 ++(s->num_free);
00274
00275 if (s->num_free == objects_per_slab)
00276 {
00277 if (!was_full)
00278 _partial_slabs.remove(s);
00279
00280 _empty_slabs.push_front(s);
00281 ++_num_free;
00282 if (_num_free > max_free_slabs)
00283 shrink();
00284
00285 was_full = false;
00286 }
00287 else if (was_full)
00288 _partial_slabs.push_front(s);
00289
00290 //L4::cerr << this << "->free(" << _o << "): of " << s << '\n';
00291 }
00292
00299 unsigned total_objects() const noexcept
00300 { return _num_slabs * objects_per_slab; }
00301
00308 unsigned free_objects() const noexcept
00309 {
00310 unsigned count = 0;
00311
00312 /* count partial slabs first */
00313 for (typename H_list<Slab_i>::Const_iterator s = _partial_slabs.begin();
00314 s != _partial_slabs.end(); ++s)
00315 count += s->num_free;
00316
00317 /* add empty slabs */
00318 count += _num_free * objects_per_slab;
00319
00320 return count;
00321 }
00322 };
00323
00333 template<typename Type, int Slab_size = L4_PAGESIZE,
00334 int Max_free = 2, template<typename A> class Alloc = New_allocator >
00335 class Slab : public Base_slab<sizeof(Type), Slab_size, Max_free, Alloc>
00336 {
00337 private:
00338 typedef Base_slab<sizeof(Type), Slab_size, Max_free, Alloc> Base_type;
00339 public:
00340
00341 typedef Type Obj_type;
00342
00343 Slab(typename Base_type::Slab_alloc const &alloc
00344 = typename Base_type::Slab_alloc()) noexcept
00345 : Base_slab<sizeof(Type), Slab_size, Max_free, Alloc>(alloc) {}
00346
00347
00355 Type *alloc() noexcept
00356 {
00357 return reinterpret_cast<Type *>(Base_type::alloc());
00358 }
00359
00366 void free(Type *o) noexcept

```

```

00367 { Base_slab<sizeof(Type), Slab_size, Max_free, Alloc>::free(o); }
00368 };
00369
00370
00386 template< int Obj_size, int Slab_size = L4_PAGESIZE,
00387 int Max_free = 2, template<typename A> class Alloc = New_allocator >
00388 class Base_slab_static
00389 {
00390 private:
00391 typedef Base_slab<Obj_size, Slab_size, Max_free, Alloc> _A;
00392 static _A _a;
00393 public:
00394 typedef void Obj_type;
00395 enum
00396 {
00398 object_size = Obj_size,
00400 slab_size = Slab_size,
00402 objects_per_slab = _A::objects_per_slab,
00404 max_free_slabs = Max_free,
00405 };
00406
00412 void *alloc() noexcept { return _a.alloc(); }
00413
00420 void free(void *p) noexcept { _a.free(p); }
00421
00430 unsigned total_objects() const noexcept { return _a.total_objects(); }
00431
00440 unsigned free_objects() const noexcept { return _a.free_objects(); }
00441 };
00442
00443
00444 template< int _O, int _S, int _M, template<typename A> class Alloc >
00445 typename Base_slab_static<_O,_S,_M,Alloc>::_A
00446 Base_slab_static<_O,_S,_M,Alloc>::_a;
00447
00463 template<typename Type, int Slab_size = L4_PAGESIZE,
00464 int Max_free = 2, template<typename A> class Alloc = New_allocator >
00465 class Slab_static
00466 : public Base_slab_static<sizeof(Type), Slab_size, Max_free, Alloc>
00467 {
00468 public:
00469
00470 typedef Type Obj_type;
00471 Type *alloc() noexcept
00472 {
00480 return reinterpret_cast<Type *>(
00481 Base_slab_static<sizeof(Type), Slab_size, Max_free, Alloc>::alloc());
00482 }
00483 };
00484
00485 }

```

## 17.212 slist

```

00001 // vi:set ft=c++: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2011 Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00009 #pragma once
00010
00011 #include "bits/list_basics.h"
00012
00013 namespace cxx {
00014
00015 class S_list_item
00016 {
00017 public:
00018 S_list_item() : _n(0) {}
00019 // BSS allocation
00020 explicit S_list_item(bool) {}
00021
00022 private:
00023 template<typename T, typename P> friend class S_list;
00024 template<typename T, typename P> friend class S_list_tail;
00025 template<typename T, typename X> friend struct Bits::Basic_list_policy;
00026
00027 S_list_item(S_list_item const &);
00028 void operator = (S_list_item const &);
00029

```

```

00030 S_list_item *_n;
00031 };
00032
00039 template< typename T, typename POLICY = Bits::Basic_list_policy< T, S_list_item > >
00040 class S_list : public Bits::Basic_list<POLICY>
00041 {
00042 S_list(S_list const &) = delete;
00043 void operator = (S_list const &) = delete;
00044
00045 private:
00046 typedef typename Bits::Basic_list<POLICY> Base;
00047
00048 public:
00049 typedef typename Base::Iterator Iterator;
00050
00051 S_list(S_list &&o) : Base(static_cast<Base&&>(o)) {}
00052
00053 S_list &operator = (S_list &&o)
00054 {
00055 Base::operator = (static_cast<Base&&>(o));
00056 return *this;
00057 }
00058
00059 // BSS allocation
00060 explicit S_list(bool x) : Base(x) {}
00061
00062 S_list() : Base() {}
00063
00065 void add(T *e)
00066 {
00067 e->_n = this->_f;
00068 this->_f = e;
00069 }
00070
00071 template< typename CAS >
00072 void add(T *e, CAS const &c)
00073 {
00074 do
00075 {
00076 e->_n = this->_f;
00077 }
00078 while (!c(&this->_f, e->_n, e));
00079 }
00080
00082 void push_front(T *e) { add(e); }
00083
00089 T *pop_front()
00090 {
00091 T *r = this->front();
00092 if (this->_f)
00093 this->_f = this->_f->_n;
00094 return r;
00095 }
00096
00097 void insert(T *e, Iterator const &pred)
00098 {
00099 S_list_item *p = *pred;
00100 e->_n = p->_n;
00101 p->_n = e;
00102 }
00103
00104 static void insert_before(T *e, Iterator const &succ)
00105 {
00106 S_list_item **x = Base::__get_internal(succ);
00107
00108 e->_n = *x;
00109 *x = e;
00110 }
00111
00112 static void replace(Iterator const &p, T*e)
00113 {
00114 S_list_item **x = Base::__get_internal(p);
00115 e->_n = (*x)->_n;
00116 *x = e;
00117 }
00118
00119 static Iterator erase(Iterator const &e)
00120 {
00121 S_list_item **x = Base::__get_internal(e);
00122 *x = (*x)->_n;
00123 return e;
00124 }
00125
00126 };
00127
00128
00129 template< typename T >

```

```

00130 class S_list_bss : public S_list<T>
00131 {
00132 public:
00133 S_list_bss() : S_list<T>(true) {}
00134 };
00135
00136 template< typename T, typename POLICY = Bits::Basic_list_policy< T, S_list_item > >
00137 class S_list_tail : public S_list<T, POLICY>
00138 {
00139 private:
00140 typedef S_list<T, POLICY> Base;
00141 void add(T *e) = delete;
00142
00143 public:
00144 using Iterator = typename Base::Iterator;
00145 S_list_tail() : Base(), _tail(&this->_f) {}
00146
00147 S_list_tail(S_list_tail &t)
00148 : Base(static_cast<Base&&>(t)), _tail(t.empty() ? &this->_f : t._tail)
00149 {
00150 t._tail = &t._f;
00151 }
00152
00153 S_list_tail &operator = (S_list_tail &t)
00154 {
00155 if (&t == this)
00156 return *this;
00157
00158 Base::operator = (static_cast<Base &&>(t));
00159 _tail = t.empty() ? &this->_f : t._tail;
00160 t._tail = &t._f;
00161 return *this;
00162 }
00163
00164 void push_front(T *e)
00165 {
00166 if (Base::empty())
00167 _tail = &e->_n;
00168
00169 Base::push_front(e);
00170 }
00171
00172 void push_back(T *e)
00173 {
00174 e->_n = 0;
00175 *_tail = e;
00176 _tail = &e->_n;
00177 }
00178
00179 void clear()
00180 {
00181 Base::clear();
00182 _tail = &this->_f;
00183 }
00184
00185 void append(S_list_tail &o)
00186 {
00187 T *x = o.front();
00188 *_tail = x;
00189 if (x)
00190 _tail = o._tail;
00191 o.clear();
00192 }
00193
00194 T *pop_front()
00195 {
00196 T *t = Base::pop_front();
00197 if (t && Base::empty())
00198 _tail = &this->_f;
00199 return t;
00200 }
00201
00202 private:
00203 static void insert(T *e, Iterator const &pred);
00204 static void insert_before(T *e, Iterator const &succ);
00205 static void replace(Iterator const &p, T *e);
00206 static Iterator erase(Iterator const &e);
00207
00208 private:
00209 S_list_item **_tail;
00210 };
00211
00212 }

```



## 17.213 static\_container

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * Copyright (C) 2012-2013 Technische Universität Dresden.
00004 * Copyright (C) 2016-2017, 2020, 2023-2024 Kernkonzept GmbH.
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00009 #pragma once
00010
00011 #include <l4/cxx/type_traits>
00012 #include <stddef.h>
00013
00014 namespace cxx {
00015
00016 template< typename T >
00017 class Static_container
00018 {
00019 private:
00020 struct X : T
00021 {
00022 void *operator new (size_t, void *p) noexcept { return p; }
00023 void operator delete (void *) {}
00024 X() = default;
00025 template<typename ...Args>
00026 X(Args && ...a) : T(cxx::forward<Args>(a)...) {}
00027 };
00028
00029 public:
00030 void operator = (Static_container const &) = delete;
00031 Static_container(Static_container const &) = delete;
00032 Static_container() = default;
00033
00034 T *get() { return reinterpret_cast<X*>(_s); }
00035 T *operator -> () { return get(); }
00036 T &operator * () { return *get(); }
00037 operator T* () { return get(); }
00038
00039 void construct()
00040 { new (reinterpret_cast<void*>(_s)) X; }
00041
00042 template< typename ...Args >
00043 void construct(Args && ...args)
00044 { new (reinterpret_cast<void*>(_s)) X(cxx::forward<Args>(args)...) };
00045
00046 private:
00047 char _s[sizeof(X)] __attribute__((aligned(__alignof(X)))));
00048 };
00049
00050 }
00051
00052

```

## 17.214 static\_vector

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002
00003 #pragma once
00004
00005 #include "type_traits"
00006
00007 namespace cxx {
00008
00015 template<typename T, typename IDX = unsigned>
00016 class static_vector
00017 {
00018 private:
00019 template<typename X, typename IDX2> friend class static_vector;
00020 T *_v;
00021 IDX _l;
00022
00023 public:
00024 typedef T value_type;
00025 typedef IDX index_type;
00026
00027 static_vector() = default;
00028 static_vector(value_type *v, index_type length) : _v(v), _l(length) {}
00029
00030 template<typename Z,
00031 typename = enable_if_t<is_same<remove_extent_t<Z>, T>::value>

```

```

00032 constexpr static_vector(Z &v) : _v(v), _l(array_size(v))
00033 {}
00034
00036 template<typename X,
00037 typename = enable_if_t<is_convertible<X, T>::value>
00038 static_vector(static_vector<X, IDX> const &o) : _v(o._v), _l(o._l) {}
00039
00040 index_type size() const { return _l; }
00041 bool empty() const { return _l == 0; }
00042
00043 value_type &operator [] (index_type idx) { return _v[idx]; }
00044 value_type const &operator [] (index_type idx) const { return _v[idx]; }
00045
00046 value_type *begin() { return _v; }
00047 value_type *end() { return _v + _l; }
00048 value_type const *begin() const { return _v; }
00049 value_type const *end() const { return _v + _l; }
00050 value_type const *cbegin() const { return _v; }
00051 value_type const *cend() const { return _v + _l; }
00052
00054 index_type index(value_type const *o) const { return o - _v; }
00055 index_type index(value_type const &o) const { return &o - _v; }
00056 };
00057
00058 }

```

## 17.215 std\_alloc

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009
00010 #pragma once
00011
00012 #include <stddef.h>
00013 namespace cxx {
00019 class Nothrow {};
00020 }
00021
00028 inline void *operator new (size_t, void *mem, cxx::Nothrow const &) noexcept
00029 { return mem; }
00030
00035 void *operator new (size_t, cxx::Nothrow const &) noexcept;
00036
00042 void operator delete (void *, cxx::Nothrow const &) noexcept;
00043
00044 namespace cxx {
00045
00046 template< typename _Type >
00056 class New_allocator
00057 {
00058 public:
00059 enum { can_free = true };
00060
00061 New_allocator() noexcept {}
00062 New_allocator(New_allocator const &) noexcept {}
00063
00064 ~New_allocator() noexcept {}
00065
00066 _Type *alloc() noexcept
00067 { return static_cast<_Type*> (::operator new(sizeof (_Type), cxx::Nothrow())); }
00068
00069 void free(_Type *t) noexcept
00070 { ::operator delete(t, cxx::Nothrow()); }
00071 };
00072
00073 }
00074

```

## 17.216 std\_ops

```

00001 // vim:set ft=cpp: -*- Mode: C++ -*-

```

```

00002 /*
00003 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>,
00004 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009
00010 #pragma once
00011
00012 namespace cxx {
00013
00017 template< typename Obj >
00018 struct Lt_functor
00019 {
00020 bool operator () (Obj const &l, Obj const &r) const
00021 { return l < r; }
00022 };
00023
00024 };
00025

```

## 17.217 string

```

00001 // vi:set ft=cpp: -- Mode: C++ --
00002 /*
00003 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00012 #pragma once
00013
00014 #include <l4/cxx/minmax>
00015 #include <l4/cxx/basic_ostream>
00016
00017 namespace cxx {
00018
00030 class String
00031 {
00032 public:
00033
00035 typedef char const *Index;
00036
00038 String(char const *s) noexcept : _start(s), _len(__builtin_strlen(s)) {}
00040 String(char const *s, unsigned long len) noexcept : _start(s), _len(len) {}
00041
00048 String(char const *s, char const *e) noexcept : _start(s), _len(e - s) {}
00049
00051 String() : _start(0), _len(0) {}
00052
00054 Index start() const { return _start; }
00056 Index end() const { return _start + _len; }
00058 int len() const { return _len; }
00059
00061 void start(char const *s) { _start = s; }
00063 void len(unsigned long len) { _len = len; }
00065 bool empty() const { return !_len; }
00066
00068 String head(Index end) const
00069 {
00070 if (end < _start)
00071 return String();
00072
00073 if (eof(end))
00074 return *this;
00075
00076 return String(_start, end - _start);
00077 }
00078
00080 String head(unsigned long end) const
00081 { return head(start() + end); }
00082
00084 String substr(unsigned long idx, unsigned long len = ~0UL) const
00085 {
00086 if (idx >= _len)
00087 return String(end(), 0UL);
00088
00089 return String(_start + idx, cxx::min(len, _len - idx));
00090 }
00091

```

```

00093 String substr(char const *start, unsigned long len = 0) const
00094 {
00095 if (start >= _start && !eof(start))
00096 {
00097 unsigned long nlen = _start + _len - start;
00098 if (len != 0)
00099 nlen = cxx::min(nlen, len);
00100 return String(start, nlen);
00101 }
00102 return String(end(), 0UL);
00103 }
00104
00105 template< typename F >
00107 char const *find_match(F &&match) const
00108 {
00109 String::Index s = _start;
00110 while (1)
00111 {
00112 if (eof(s))
00113 return s;
00114 if (match(*s))
00115 return s;
00116 ++s;
00117 }
00118 }
00119
00120 char const *find(char const *c) const
00121 { return find(c, start()); }
00122
00123 char const *find(int c) const
00124 { return find(c, start()); }
00125
00126 char const *rfind(char const *c) const
00127 {
00128 if (!_len)
00129 return end();
00130 char const *p = end();
00131 --p;
00132 while (p >= _start)
00133 {
00134 if (*p == *c)
00135 return p;
00136 --p;
00137 }
00138 return end();
00139 }
00140
00141 Index starts_with(cxx::String const &c) const
00142 {
00143 unsigned long i;
00144 for (i = 0; i < c._len && i < _len; ++i)
00145 if (_start[i] != c[i])
00146 return 0;
00147 return i == c._len ? start() + i : 0;
00148 }
00149
00150 char const *find(int c, char const *s) const
00151 {
00152 if (s < _start)
00153 return end();
00154 while (1)
00155 {
00156 if (eof(s))
00157 return s;
00158 if (*s == c)
00159 return s;
00160 ++s;
00161 }
00162 }
00163
00164 char const *find(char const *c, char const *s) const
00165 {
00166 if (s < _start)
00167 return end();
00168 while (1)
00169 {
00170 if (eof(s))
00171 return s;
00172 if (*s == *c)
00173 return s;
00174 ++s;
00175 }
00176 }
00177
00178 char const *find(char const *c, char const *s) const
00179 {
00180 if (s < _start)
00181 return end();
00182 while (1)
00183 {
00184 if (eof(s))
00185 return s;
00186 if (*s == *c)
00187 return s;
00188 ++s;
00189 }
00190 }

```

```

00200
00201 for (char const *x = c; *x; ++x)
00202 if (*s == *x)
00203 return s;
00204
00205 ++s;
00206 }
00207 }
00208
00210 char const &operator [] (unsigned long idx) const { return _start[idx]; }
00212 char const &operator [] (int idx) const { return _start[idx]; }
00214 char const &operator [] (Index idx) const { return *idx; }
00215
00217 bool eof(char const *s) const { return s >= _start + _len || !*s; }
00218
00227 template<typename INT>
00228 int from_dec(INT *v) const
00229 {
00230 *v = 0;
00231 Index c;
00232 for (c = start(); !eof(c); ++c)
00233 {
00234 unsigned char n;
00235 if (*c >= '0' && *c <= '9')
00236 n = *c - '0';
00237 else
00238 return c - start();
00239
00240 *v *= 10;
00241 *v += n;
00242 }
00243 return c - start();
00244 }
00245
00256 template<typename INT>
00257 int from_hex(INT *v) const
00258 {
00259 *v = 0;
00260 unsigned shift = 0;
00261 Index c;
00262 for (c = start(); !eof(c); ++c)
00263 {
00264 shift += 4;
00265 if (shift > sizeof(INT) * 8)
00266 return -1;
00267 unsigned char n;
00268 if (*c >= '0' && *c <= '9')
00269 n = *c - '0';
00270 else if (*c >= 'A' && *c <= 'F')
00271 n = *c - 'A' + 10;
00272 else if (*c >= 'a' && *c <= 'f')
00273 n = *c - 'a' + 10;
00274 else
00275 return c - start();
00276
00277 *v <<= 4;
00278 *v |= n;
00279 }
00280 return c - start();
00281 }
00282
00284 bool operator == (String const &o) const
00285 {
00286 if (len() != o.len())
00287 return false;
00288
00289 for (unsigned long i = 0; i < _len; ++i)
00290 if (_start[i] != o._start[i])
00291 return false;
00292
00293 return true;
00294 }
00295
00297 bool operator != (String const &o) const
00298 { return ! (operator == (o)); }
00299
00300 private:
00301 char const *_start;
00302 unsigned long _len;
00303 };
00304
00305 }
00306
00308 inline
00309 L4::BasicOStream &operator << (L4::BasicOStream &s, cxx::String const &str)
00310 {
00311 s.write(str.start(), str.len());

```

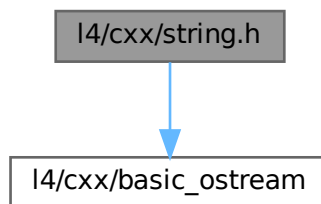
```
00312 return s;
00313 }
```

## 17.218 l4/cxx/string.h File Reference

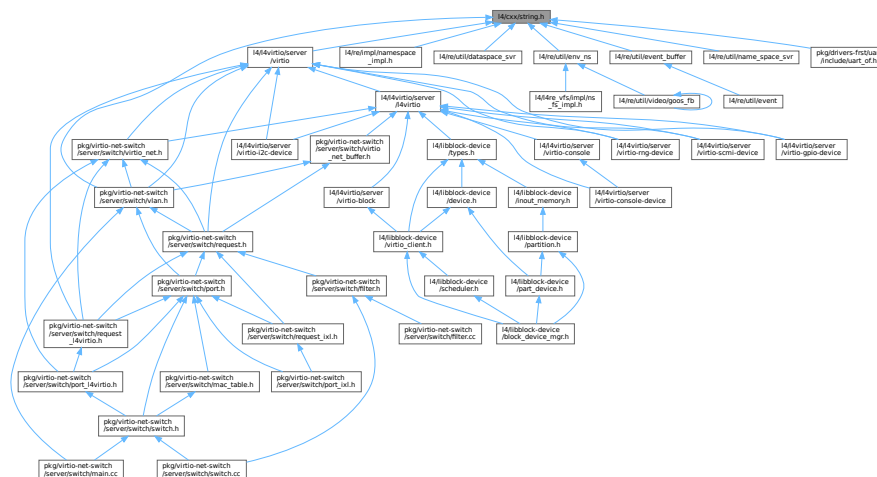
String.

```
#include <l4/cxx/basic_ostream>
```

Include dependency graph for string.h:



This graph shows which files directly or indirectly include this file:



### Data Structures

- class [L4::String](#)  
*A null-terminated string container class.*

### Namespaces

- namespace [L4](#)  
*L4 low-level kernel interface.*

## 17.218.1 Detailed Description

String.

Definition in file [string.h](#).

## 17.219 string.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2004-2009 Alexander Warg <warg@os.inf.tu-dresden.de>,
00007 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00014 #include <l4/cxx/basic_ostream>
00015
00016 namespace L4 {
00017
00022 class String
00023 {
00024 public:
00025 String(char const *str = "") : _str(str)
00026 {}
00027
00028 unsigned length() const
00029 {
00030 unsigned l;
00031 for (l = 0; _str[l]; l++)
00032 ;
00033 return l;
00034 }
00035
00036 char const *p_str() const { return _str; }
00037
00038 private:
00039 char const *_str;
00040 };
00041
00042 inline
00044 L4::BasicOStream &operator « (L4::BasicOStream &o, L4::String const &s)
00045 {
00046 o « s.p_str();
00047 return o;
00048 }

```

## 17.220 type\_list

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 #pragma once
00003
00004 /*
00005 * (c) 2012 Alexander Warg <warg@os.inf.tu-dresden.de>,
00006 * economic rights: Technische Universität Dresden (Germany)
00007 *
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010
00011 #include "type_traits"
00012
00014 namespace cxx {
00015
00016 template< typename ...T >
00017 struct type_list;
00018
00019 template<>
00020 struct type_list<>

```

```

00021 {
00022 typedef false_type head;
00023 typedef false_type tail;
00024 };
00025
00026 template<typename HEAD, typename ...TAIL>
00027 struct type_list<HEAD, TAIL...>
00028 {
00029 typedef HEAD head;
00030 typedef type_list<TAIL...> tail;
00031 };
00032
00033 template<typename TYPELIST, template <typename T> class PREDICATE>
00034 struct find_type;
00035
00036 template<template <typename T> class PREDICATE>
00037 struct find_type<type_list<>, PREDICATE>
00038 {
00039 typedef false_type type;
00040 };
00041
00042 template<typename TYPELIST, template <typename T> class PREDICATE>
00043 struct find_type
00044 {
00045 typedef typename conditional<PREDICATE<typename TYPELIST::head>::value,
00046 typename TYPELIST::head,
00047 typename find_type<typename TYPELIST::tail, PREDICATE>::type>::type
00048 type;
00049 };
00050
00051 template<typename TYPELIST, template <typename T> class PREDICATE>
00052 using find_type_t = typename find_type<TYPELIST, PREDICATE>::type;
00053 }
00054

```

## 17.221 type\_traits

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002
00003 /*
00004 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>,
00005 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00006 * economic rights: Technische Universität Dresden (Germany)
00007 *
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010
00011
00012 #pragma once
00013
00014 #pragma GCC system_header
00015
00016 #include <14/sys/compiler.h>
00017 #include "bits/type_traits.h"
00018
00019 namespace cxx {
00020
00021 template< typename T, T V >
00022 struct integral_constant
00023 {
00024 static T const value = V;
00025 typedef T value_type;
00026 typedef integral_constant<T, V> type;
00027 };
00028
00029 typedef integral_constant<bool, true> true_type;
00030 typedef integral_constant<bool, false> false_type;
00031
00032 template< typename T > struct remove_reference;
00033
00034 template< typename T > struct identity { typedef T type; };
00035 template< typename T > using identity_t = typename identity<T>::type;
00036
00037 template< typename T1, typename T2 > struct is_same;
00038
00039 template< typename T > struct remove_const;
00040
00041 template< typename T > struct remove_volatile;
00042
00043 template< typename T > struct remove_cv;
00044
00045 template< typename T > struct remove_pointer;

```



```

00046
00047 template< typename T > struct remove_extent;
00048
00049 template< typename T > struct remove_all_extents;
00050
00051
00052
00053 template< typename, typename >
00054 struct is_same : false_type {};
00055
00056 template< typename T >
00057 struct is_same<T, T> : true_type {};
00058
00059 template< typename T1, typename T2 >
00060 inline constexpr bool is_same_v = is_same<T1, T2>::value;
00061
00062 template< typename T >
00063 struct remove_reference { typedef T type; };
00064
00065 template< typename T >
00066 struct remove_reference<T &> { typedef T type; };
00067
00068 template< typename T >
00069 struct remove_reference<T &&> { typedef T type; };
00070
00071 template< typename T >
00072 using remove_reference_t = typename remove_reference<T>::type;
00073
00074 template< typename T > struct remove_const { typedef T type; };
00075 template< typename T > struct remove_const<T const> { typedef T type; };
00076 template< typename T > using remove_const_t = typename remove_const<T>::type;
00077
00078 template< typename T > struct remove_volatile { typedef T type; };
00079 template< typename T > struct remove_volatile<T volatile> { typedef T type; };
00080 template< typename T > using remove_volatile_t = typename remove_volatile<T>::type;
00081
00082 template< typename T >
00083 struct remove_cv { typedef remove_const_t<remove_volatile_t<T>> type; };
00084
00085 template< typename T >
00086 using remove_cv_t = typename remove_cv<T>::type;
00087
00088 template<class T>
00089 struct remove_cvref { using type = remove_cv_t<remove_reference_t<T>> };
00090
00091 template< typename T >
00092 using remove_cvref_t = typename remove_cvref<T>::type;
00093
00094 template< typename T, typename >
00095 struct __remove_pointer_h { typedef T type; };
00096
00097 template< typename T, typename I >
00098 struct __remove_pointer_h<T, I*> { typedef I type; };
00099
00100 template< typename T >
00101 struct remove_pointer : __remove_pointer_h<T, remove_cv_t<T>> {};
00102
00103 template< typename T >
00104 using remove_pointer_t = typename remove_pointer<T>::type;
00105
00106
00107 template< typename T >
00108 struct remove_extent { typedef T type; };
00109
00110 template< typename T >
00111 struct remove_extent<T[]> { typedef T type; };
00112
00113 template< typename T, unsigned long N >
00114 struct remove_extent<T[N]> { typedef T type; };
00115
00116 template< typename T >
00117 using remove_extent_t = typename remove_extent<T>::type;
00118
00119
00120 template< typename T >
00121 struct remove_all_extents { typedef T type; };
00122
00123 template< typename T >
00124 struct remove_all_extents<T[]> { typedef typename remove_all_extents<T>::type type; };
00125
00126 template< typename T, unsigned long N >
00127 struct remove_all_extents<T[N]> { typedef typename remove_all_extents<T>::type type; };
00128
00129 template< typename T >
00130 using remove_all_extents_t = typename remove_all_extents<T>::type;
00131
00132 template< typename T >

```

```

00133 constexpr T &&
00134 forward(cxx::remove_reference_t<T> &t)
00135 { return static_cast<T &&>(t); }
00136
00137 template< typename T >
00138 constexpr T &&
00139 forward(cxx::remove_reference_t<T> &&t)
00140 { return static_cast<T &&>(t); }
00141
00142 template< typename T >
00143 constexpr cxx::remove_reference_t<T> &&
00144 move(T &&t) { return static_cast< cxx::remove_reference_t<T> &&>(t); }
00145
00146 template< bool, typename T = void >
00147 struct enable_if {};
00148
00149 template< typename T >
00150 struct enable_if<true, T> { typedef T type; };
00151
00152 template< bool C, typename T = void >
00153 using enable_if_t = typename enable_if<C, T>::type;
00154
00155 template< typename T >
00156 struct is_const : false_type {};
00157
00158 template< typename T >
00159 struct is_const<T const> : true_type {};
00160
00161 template< typename T >
00162 inline constexpr bool is_const_v = is_const<T>::value;
00163
00164 template< typename T >
00165 struct is_volatile : false_type {};
00166
00167 template< typename T >
00168 struct is_volatile<T volatile> : true_type {};
00169
00170 template< typename T >
00171 inline constexpr bool is_volatile_v = is_volatile<T>::value;
00172
00173 template< typename T >
00174 struct is_pointer : false_type {};
00175
00176 template< typename T >
00177 struct is_pointer<T *> : true_type {};
00178
00179 template< typename T >
00180 inline constexpr bool is_pointer_v = is_pointer<T>::value;
00181
00182 template<class T>
00183 inline constexpr bool is_null_pointer_v = is_same_v<decltype(nullptr), remove_cv_t<T>>;
00184
00185 template< typename T >
00186 struct is_reference : false_type {};
00187
00188 template< typename T >
00189 struct is_reference<T &> : true_type {};
00190
00191 template< typename T >
00192 struct is_reference<T &&> : true_type {};
00193
00194 template< typename T >
00195 inline constexpr bool is_reference_v = is_reference<T>::value;
00196
00197 template< bool, typename, typename >
00198 struct conditional;
00199
00200 template< bool C, typename T_TRUE, typename T_FALSE >
00201 struct conditional { typedef T_TRUE type; };
00202
00203 template< typename T_TRUE, typename T_FALSE >
00204 struct conditional< false, T_TRUE, T_FALSE > { typedef T_FALSE type; };
00205
00206 template< bool C, typename T_TRUE, typename T_FALSE >
00207 using conditional_t = typename conditional<C, T_TRUE, T_FALSE>::type;
00208
00209 template<typename T>
00210 struct is_enum : integral_constant<bool, __is_enum(T)> {};
00211
00212 template< typename T >
00213 inline constexpr bool is_enum_v = is_enum<T>::value;
00214
00215 template<typename T>
00216 struct is_polymorphic : cxx::integral_constant<bool, __is_polymorphic(T)> {};
00217
00218 template< typename T > struct is_integral : false_type {};
00219

```

```

00220 template<> struct is_integral<bool> : true_type {};
00221
00222 template<> struct is_integral<char> : true_type {};
00223 template<> struct is_integral<signed char> : true_type {};
00224 template<> struct is_integral<unsigned char> : true_type {};
00225 template<> struct is_integral<short> : true_type {};
00226 template<> struct is_integral<unsigned short> : true_type {};
00227 template<> struct is_integral<int> : true_type {};
00228 template<> struct is_integral<unsigned int> : true_type {};
00229 template<> struct is_integral<long> : true_type {};
00230 template<> struct is_integral<unsigned long> : true_type {};
00231 template<> struct is_integral<long long> : true_type {};
00232 template<> struct is_integral<unsigned long long> : true_type {};
00233
00234 template< typename T >
00235 inline constexpr bool is_integral_v = is_integral<T>::value;
00236
00237 template< typename T, bool = is_integral_v<T> || is_enum_v<T> >
00238 struct __is_signed_helper : integral_constant<bool, static_cast<bool>(T(-1) < T(0))> {};;
00239
00240 template< typename T >
00241 struct __is_signed_helper<T, false> : integral_constant<bool, false> {};;
00242
00243 template< typename T >
00244 struct is_signed : __is_signed_helper<T> {};;
00245
00246 template< typename T >
00247 inline constexpr bool is_signed_v = is_signed<T>::value;
00248
00249
00250 template< typename >
00251 struct is_array : false_type {};;
00252
00253 template< typename T >
00254 struct is_array<T[]> : true_type {};;
00255
00256 template< typename T, unsigned long N >
00257 struct is_array<T[N]> : true_type {};;
00258
00259 template< typename T >
00260 inline constexpr bool is_array_v = is_array<T>::value;
00261
00262 template< typename T, unsigned N >
00263 constexpr unsigned array_size(T const (&)[N]) { return N; }
00264
00265 template< int SIZE, bool SIGN = false, bool = true > struct int_type_for_size;
00266
00267 template<> struct int_type_for_size<sizeof(char), true, true>
00268 { typedef signed char type; };
00269
00270 template<> struct int_type_for_size<sizeof(char), false, true>
00271 { typedef unsigned char type; };
00272
00273 template<> struct int_type_for_size<sizeof(short), true, (sizeof(short) > sizeof(char))>
00274 { typedef short type; };
00275
00276 template<> struct int_type_for_size<sizeof(short), false, (sizeof(short) > sizeof(char))>
00277 { typedef unsigned short type; };
00278
00279 template<> struct int_type_for_size<sizeof(int), true, (sizeof(int) > sizeof(short))>
00280 { typedef int type; };
00281
00282 template<> struct int_type_for_size<sizeof(int), false, (sizeof(int) > sizeof(short))>
00283 { typedef unsigned int type; };
00284
00285 template<> struct int_type_for_size<sizeof(long), true, (sizeof(long) > sizeof(int))>
00286 { typedef long type; };
00287
00288 template<> struct int_type_for_size<sizeof(long), false, (sizeof(long) > sizeof(int))>
00289 { typedef unsigned long type; };
00290
00291 template<> struct int_type_for_size<sizeof(long long), true, (sizeof(long long) > sizeof(long))>
00292 { typedef long long type; };
00293
00294 template<> struct int_type_for_size<sizeof(long long), false, (sizeof(long long) > sizeof(long))>
00295 { typedef unsigned long long type; };
00296
00297 template< int SIZE, bool SIGN = false>
00298 using int_type_for_size_t = typename int_type_for_size<SIZE, SIGN>::type;
00299
00300 template< typename T, class Enable = void > struct underlying_type {};;
00301
00302 template< typename T >
00303 struct underlying_type<T, typename enable_if<is_enum_v<T>::type >
00304 {
00305 typedef int_type_for_size_t<sizeof(T), is_signed_v<T> type;
00306 };

```

```

00307
00308 template< typename T >
00309 using underlying_type_t = typename underlying_type<T>::type;
00310
00311 template< typename T > struct make_signed;
00312 template<> struct make_signed<char> { typedef signed char type; };
00313 template<> struct make_signed<unsigned char> { typedef signed char type; };
00314 template<> struct make_signed<signed char> { typedef signed char type; };
00315 template<> struct make_signed<unsigned int> { typedef signed int type; };
00316 template<> struct make_signed<signed int> { typedef signed int type; };
00317 template<> struct make_signed<unsigned long int> { typedef signed long int type; };
00318 template<> struct make_signed<signed long int> { typedef signed long int type; };
00319 template<> struct make_signed<unsigned long long int> { typedef signed long long int type; };
00320 template<> struct make_signed<signed long long int> { typedef signed long long int type; };
00321 template< typename T > using make_signed_t = typename make_signed<T>::type;
00322
00323 template< typename T > struct make_unsigned;
00324 template<> struct make_unsigned<char> { typedef unsigned char type; };
00325 template<> struct make_unsigned<unsigned char> { typedef unsigned char type; };
00326 template<> struct make_unsigned<signed char> { typedef unsigned char type; };
00327 template<> struct make_unsigned<unsigned int> { typedef unsigned int type; };
00328 template<> struct make_unsigned<signed int> { typedef unsigned int type; };
00329 template<> struct make_unsigned<unsigned long int> { typedef unsigned long int type; };
00330 template<> struct make_unsigned<signed long int> { typedef unsigned long int type; };
00331 template<> struct make_unsigned<unsigned long long int> { typedef unsigned long long int type; };
00332 template<> struct make_unsigned<signed long long int> { typedef unsigned long long int type; };
00333 template< typename T > using make_unsigned_t = typename make_unsigned<T>::type;
00334
00335
00336 template<typename From, typename To>
00337 struct is_convertible
00338 {
00339 private:
00340 struct _true { char x[2]; };
00341 struct _false {};
00342
00343 static _true _helper(To const *);
00344 static _false _helper(...);
00345 public:
00346 enum
00347 {
00348 value = sizeof(_true) == sizeof(_helper(static_cast<From*>(0)))
00349 ? true : false
00350 };
00351
00352 typedef bool value_type;
00353 };
00354
00355 template<typename From, typename To>
00356 inline constexpr bool is_convertible_v = is_convertible<From, To>::value;
00357
00358 template< typename T >
00359 struct is_empty : integral_constant<bool, __is_empty(T)> {};
00360
00361 template< typename T >
00362 inline constexpr bool is_empty_v = is_empty<T>::value;
00363
00364
00365 #if L4_HAS_BUILTIN(__is_function)
00366 template < typename T >
00367 struct is_function : integral_constant<bool, __is_function(T)> {};
00368 #else
00369 template < typename T >
00370 struct is_function : integral_constant<bool, !is_reference_v<T>
00371 && !is_const_v<const T> > {};
00372 #endif
00373
00374 template< typename T >
00375 inline constexpr bool is_function_v = is_function<T>::value;
00376
00377 }
00378

```

## 17.222 unique\_ptr

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * Copyright (C) 2013 Technische Universität Dresden.
00004 * Copyright (C) 2014-2017, 2020, 2023-2024 Kernkonzept GmbH.
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008

```

```

00009 #pragma once
00010
00011 #include "type_traits"
00012
00013 namespace cxx
00014 {
00015
00016 template< typename T >
00017 struct default_delete
00018 {
00019 default_delete() {}
00020
00021 template< typename U >
00022 default_delete(default_delete<U> const &) {}
00023
00024 void operator () (T *p) const
00025 { delete p; }
00026 };
00027
00028 template< typename T >
00029 struct default_delete<T[]>
00030 {
00031 default_delete() {}
00032
00033 void operator () (T *p)
00034 { delete [] p; }
00035 };
00036
00037 template< typename T, typename C >
00038 struct unique_ptr_index_op {};
00039
00040 template< typename T, typename C >
00041 struct unique_ptr_index_op<T[], C>
00042 {
00043 typedef T &reference;
00044 reference operator [] (int idx) const
00045 { return static_cast<C const *>(this)->get()[idx]; }
00046 };
00047
00048 template< typename T, typename T_Del = default_delete<T> >
00049 class unique_ptr : public unique_ptr_index_op<T, unique_ptr<T, T_Del> >
00050 {
00051 private:
00052 struct _unspec;
00053 typedef _unspec* _unspec_ptr_type;
00054
00055 public:
00056 typedef cxx::remove_extent_t<T> element_type;
00057 typedef element_type *pointer;
00058 typedef element_type &reference;
00059 typedef T_Del deleter_type;
00060
00061 unique_ptr() : _ptr(pointer()) {}
00062
00063 explicit unique_ptr(pointer p) : _ptr(p) {}
00064
00065 unique_ptr(unique_ptr &&o) : _ptr(o.release()) {}
00066
00067 ~unique_ptr() { reset(); }
00068
00069 unique_ptr &operator = (unique_ptr &&o)
00070 {
00071 reset(o.release());
00072 return *this;
00073 }
00074
00075 unique_ptr &operator = (_unspec_ptr_type)
00076 {
00077 reset();
00078 return *this;
00079 }
00080
00081 element_type &operator * () const { return *get(); }
00082 pointer operator -> () const { return get(); }
00083
00084 pointer get() const { return _ptr; }
00085
00086 operator _unspec_ptr_type () const
00087 { return reinterpret_cast<_unspec_ptr_type>(get()); }
00088
00089 pointer release()
00090 {
00091 pointer r = _ptr;
00092 _ptr = 0;
00093 return r;
00094 }
00095

```

```

00096 void reset(pointer p = pointer())
00097 {
00098 if (p != get())
00099 {
00100 deleter_type()(get());
00101 _ptr = p;
00102 }
00103 }
00104
00105 unique_ptr(unique_ptr const &) = delete;
00106 unique_ptr &operator = (unique_ptr const &) = delete;
00107
00108 private:
00109 pointer _ptr;
00110 };
00111
00112 template< typename T >
00113 unique_ptr<T>
00114 make_unique_ptr(T *p)
00115 { return unique_ptr<T>(p); }
00116
00117 template< typename T >
00118 cxx::enable_if_t<cxx::is_array<T>::value, unique_ptr<T>»
00119 make_unique(unsigned long size)
00120 { return cxx::unique_ptr<T>(new cxx::remove_extent_t<T>[size]()); }
00121
00122 template< typename T, typename... Args >
00123 cxx::enable_if_t<!cxx::is_array<T>::value, unique_ptr<T>»
00124 make_unique(Args &&... args)
00125 { return cxx::unique_ptr<T>(new T(cxx::forward<Args>(args)...)); }
00126
00127 }

```

## 17.223 l4/cxx/unique\_ptr\_list File Reference

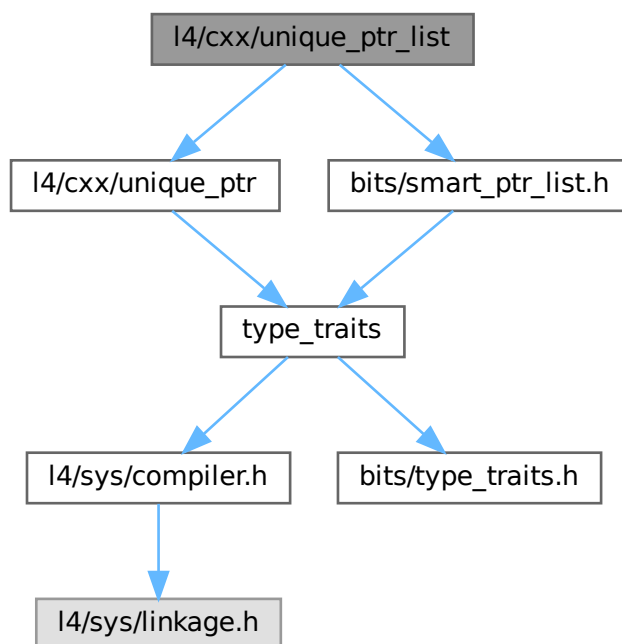
Implementation of a list of unique-ptr-managed objects.

```

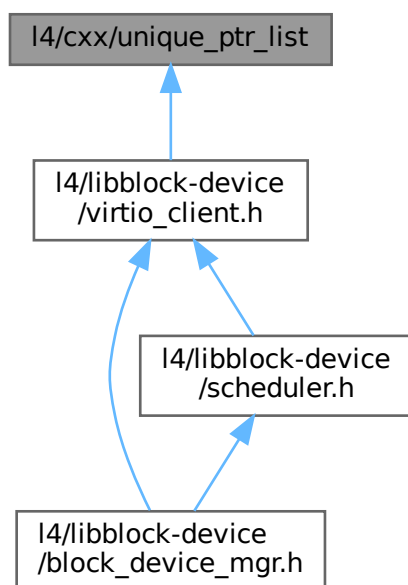
#include <l4/cxx/unique_ptr>
#include "bits/smart_ptr_list.h"

```

Include dependency graph for unique\_ptr\_list:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace `cxx`  
*Our C++ library.*

## Typedefs

- `template<typename T>`  
`using cxx::Unique_ptr_list_item = Bits::Smart_ptr_list_item<T, cxx::unique_ptr<T> >`  
*Item for list linked with `cxx::unique_ptr`.*
- `template<typename T>`  
`using cxx::Unique_ptr_list = Bits::Smart_ptr_list<Unique_ptr_list_item<T> >`  
*Single-linked list where elements are connected with a `cxx::unique_ptr`.*

### 17.223.1 Detailed Description

Implementation of a list of unique-ptr-managed objects.

Definition in file `unique_ptr_list`.

## 17.224 `unique_ptr_list`

[Go to the documentation of this file.](#)

```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * Copyright (C) 2018–2019, 2022, 2024 Kernkonzept GmbH.
00008 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00014 #include <l4/cxx/unique_ptr>
00015
00016 #include "bits/smart_ptr_list.h"
00017
00018 namespace cxx {
00019
00021 template <typename T>
00022 using Unique_ptr_list_item = Bits::Smart_ptr_list_item<T, cxx::unique_ptr<T> >;
00023
00027 template <typename T>
00028 using Unique_ptr_list = Bits::Smart_ptr_list<Unique_ptr_list_item<T> >;
00029
00030 }
```

## 17.225 `utils`

```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * Copyright (C) 2013 Technische Universität Dresden.
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #pragma once
00009
00010 namespace cxx {
00011
00039 template< typename T > inline
00040 T access_once(T const *a)
```



```

00041 {
00042 #if 1
00043 __asm__ __volatile__ ("" : "=m"(*const_cast<T*>(a));
00044 T tmp = *a;
00045 __asm__ __volatile__ ("" : "=m"(*const_cast<T*>(a));
00046 return tmp;
00047 #else
00048 return *static_cast<T const volatile *>(a);
00049 #endif
00050 }
00051
00070 template< typename T, typename VAL > inline
00071 void write_now(T *a, VAL &&val)
00072 {
00073 __asm__ __volatile__ ("" : "=m"(*a));
00074 *a = val;
00075 __asm__ __volatile__ ("" : : "m"(*a));
00076 }
00077
00078
00079 }
00080

```

## 17.226 weak\_ref

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * Copyright (C) 2015, 2017, 2024 Kernkonzept GmbH.
00004 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00005 * Alexander Warg <alexander.warg@kernkonzept.com>
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 #pragma once
00010
00011 #include "hlist"
00012
00013 namespace cxx {
00014
00024 class Weak_ref_base : public H_list_item_t<Weak_ref_base>
00025 {
00026 protected:
00027 Weak_ref_base(void const *ptr = nullptr) : _obj(ptr) {}
00028 void reset_hard() { _obj = nullptr; }
00029 void const *_obj;
00030
00031 public:
00038 struct List : H_list_t<Weak_ref_base>
00039 {
00040 void reset()
00041 {
00042 while (!empty())
00043 pop_front()->reset_hard();
00044 }
00045
00046 ~List()
00047 { reset(); }
00048 };
00049
00050 explicit operator bool () const
00051 { return _obj ? true : false; }
00052 };
00053
00054
00094 template <typename T>
00095 class Weak_ref : public Weak_ref_base
00096 {
00097 public:
00098 T *get() const
00099 { return reinterpret_cast<T*>(const_cast<void *>(_obj)); }
00100
00101 T *reset(T *n)
00102 {
00103 T *r = get();
00104 if (r)
00105 r->remove_weak_ref(this);
00106
00107 _obj = n;
00108 if (n)
00109 n->add_weak_ref(this);
00110
00111 return r;
00112 }

```

```

00113
00114 Weak_ref(T *s = nullptr) : Weak_ref_base(s)
00115 {
00116 if (s)
00117 s->add_weak_ref(this);
00118 }
00119
00120 ~Weak_ref() { reset(0); }
00121
00122 void operator = (T *n)
00123 { reset(n); }
00124
00125 Weak_ref(Weak_ref const &o) : Weak_ref_base(o._obj)
00126 {
00127 if (T *x = get())
00128 x->add_weak_ref(this);
00129 }
00130
00131 Weak_ref &operator = (Weak_ref const &o)
00132 {
00133 if (&o == this)
00134 return *this;
00135
00136 reset(o.get());
00137 return *this;
00138 }
00139
00140 T &operator * () const { return get(); }
00141 T *operator -> () const { return get(); }
00142 };
00143
00144 class Weak_ref_obj
00145 {
00146 protected:
00147 template <typename T> friend class Weak_ref;
00148 mutable Weak_ref_base::List weak_references;
00149
00150 void add_weak_ref(Weak_ref_base *ref) const
00151 { weak_references.push_front(ref); }
00152
00153 void remove_weak_ref(Weak_ref_base *ref) const
00154 { weak_references.remove(ref); }
00155 };
00156
00157 }

```

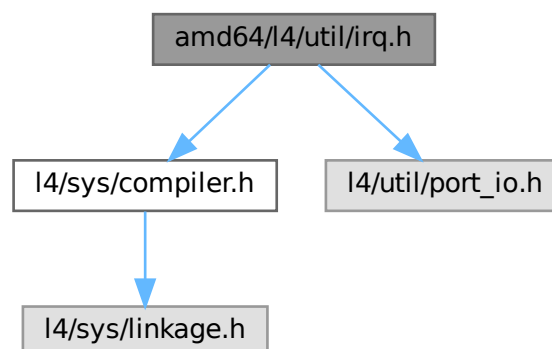
## 17.227 amd64/l4/util/irq.h File Reference

some PIC and hardware interrupt related functions

```
#include <l4/sys/compiler.h>
```

```
#include <l4/util/port_io.h>
```

Include dependency graph for irq.h:



## 17.227.1 Detailed Description

some PIC and hardware interrupt related functions

### Date

2003

### Author

Jork Loeser [jork.loeser@inf.tu-dresden.de](mailto:jork.loeser@inf.tu-dresden.de) Frank Mehnert [fm3@os.inf.tu-dresden.de](mailto:fm3@os.inf.tu-dresden.de)

Definition in file [irq.h](#).

## 17.228 irq.h

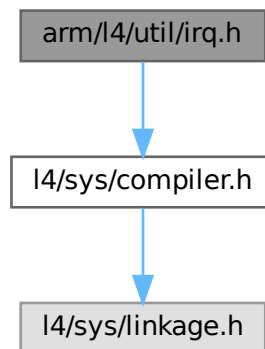
[Go to the documentation of this file.](#)

```
00001
00008
00009 /*
00010 * (c) 2003-2009 Author(s)
00011 * economic rights: Technische Universität Dresden (Germany)
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014
00015 #ifndef __L4_IRQ_H__
00016 #define __L4_IRQ_H__
00017
00018 #include <l4/sys/compiler.h>
00019 #include <l4/util/port_io.h>
00020
00021 L4_BEGIN_DECLS
00022
00027
00030 static inline void
00031 l4util_cli (void)
00032 {
00033 __asm__ __volatile__ ("cli" : : : "memory");
00034 }
00035
00038 static inline void
00039 l4util_sti (void)
00040 {
00041 __asm__ __volatile__ ("sti" : : : "memory");
00042 }
00043
00047 static inline void
00048 l4util_flags_save (l4_umword_t *flags)
00049 {
00050 __asm__ __volatile__ ("pushf ; popq %0" : "=g" (*flags) : : "memory");
00051 }
00052
00055 static inline void
00056 l4util_flags_restore (l4_umword_t *flags)
00057 {
00058 __asm__ __volatile__ ("pushq %0 ; popf" : : "g" (*flags) : "memory");
00059 }
00061
00062 L4_END_DECLS
00063
00064 #endif
```

## 17.229 arm/l4/util/irq.h File Reference

ARM specific implementation of irq functions.

#include <l4/sys/compiler.h>  
Include dependency graph for irq.h:



### 17.229.1 Detailed Description

ARM specific implementation of irq functions.

Do not use.

Definition in file [irq.h](#).

## 17.230 irq.h

[Go to the documentation of this file.](#)

```

00001
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00005 * Frank Mehnert <fm3@os.inf.tu-dresden.de>
00006 * economic rights: Technische Universität Dresden (Germany)
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 #ifndef __L4UTIL__ARCH_ARCH__IRQ_H__
00010 #define __L4UTIL__ARCH_ARCH__IRQ_H__
00011
00012 #ifdef __GNUC__
00013 #include <l4/sys/compiler.h>
00014
00015 L4_BEGIN_DECLS
00016
00017 L4_INLINE void l4util_cli (void);
00018 L4_INLINE void l4util_sti (void);
00019 L4_INLINE void l4util_flags_save(l4_umword_t *flags);
00020 L4_INLINE void l4util_flags_restore(l4_umword_t *flags);
00021
00022 L4_INLINE

```

```

00029 void
00030 l4util_cli(void)
00031 {
00032 extern void __do_not_use_l4util_cli(void);
00033 __do_not_use_l4util_cli();
00034 }
00035
00036
00037 L4_INLINE
00038 void
00039 l4util_sti(void)
00040 {
00041 extern void __do_not_use_l4util_sti(void);
00042 __do_not_use_l4util_sti();
00043 }
00044
00045
00046 L4_INLINE
00047 void
00048 l4util_flags_save(l4_umword_t *flags)
00049 {
00050 (void) flags;
00051 extern void __do_not_use_l4util_flags_save(void);
00052 __do_not_use_l4util_flags_save();
00053 }
00054
00055 L4_INLINE
00056 void
00057 l4util_flags_restore(l4_umword_t *flags)
00058 {
00059 (void) flags;
00060 extern void __do_not_use_l4util_flags_restore(void);
00061 __do_not_use_l4util_flags_restore();
00062 }
00063
00064 L4_END_DECLS
00065
00066 #endif // __GNUC__
00067
00068 #endif /* ! __L4UTIL__ARCH_ARCH__IRQ_H__ */

```

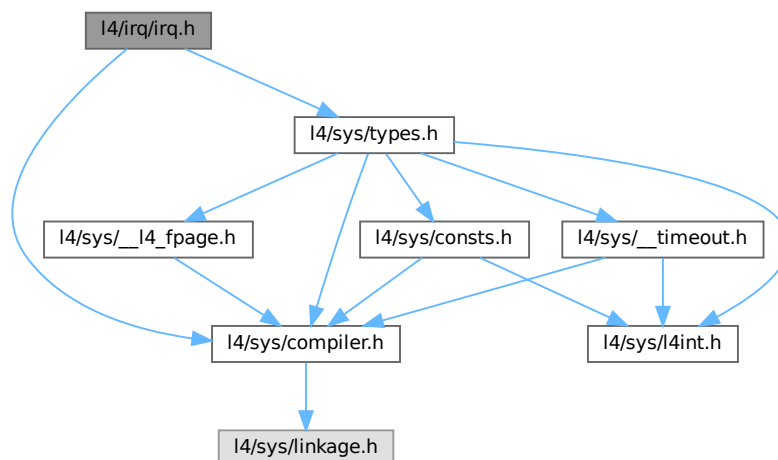
## 17.231 l4/irq/irq.h File Reference

IRQ handling routines.

```
#include <l4/sys/compiler.h>
```

```
#include <l4/sys/types.h>
```

Include dependency graph for irq.h:



## Functions

- `l4irq_t * l4irq_attach` (int irqnum)  
*Attach/connect to IRQ.*
- `l4irq_t * l4irq_attach_ft` (int irqnum, unsigned mode)  
*Attach/connect to IRQ using given type.*
- `l4irq_t * l4irq_attach_thread` (int irqnum, `l4_cap_idx_t` to\_thread)  
*Attach/connect to IRQ.*
- `l4irq_t * l4irq_attach_thread_ft` (int irqnum, `l4_cap_idx_t` to\_thread, unsigned mode)  
*Attach/connect to IRQ using given type.*
- `long l4irq_wait` (`l4irq_t` \*irq)  
*Wait for specified IRQ.*
- `long l4irq_unmask_and_wait_any` (`l4irq_t` \*unmask\_irq, `l4irq_t` \*\*ret\_irq)  
*Unmask a specific IRQ and wait for any attached IRQ.*
- `long l4irq_wait_any` (`l4irq_t` \*\*irq)  
*Wait for any attached IRQ.*
- `long l4irq_unmask` (`l4irq_t` \*irq)  
*Unmask a specific IRQ.*
- `long l4irq_detach` (`l4irq_t` \*irq)  
*Detach from IRQ.*
- `l4irq_t * l4irq_request` (int irqnum, void(\*isr\_handler)(void \*), void \*isr\_data, int irq\_thread\_prio, unsigned mode)  
*Attach asynchronous ISR handler to IRQ.*
- `long l4irq_release` (`l4irq_t` \*irq)  
*Release asynchronous ISR handler and free resources.*
- `l4irq_t * l4irq_attach_cap` (`l4_cap_idx_t` irqcap)  
*Attach/connect to IRQ.*
- `l4irq_t * l4irq_attach_cap_ft` (`l4_cap_idx_t` irqcap, unsigned mode)  
*Attach/connect to IRQ using given type.*
- `l4irq_t * l4irq_attach_thread_cap` (`l4_cap_idx_t` irqcap, `l4_cap_idx_t` to\_thread)  
*Attach/connect to IRQ.*
- `l4irq_t * l4irq_attach_thread_cap_ft` (`l4_cap_idx_t` irqcap, `l4_cap_idx_t` to\_thread, unsigned mode)  
*Attach/connect to IRQ using given type.*
- `l4irq_t * l4irq_request_cap` (`l4_cap_idx_t` irqcap, void(\*isr\_handler)(void \*), void \*isr\_data, int irq\_thread\_prio, unsigned mode)  
*Attach asynchronous ISR handler to IRQ.*

### 17.231.1 Detailed Description

IRQ handling routines.

Definition in file [irq.h](#).

## 17.232 irq.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Henning Schild <hschild@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * This file is part of TUD:OS and distributed under the terms of the
00011 * GNU General Public License 2.
00012 * Please see the COPYING-GPL-2 file for details.
00013 */
00014 #pragma once
00015
00016 #include <l4/sys/compiler.h>
00017 #include <l4/sys/types.h>
00018
00019 __BEGIN_DECLS
00020
00027
00028 struct l4irq_t;
00029 typedef struct l4irq_t l4irq_t;
00030
00041 L4_CV l4irq_t *
00042 l4irq_attach(int irqnum);
00043
00055 L4_CV l4irq_t *
00056 l4irq_attach_ft(int irqnum, unsigned mode);
00057
00068 L4_CV l4irq_t *
00069 l4irq_attach_thread(int irqnum, l4_cap_idx_t to_thread);
00070
00082 L4_CV l4irq_t *
00083 l4irq_attach_thread_ft(int irqnum, l4_cap_idx_t to_thread,
00084 unsigned mode);
00085
00093 L4_CV long
00094 l4irq_wait(l4irq_t *irq);
00095
00105 L4_CV long
00106 l4irq_unmask_and_wait_any(l4irq_t *unmask_irq, l4irq_t **ret_irq);
00107
00115 L4_CV long
00116 l4irq_wait_any(l4irq_t **irq);
00117
00128 L4_CV long
00129 l4irq_unmask(l4irq_t *irq);
00130
00138 L4_CV long
00139 l4irq_detach(l4irq_t *irq);
00140
00141
00142
00143 /*****
00152
00165 L4_CV l4irq_t *
00166 l4irq_request(int irqnum, void (*isr_handler)(void *), void *isr_data,
00167 int irq_thread_prio, unsigned mode);
00168
00176 L4_CV long
00177 l4irq_release(l4irq_t *irq);
00178
00179
00180
00181 /*****
00182 /*****
00183
00188
00199 L4_CV l4irq_t *
00200 l4irq_attach_cap(l4_cap_idx_t irqcap);
00201
00213 L4_CV l4irq_t *
00214 l4irq_attach_cap_ft(l4_cap_idx_t irqcap, unsigned mode);
00215
00226 L4_CV l4irq_t *
00227 l4irq_attach_thread_cap(l4_cap_idx_t irqcap, l4_cap_idx_t to_thread);
00228
00240 L4_CV l4irq_t *
00241 l4irq_attach_thread_cap_ft(l4_cap_idx_t irqcap, l4_cap_idx_t to_thread,
00242 unsigned mode);
00243
00244 /*****
00252
00265 L4_CV l4irq_t *

```

```

00266 l4irq_request_cap(l4_cap_idx_t irqcap,
00267 void (*isr_handler)(void *), void *isr_data,
00268 int irq_thread_prio, unsigned mode);
00269
00270 __END_DECLS

```

## 17.233 l4/sys/irq.h File Reference

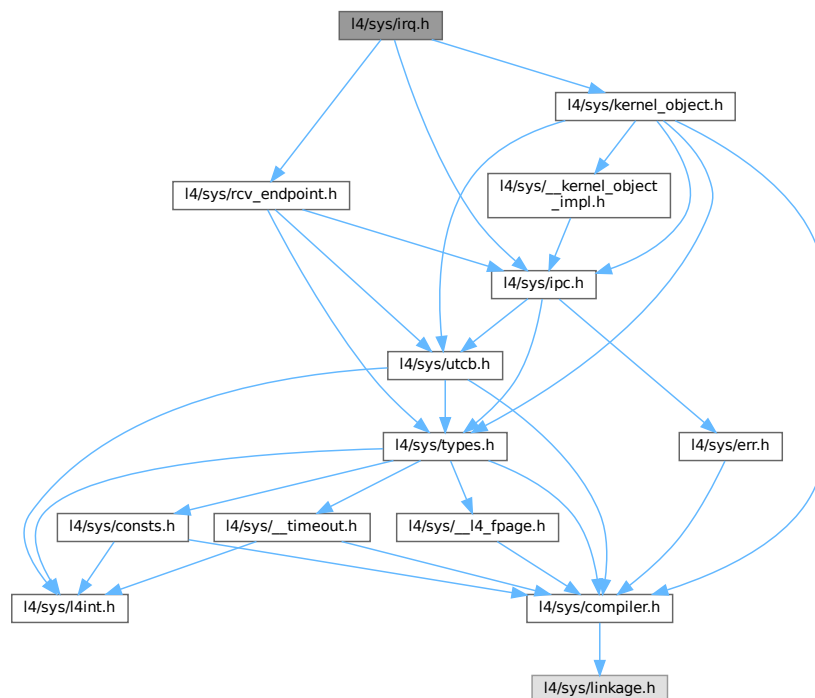
C Irq interface.

```

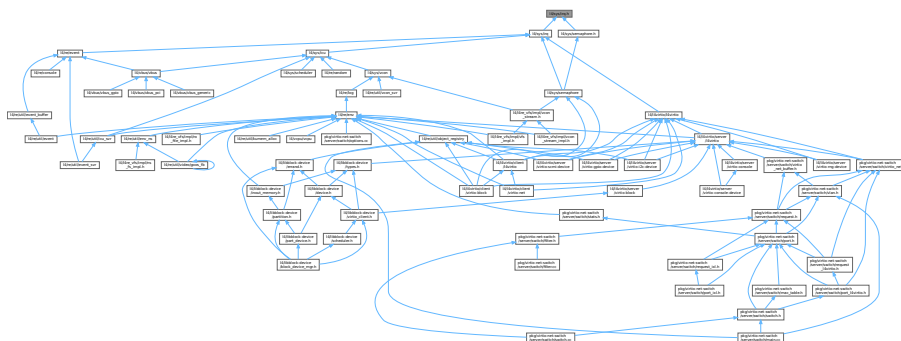
#include <l4/sys/kernel_object.h>
#include <l4/sys/ipc.h>
#include <l4/sys/rcv_endpoint.h>

```

Include dependency graph for irq.h:



This graph shows which files directly or indirectly include this file:





## Functions

- [l4\\_msgtag\\_t l4\\_irq\\_detach \(l4\\_cap\\_idx\\_t irq\) L4\\_NOTHROW](#)  
*Detach from an interrupt source.*
- [l4\\_msgtag\\_t l4\\_irq\\_detach\\_u \(l4\\_cap\\_idx\\_t irq, l4\\_utcb\\_t \\*utcb\) L4\\_NOTHROW](#)  
*Detach from this interrupt.*
- [l4\\_msgtag\\_t l4\\_irq\\_bind\\_vcpu \(l4\\_cap\\_idx\\_t irq, l4\\_cap\\_idx\\_t thread, l4\\_umword\\_t cfg\) L4\\_NOTHROW](#)  
*Bind a thread to this Irq for vCPU interrupt forwarding.*
- [l4\\_msgtag\\_t l4\\_irq\\_bind\\_vcpu\\_u \(l4\\_cap\\_idx\\_t irq, l4\\_cap\\_idx\\_t thread, l4\\_umword\\_t cfg, l4\\_utcb\\_t \\*utcb\) L4\\_NOTHROW](#)  
*Bind a thread to this Irq for vCPU interrupt forwarding.*
- [l4\\_msgtag\\_t l4\\_irq\\_trigger \(l4\\_cap\\_idx\\_t irq\) L4\\_NOTHROW](#)  
*Trigger an IRQ.*
- [l4\\_msgtag\\_t l4\\_irq\\_trigger\\_u \(l4\\_cap\\_idx\\_t irq, l4\\_utcb\\_t \\*utcb\) L4\\_NOTHROW](#)  
*Trigger the object.*
- [l4\\_msgtag\\_t l4\\_irq\\_receive \(l4\\_cap\\_idx\\_t irq, l4\\_timeout\\_t to\) L4\\_NOTHROW](#)  
*Unmask and wait for specified IRQ.*
- [l4\\_msgtag\\_t l4\\_irq\\_receive\\_u \(l4\\_cap\\_idx\\_t irq, l4\\_timeout\\_t timeout, l4\\_utcb\\_t \\*utcb\) L4\\_NOTHROW](#)  
*Unmask and wait for this IRQ.*
- [l4\\_msgtag\\_t l4\\_irq\\_wait \(l4\\_cap\\_idx\\_t irq, l4\\_umword\\_t \\*label, l4\\_timeout\\_t to\) L4\\_NOTHROW](#)  
*Unmask IRQ and wait for any message.*
- [l4\\_msgtag\\_t l4\\_irq\\_wait\\_u \(l4\\_cap\\_idx\\_t irq, l4\\_umword\\_t \\*label, l4\\_timeout\\_t timeout, l4\\_utcb\\_t \\*utcb\) L4\\_NOTHROW](#)  
*Unmask IRQ and (open) wait for any message.*
- [l4\\_msgtag\\_t l4\\_irq\\_unmask \(l4\\_cap\\_idx\\_t irq\) L4\\_NOTHROW](#)  
*Unmask IRQ.*
- [l4\\_msgtag\\_t l4\\_irq\\_unmask\\_u \(l4\\_cap\\_idx\\_t irq, l4\\_utcb\\_t \\*utcb\) L4\\_NOTHROW](#)  
*Unmask this IRQ.*

### 17.233.1 Detailed Description

C Irq interface.

Definition in file [irq.h](#).

## 17.234 irq.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00009 * Björn Döbel <doebel@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #pragma once
00015
00016 #include <l4/sys/kernel_object.h>
00017 #include <l4/sys/ipc.h>
00018 #include <l4/sys/rcv_endpoint.h>
00019
00046
00062 L4_INLINE l4_msgtag_t
00063 l4_irq_detach(l4_cap_idx_t irq) L4_NOTHROW;
```

```

00064
00071 L4_INLINE l4_msgtag_t
00072 l4_irq_detach_u(l4_cap_idx_t irq, l4_utcb_t *utcb) L4_NOTHROW;
00073
00074
00109 L4_INLINE l4_msgtag_t
00110 l4_irq_bind_vcpu(l4_cap_idx_t irq, l4_cap_idx_t thread,
00111 l4_umword_t cfg) L4_NOTHROW;
00112
00119 L4_INLINE l4_msgtag_t
00120 l4_irq_bind_vcpu_u(l4_cap_idx_t irq, l4_cap_idx_t thread, l4_umword_t cfg,
00121 l4_utcb_t *utcb) L4_NOTHROW;
00122
00123
00139 L4_INLINE l4_msgtag_t
00140 l4_irq_trigger(l4_cap_idx_t irq) L4_NOTHROW;
00141
00148 L4_INLINE l4_msgtag_t
00149 l4_irq_trigger_u(l4_cap_idx_t irq, l4_utcb_t *utcb) L4_NOTHROW;
00150
00160 L4_INLINE l4_msgtag_t
00161 l4_irq_receive(l4_cap_idx_t irq, l4_timeout_t to) L4_NOTHROW;
00162
00169 L4_INLINE l4_msgtag_t
00170 l4_irq_receive_u(l4_cap_idx_t irq, l4_timeout_t timeout, l4_utcb_t *utcb) L4_NOTHROW;
00171
00182 L4_INLINE l4_msgtag_t
00183 l4_irq_wait(l4_cap_idx_t irq, l4_umword_t *label,
00184 l4_timeout_t to) L4_NOTHROW;
00185
00192 L4_INLINE l4_msgtag_t
00193 l4_irq_wait_u(l4_cap_idx_t irq, l4_umword_t *label,
00194 l4_timeout_t timeout, l4_utcb_t *utcb) L4_NOTHROW;
00195
00206 L4_INLINE l4_msgtag_t
00207 l4_irq_unmask(l4_cap_idx_t irq) L4_NOTHROW;
00208
00215 L4_INLINE l4_msgtag_t
00216 l4_irq_unmask_u(l4_cap_idx_t irq, l4_utcb_t *utcb) L4_NOTHROW;
00217
00221 enum L4_irq_sender_op
00222 {
00223 L4_IRQ_SENDER_OP_RESERVED1 = 0, // Ex ATTACH
00224 L4_IRQ_SENDER_OP_DETACH = 1,
00225 L4_IRQ_SENDER_OP_BIND_VCPU = 2,
00226 };
00227
00231 enum L4_irq_op
00232 {
00233 L4_IRQ_OP_TRIGGER = 2,
00234 L4_IRQ_OP_EOI = 4
00235 };
00236
00237 /*****
00238 * Implementations
00239 */
00240
00241 L4_INLINE l4_msgtag_t
00242 l4_irq_detach_u(l4_cap_idx_t irq, l4_utcb_t *utcb) L4_NOTHROW
00243 {
00244 l4_utcb_mr_u(utcb)->mr[0] = L4_IRQ_SENDER_OP_DETACH;
00245 return l4_ipc_call(irq, utcb, l4_msgtag(L4_PROTO_IRQ_SENDER, 1, 0, 0),
00246 L4_IPC_NEVER);
00247 }
00248
00249 L4_INLINE l4_msgtag_t
00250 l4_irq_bind_vcpu_u(l4_cap_idx_t irq, l4_cap_idx_t thread, l4_umword_t cfg,
00251 l4_utcb_t *utcb) L4_NOTHROW
00252 {
00253 l4_msg_regs_t *m = l4_utcb_mr_u(utcb);
00254 m->mr[0] = L4_IRQ_SENDER_OP_BIND_VCPU;
00255 m->mr[1] = cfg;
00256 m->mr[2] = l4_map_obj_control(0, 0);
00257 m->mr[3] = l4_obj_fpage(thread, 0, L4_CAP_FPAGE_RWS).raw;
00258 return l4_ipc_call(irq, utcb, l4_msgtag(L4_PROTO_IRQ_SENDER, 2, 1, 0),
00259 L4_IPC_NEVER);
00260 }
00261
00262 L4_INLINE l4_msgtag_t
00263 l4_irq_trigger_u(l4_cap_idx_t irq, l4_utcb_t *utcb) L4_NOTHROW
00264 {
00265 return l4_ipc_send(irq, utcb, l4_msgtag(L4_PROTO_IRQ, 0, 0, 0),
00266 L4_IPC_BOTH_TIMEOUT_0);
00267 }
00268
00269 L4_INLINE l4_msgtag_t
00270 l4_irq_receive_u(l4_cap_idx_t irq, l4_timeout_t to, l4_utcb_t *utcb) L4_NOTHROW

```

```

00271 {
00272 l4_utcb_mr_u(utcb)->mr[0] = L4_IRQ_OP_EOI;
00273 return l4_ipc_call(irq, utcb, l4_msgtag(L4_PROTO_IRQ, 1, 0, 0), to);
00274 }
00275
00276 L4_INLINE l4_msgtag_t
00277 l4_irq_wait_u(l4_cap_idx_t irq, l4_umword_t *label,
00278 l4_timeout_t to, l4_utcb_t *utcb) L4_NOTHROW
00279 {
00280 l4_utcb_mr_u(utcb)->mr[0] = L4_IRQ_OP_EOI;
00281 return l4_ipc_send_and_wait(irq, utcb, l4_msgtag(L4_PROTO_IRQ, 1, 0, 0),
00282 label, to);
00283 }
00284
00285 L4_INLINE l4_msgtag_t
00286 l4_irq_unmask_u(l4_cap_idx_t irq, l4_utcb_t *utcb) L4_NOTHROW
00287 {
00288 l4_utcb_mr_u(utcb)->mr[0] = L4_IRQ_OP_EOI;
00289 return l4_ipc_send(irq, utcb, l4_msgtag(L4_PROTO_IRQ, 1, 0, 0), L4_IPC_NEVER);
00290 }
00291
00292
00293 L4_INLINE l4_msgtag_t
00294 l4_irq_detach(l4_cap_idx_t irq) L4_NOTHROW
00295 {
00296 return l4_irq_detach_u(irq, l4_utcb());
00297 }
00298
00299 L4_INLINE l4_msgtag_t
00300 l4_irq_bind_vcpu(l4_cap_idx_t irq, l4_cap_idx_t thread,
00301 l4_umword_t cfg) L4_NOTHROW
00302 {
00303 return l4_irq_bind_vcpu_u(irq, thread, cfg, l4_utcb());
00304 }
00305
00306 L4_INLINE l4_msgtag_t
00307 l4_irq_trigger(l4_cap_idx_t irq) L4_NOTHROW
00308 {
00309 return l4_irq_trigger_u(irq, l4_utcb());
00310 }
00311
00312 L4_INLINE l4_msgtag_t
00313 l4_irq_receive(l4_cap_idx_t irq, l4_timeout_t to) L4_NOTHROW
00314 {
00315 return l4_irq_receive_u(irq, to, l4_utcb());
00316 }
00317
00318 L4_INLINE l4_msgtag_t
00319 l4_irq_wait(l4_cap_idx_t irq, l4_umword_t *label,
00320 l4_timeout_t to) L4_NOTHROW
00321 {
00322 return l4_irq_wait_u(irq, label, to, l4_utcb());
00323 }
00324
00325 L4_INLINE l4_msgtag_t
00326 l4_irq_unmask(l4_cap_idx_t irq) L4_NOTHROW
00327 {
00328 return l4_irq_unmask_u(irq, l4_utcb());
00329 }
00330

```

## 17.235 x86/I4/util/irq.h File Reference

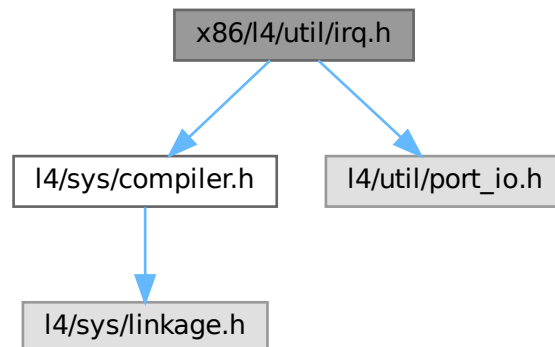
some PIC and hardware interrupt related functions

```

#include <l4/sys/compiler.h>
#include <l4/util/port_io.h>

```

Include dependency graph for irq.h:



### 17.235.1 Detailed Description

some PIC and hardware interrupt related functions

Date

2003

Author

Jork Loeser [jork.loeser@inf.tu-dresden.de](mailto:jork.loeser@inf.tu-dresden.de) Frank Mehnert [fm3@os.inf.tu-dresden.de](mailto:fm3@os.inf.tu-dresden.de)

Definition in file [irq.h](#).

## 17.236 irq.h

[Go to the documentation of this file.](#)

```

00001
00008
00009 /*
00010 * (c) 2003-2009 Author(s)
00011 * economic rights: Technische Universität Dresden (Germany)
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014
00015 #ifndef __L4_IRQ_H__
00016 #define __L4_IRQ_H__
00017
00018 #include <l4/sys/compiler.h>
00019 #include <l4/util/port_io.h>
00020
00021 L4_BEGIN_DECLS
00022
00027
00030 static inline void
00031 l4util_cli (void)
00032 {
00033 __asm__ __volatile__ ("cli" : : : "memory");

```

```

00034 }
00035
00038 static inline void
00039 l4util_sti (void)
00040 {
00041 __asm__ __volatile__ ("sti" : : : "memory");
00042 }
00043
00047 static inline void
00048 l4util_flags_save (l4_umword_t *flags)
00049 {
00050 __asm__ __volatile__ ("pushfl ; popl %0 " : "=g" (*flags) : : "memory");
00051 }
00052
00055 static inline void
00056 l4util_flags_restore (l4_umword_t *flags)
00057 {
00058 __asm__ __volatile__ ("pushl %0 ; popfl " : : "g" (*flags) : "memory");
00059 }
00061
00062 L4_END_DECLS
00063
00064 #endif

```

## 17.237 backend

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2010 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 #pragma once
00010
00011 #include <l4/l4re_vfs/vfs.h>
00012 #include <l4/crtn/initpriorities.h>
00013
00014 extern "C" void l4re_vfs_select_poll_notify(void);
00015
00016 namespace L4Re { namespace Vfs {
00017
00019 extern L4Re::Vfs::Ops *vfs_ops asm ("l4re_env_posix_vfs_ops");
00020
00021 class Mount_tree;
00022
00030 class Be_file : public File
00031 {
00032 public:
00033 void *operator new (size_t size) noexcept
00034 { return vfs_ops->malloc(size); }
00035
00036 void *operator new (size_t, void *m) noexcept
00037 { return m; }
00038
00039 void operator delete (void *m)
00040 { vfs_ops->free(m); }
00041
00042 // used in close, to unlock all locks of a file (as POSIX says)
00043 int unlock_all_locks() noexcept override
00044 { return 0; }
00045
00046 // for mmap
00047 L4::Cap<L4Re::Dataspace> data_space() noexcept override
00048 { return L4::Cap<L4Re::Dataspace>::Invalid; }
00049
00051 ssize_t readv(const struct iovec*, int) noexcept override
00052 { return -EINVAL; }
00053
00055 ssize_t writev(const struct iovec*, int) noexcept override
00056 { return -EINVAL; }
00057
00059 ssize_t pwritev(const struct iovec*, int, off64_t) noexcept override
00060 { return -EINVAL; }
00061
00063 ssize_t preadv(const struct iovec*, int, off64_t) noexcept override
00064 { return -EINVAL; }
00065
00067 off64_t lseek(off64_t, int) noexcept override
00068 { return -ESPIPE; }
00069
00071 int ftruncate(off64_t) noexcept override

```

```

00072 { return -EINVAL; }
00073
00075 int fsync() const noexcept override
00076 { return -EINVAL; }
00077
00079 int fdatsync() const noexcept override
00080 { return -EINVAL; }
00081
00083 int ioctl(unsigned long, va_list) noexcept override
00084 { return -EINVAL; }
00085
00086 int fstat(struct stat64 *) const noexcept override
00087 { return -EINVAL; }
00088
00090 int fchmod(mode_t) noexcept override
00091 { return -EINVAL; }
00092
00094 int get_status_flags() const noexcept override
00095 { return 0; }
00096
00098 int set_status_flags(long) noexcept override
00099 { return 0; }
00100
00102 int get_lock(struct flock64 *) noexcept override
00103 { return -EOLCK; }
00104
00106 int set_lock(struct flock64 *, bool) noexcept override
00107 { return -EOLCK; }
00108
00110 int faccessat(const char *, int, int) noexcept override
00111 { return -ENOTDIR; }
00112
00114 int fchmodat(const char *, mode_t, int) noexcept override
00115 { return -ENOTDIR; }
00116
00118 int utime(const struct utimbuf *) noexcept override
00119 { return -EROFS; }
00120
00122 int utimes(const struct timeval [2]) noexcept override
00123 { return -EROFS; }
00124
00126 int utimensat(const char *, const struct timespec [2], int) noexcept override
00127 { return -EROFS; }
00128
00130 int mkdir(const char *, mode_t) noexcept override
00131 { return -ENOTDIR; }
00132
00134 int unlink(const char *) noexcept override
00135 { return -ENOTDIR; }
00136
00138 int rename(const char *, const char *) noexcept override
00139 { return -ENOTDIR; }
00140
00142 int link(const char *, const char *) noexcept override
00143 { return -ENOTDIR; }
00144
00146 int symlink(const char *, const char *) noexcept override
00147 { return -EPERM; }
00148
00150 int rmdir(const char *) noexcept override
00151 { return -ENOTDIR; }
00152
00154 ssize_t readlink(char *, size_t) override
00155 { return -EINVAL; }
00156
00157 ssize_t getdents(char *, size_t) noexcept override
00158 { return -ENOTDIR; }
00159
00160
00161
00162 // Socket interface
00163 int bind(sockaddr const *, socklen_t) noexcept override
00164 { return -ENOTSOCK; }
00165
00166 int connect(sockaddr const *, socklen_t) noexcept override
00167 { return -ENOTSOCK; }
00168
00169 ssize_t send(void const *, size_t, int) noexcept override
00170 { return -ENOTSOCK; }
00171
00172 ssize_t recv(void *, size_t, int) noexcept override
00173 { return -ENOTSOCK; }
00174
00175 ssize_t sendto(void const *, size_t, int, sockaddr const *, socklen_t) noexcept
00176 override
00177 { return -ENOTSOCK; }
00178

```

```

00179 ssize_t recvfrom(void *, size_t, int, sockaddr *, socklen_t *) noexcept override
00180 { return -ENOTSOCK; }
00181
00182 ssize_t sendmsg(msghdr const *, int) noexcept override
00183 { return -ENOTSOCK; }
00184
00185 ssize_t recvmsg(msghdr *, int) noexcept override
00186 { return -ENOTSOCK; }
00187
00188 int getsockopt(int, int, void *, socklen_t *) noexcept override
00189 { return -ENOTSOCK; }
00190
00191 int setsockopt(int, int, void const *, socklen_t) noexcept override
00192 { return -ENOTSOCK; }
00193
00194 int listen(int) noexcept override
00195 { return -ENOTSOCK; }
00196
00197 int accept(sockaddr *, socklen_t *) noexcept override
00198 { return -ENOTSOCK; }
00199
00200 int shutdown(int) noexcept override
00201 { return -ENOTSOCK; }
00202
00203 int getsockname(sockaddr *, socklen_t *) noexcept override
00204 { return -ENOTSOCK; }
00205
00206 int getpeername(sockaddr *, socklen_t *) noexcept override
00207 { return -ENOTSOCK; }
00208
00217 bool check_ready(Ready_type) noexcept override
00218 { return false; }
00219
00220 ~Be_file() noexcept = 0;
00221
00222 private:
00224 int get_entry(const char *, int, mode_t, cxx::Ref_ptr<File> *) noexcept override
00225 { return -ENOTDIR; }
00226
00227 protected:
00228 const char *get_mount(const char *path, cxx::Ref_ptr<File> *dir) noexcept;
00229 };
00230
00231 inline
00232 Be_file::~~Be_file() noexcept {}
00233
00234 class Be_file_pos : public Be_file
00235 {
00236 public:
00237 Be_file_pos() noexcept : Be_file(), _pos(0) {}
00238
00239 virtual off64_t size() const noexcept = 0;
00240
00241 ssize_t readv(const struct iovec *v, int iovcnt) noexcept override
00242 {
00243 ssize_t r = preadv(v, iovcnt, _pos);
00244 if (r > 0)
00245 _pos += r;
00246 return r;
00247 }
00248
00249 ssize_t writev(const struct iovec *v, int iovcnt) noexcept override
00250 {
00251 ssize_t r = pwritev(v, iovcnt, _pos);
00252 if (r > 0)
00253 _pos += r;
00254 return r;
00255 }
00256
00257 ssize_t preadv(const struct iovec *v, int iovcnt, off64_t offset) noexcept override = 0;
00258 ssize_t pwritev(const struct iovec *v, int iovcnt, off64_t offset) noexcept override = 0;
00259
00260 off64_t lseek(off64_t offset, int whence) noexcept override
00261 {
00262 off64_t r;
00263 switch (whence)
00264 {
00265 case SEEK_SET: r = offset; break;
00266 case SEEK_CUR: r = _pos + offset; break;
00267 case SEEK_END: r = size() + offset; break;
00268 default: return -EINVAL;
00269 };
00270
00271 if (r < 0)
00272 return -EINVAL;
00273
00274 _pos = r;

```

```

00275 return _pos;
00276 }
00277
00278 ~Be_file_pos() noexcept = 0;
00279
00280 protected:
00281 off64_t pos() const noexcept { return _pos; }
00282
00283 private:
00284 off64_t _pos;
00285 };
00286
00287 inline Be_file_pos::~Be_file_pos() noexcept {}
00288
00289 class Be_file_stream : public Be_file
00290 {
00291 public:
00292 ssize_t preadv(const struct iovec *v, int iovcnt, off64_t) noexcept override
00293 { return readv(v, iovcnt); }
00294
00295 ssize_t pwritev(const struct iovec *v, int iovcnt, off64_t) noexcept override
00296 { return writev(v, iovcnt); }
00297
00298 ~Be_file_stream() noexcept = 0;
00299 };
00300
00301 inline Be_file_stream::~Be_file_stream() noexcept {}
00302
00303 class Be_file_system : public File_system
00304 {
00305 private:
00306 char const *const _fstype;
00307
00308 public:
00309 explicit Be_file_system(char const *fstype) noexcept
00310 : File_system(), _fstype(fstype)
00311 {
00312 vfs_ops->register_file_system(this);
00313 }
00314
00315 ~Be_file_system() noexcept
00316 {
00317 vfs_ops->unregister_file_system(this);
00318 }
00319
00320 char const *type() const noexcept override { return _fstype; }
00321 };
00322
00323 /* Make sure filesystems can register before the constructor of libmount
00324 * runs */
00325 #define L4RE_VFS_FILE_SYSTEM_ATTRIBUTE \
00326 __attribute__((init_priority(INIT_PRIO_LATE)))
00327
00328 }

```

## 17.238 default\_ops\_impl.h

```

00001 // vi:ft=cpp
00002 /*
00003 * Copyright (C) 2016, 2023-2024 Kernkonzept GmbH.
00004 * Author(s): Alexander Warg <alexander.warg@kernkonzept.com>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #include <l4/cxx/static_container>
00009
00010 namespace {
00011 struct Vfs_init
00012 {
00013 // The Static_containers are used to prevent automatic destruction during
00014 // program shutdown. At least the `vfs` object must never be destructed
00015 // because any later attempt to do any kind of file-descriptor access in
00016 // the program would crash, and we could not be sure that the destructor
00017 // would really be executed after each possible operation using files or file
00018 // descriptors.
00019 cxx::Static_container<Vfs> vfs;
00020
00021 // The Static_containers below are just for providing ordering. The factories
00022 // must be initialized after the `vfs` object.
00023 cxx::Static_container<L4Re::Vfs::File_factory_t<L4Re::Dataspace, L4Re::Core::Ro_file> > ro_file;
00024 cxx::Static_container<L4Re::Vfs::File_factory_t<L4Re::Namespace, L4Re::Core::Ns_dir> > ns_dir;

```



```

00025 cxx::Static_container<L4Re::Vfs::File_factory_t<L4::Vcon, L4Re::Core::Vcon_stream> > vcon_stream;
00026
00027 Vfs_init()
00028 {
00029 vfs.construct();
00030 __rtld_l4re_env_posix_vfs_ops = vfs;
00031 ns_dir.construct();
00032 auto ns_ptr = cxx::ref_ptr(ns_dir.get());
00033 vfs->register_file_factory(ns_ptr);
00034 ns_ptr.release(); // prevent deletion of static object
00035
00036 ro_file.construct();
00037 auto ro_ptr = cxx::ref_ptr(ro_file.get());
00038 vfs->register_file_factory(ro_ptr);
00039 ro_ptr.release(); // prevent deletion of static object
00040
00041 vcon_stream.construct();
00042 auto vcon_ptr = cxx::ref_ptr(vcon_stream.get());
00043 vfs->register_file_factory(vcon_ptr);
00044 vcon_ptr.release(); // prevent deletion of static object
00045 }
00046 };
00047
00048 static Vfs_init __vfs_init __attribute__((init_priority(INIT_PRIO_VFS_INIT)));
00049
00050 };

```

## 17.239 fd\_store.h

```

00001 /*
00002 * (c) 2010 Alexander Warg <warg@os.inf.tu-dresden.de>
00003 * economic rights: Technische Universität Dresden (Germany)
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/l4re_vfs/vfs.h>
00010
00011 namespace L4Re { namespace Core {
00012
00013 using cxx::Ref_ptr;
00014
00015 class Fd_store
00016 {
00017 public:
00018 enum { MAX_FILES = 50 };
00019
00020 Fd_store() noexcept : _fd_hint(0) {}
00021
00022 int alloc() noexcept;
00023 void free(int fd) noexcept;
00024 bool check_fd(int fd) noexcept;
00025 Ref_ptr<L4Re::Vfs::File> get(int fd) noexcept;
00026 void set(int fd, Ref_ptr<L4Re::Vfs::File> const &f) noexcept;
00027
00028 private:
00029 int _fd_hint;
00030 Ref_ptr<L4Re::Vfs::File> _files[MAX_FILES];
00031 };
00032
00033 inline
00034 bool
00035 Fd_store::check_fd(int fd) noexcept
00036 {
00037 return fd >= 0 && fd < MAX_FILES;
00038 }
00039
00040 inline
00041 Ref_ptr<L4Re::Vfs::File>
00042 Fd_store::get(int fd) noexcept
00043 {
00044 if (check_fd(fd))
00045 return _files[fd];
00046 return Ref_ptr<>::Nil;
00047 }
00048
00049 inline
00050 void
00051 Fd_store::set(int fd, Ref_ptr<L4Re::Vfs::File> const &f) noexcept
00052 {
00053 _files[fd] = f;
00054 }

```

```
00055 }
00056
00057 }}
```

## 17.240 fd\_store\_impl.h

```
00001 /*
00002 * (c) 2010 Alexander Warg <warg@os.inf.tu-dresden.de>
00003 * economic rights: Technische Universität Dresden (Germany)
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #include "fd_store.h"
00008
00009 namespace L4Re { namespace Core {
00010
00011 int
00012 Fd_store::alloc() noexcept
00013 {
00014 for (int i = _fd_hint; i < MAX_FILES; ++i)
00015 {
00016 if (!_files[i])
00017 {
00018 _fd_hint = i + 1;
00019 return i;
00020 }
00021 }
00022 return -1;
00023 }
00024 }
00025
00026 void
00027 Fd_store::free(int fd) noexcept
00028 {
00029 _files[fd] = 0;
00030 if (fd < _fd_hint)
00031 _fd_hint = fd;
00032 }
00033
00034 }}
00035
```

## 17.241 ns\_fs.h

```
00001 /*
00002 * (c) 2010 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include <l4/l4re_vfs/backend>
00011 #include <l4/sys/capability>
00012 #include <l4/re/namespace>
00013 #include <l4/re/unique_cap>
00014
00015 namespace L4Re { namespace Core {
00016
00017 using cxx::Ref_ptr;
00018
00019 class Env_dir : public L4Re::Vfs::Be_file
00020 {
00021 public:
00022 explicit Env_dir(L4Re::Env const *env)
00023 : _env(env), _current_cap_entry(env->initial_caps())
00024 {}
00025
00026 ssize_t readv(const struct iovec*, int) noexcept override { return -EISDIR; }
00027 ssize_t writev(const struct iovec*, int) noexcept override { return -EISDIR; }
00028 ssize_t preadv(const struct iovec*, int, off64_t) noexcept override { return -EISDIR; }
00029 ssize_t pwritev(const struct iovec*, int, off64_t) noexcept override { return -EISDIR; }
00030 int fstat(struct stat64 *) const noexcept override;
00031 int faccessat(const char *path, int mode, int flags) noexcept override;
00032 int get_entry(const char *path, int flags, mode_t mode,
00033 Ref_ptr<L4Re::Vfs::File> *) noexcept override;
00034 ssize_t getdents(char *, size_t) noexcept override;
00035 };
00036 }
```

```

00035
00036 ~Env_dir() noexcept {}
00037
00038 private:
00039 int get_ds(const char *path, L4Re::Unique_cap<L4Re::Dataspace> *ds) noexcept;
00040 bool check_type(Env::Cap_entry const *e, long protocol) noexcept;
00041
00042 L4Re::Env const *_env;
00043 Env::Cap_entry const *_current_cap_entry;
00044 };
00045
00046 class Ns_dir : public L4Re::Vfs::Be_file
00047 {
00048 public:
00049 explicit Ns_dir(L4::Cap<L4Re::Namespace> ns)
00050 : _ns(ns), _current_dir_pos(0)
00051 {}
00052
00053 ssize_t readv(const struct iovec*, int) noexcept override { return -EISDIR; }
00054 ssize_t writev(const struct iovec*, int) noexcept override { return -EISDIR; }
00055 ssize_t preadv(const struct iovec*, int, off64_t) noexcept override { return -EISDIR; }
00056 ssize_t pwritev(const struct iovec*, int, off64_t) noexcept override { return -EISDIR; }
00057 int fstat(struct stat64 *) const noexcept override;
00058 int faccessat(const char *path, int mode, int flags) noexcept override;
00059 int get_entry(const char *path, int flags, mode_t mode,
00060 Ref_ptr<L4Re::Vfs::File> *) noexcept override;
00061 ssize_t getdents(char *, size_t) noexcept override;
00062
00063 ~Ns_dir() noexcept {}
00064
00065 private:
00066 int get_ds(const char *path, L4Re::Unique_cap<L4Re::Dataspace> *ds) noexcept;
00067
00068 L4::Cap<L4Re::Namespace> _ns;
00069 size_t _current_dir_pos;
00070 };
00071
00072 }

```

## 17.242 ns\_fs\_impl.h

```

00001 /*
00002 * (c) 2010 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #include "ns_fs.h"
00009
00010 #include <l4/re/dataspace>
00011 #include <l4/re/util/env_ns>
00012 #include <l4/re/unique_cap>
00013 #include <dirent.h>
00014
00015 namespace L4Re { namespace Core {
00016
00017 static
00018 Ref_ptr<L4Re::Vfs::File>
00019 cap_to_vfs_object(L4::Cap<void> o, int *err)
00020 {
00021 L4::Cap<L4::Meta> m = L4::cap_reinterpret_cast<L4::Meta>(o);
00022 long proto = 0;
00023 char name_buf[256];
00024 L4::Ipc::String<char> name(sizeof(name_buf), name_buf);
00025 int r = l4_error(m->interface(0, &proto, &name));
00026 *err = -ENOPROTOPT;
00027 if (r < 0)
00028 // could not get type of object so bail out
00029 return Ref_ptr<L4Re::Vfs::File>();
00030
00031 *err = -EPROTO;
00032 Ref_ptr<L4Re::Vfs::File_factory> factory;
00033
00034 if (proto != 0)
00035 factory = L4Re::Vfs::vfs_ops->get_file_factory(proto);
00036
00037 if (!factory)
00038 factory = L4Re::Vfs::vfs_ops->get_file_factory(name.data);
00039
00040 if (!factory)
00041 return Ref_ptr<L4Re::Vfs::File>();
00042

```

```

00043 *err = -ENOMEM;
00044 return factory->create(o);
00045 }
00046
00047
00048 int
00049 Ns_dir::get_ds(const char *path, L4Re::Unique_cap<L4Re::Dataspace> *ds) noexcept
00050 {
00051 auto file = L4Re::make_unique_cap<L4Re::Dataspace>(L4Re::virt_cap_alloc);
00052
00053 if (!file.is_valid())
00054 return -ENOMEM;
00055
00056 int err = _ns->query(path, file.get());
00057
00058 if (err < 0)
00059 return -ENOENT;
00060
00061 *ds = cxx::move(file);
00062 return err;
00063 }
00064
00065 int
00066 Ns_dir::get_entry(const char *path, int /*flags*/, mode_t /*mode*/,
00067 Ref_ptr<L4Re::Vfs::File> *f) noexcept
00068 {
00069 if (!*path)
00070 {
00071 *f = cxx::ref_ptr(this);
00072 return 0;
00073 }
00074
00075 L4Re::Unique_cap<Dataspace> file;
00076 int err = get_ds(path, &file);
00077
00078 if (err < 0)
00079 return -ENOENT;
00080
00081 cxx::Ref_ptr<L4Re::Vfs::File> fi = cap_to_vfs_object(file.get(), &err);
00082 if (!fi)
00083 return err;
00084
00085 file.release();
00086 *f = cxx::move(fi);
00087 return 0;
00088 }
00089
00090 int
00091 Ns_dir::faccessat(const char *path, int mode, int /*flags*/) noexcept
00092 {
00093 auto tmpcap = L4Re::make_unique_cap<void>(L4Re::virt_cap_alloc);
00094
00095 if (!tmpcap.is_valid())
00096 return -ENOMEM;
00097
00098 if (_ns->query(path, tmpcap.get()))
00099 return -ENOENT;
00100
00101 if (mode & W_OK)
00102 return -EACCES;
00103
00104 return 0;
00105 }
00106
00107 int
00108 Ns_dir::fstat(struct stat64 *b) const noexcept
00109 {
00110 b->st_dev = 1;
00111 b->st_ino = 1;
00112 b->st_mode = S_IRWXU | S_IFDIR;
00113 b->st_nlink = 0;
00114 b->st_uid = 0;
00115 b->st_gid = 0;
00116 b->st_rdev = 0;
00117 b->st_size = 0;
00118 b->st_blksize = 0;
00119 b->st_blocks = 0;
00120 b->st_atime = 0;
00121 b->st_mtime = 0;
00122 b->st_ctime = 0;
00123 return 0;
00124 }
00125
00126 ssize_t
00127 Ns_dir::getdents(char *buf, size_t dest_sz) noexcept
00128 {
00129 struct dirent64 *dest = reinterpret_cast<struct dirent64 *>(buf);

```

```

00130 ssize_t ret = 0;
00131 l4_addr_t infoaddr;
00132 size_t infosz;
00133
00134 L4Re::Unique_cap<Dataspace> dirinfofile;
00135 int err = get_ds(".dirinfo", &dirinfofile);
00136 if (err)
00137 return 0;
00138
00139 infosz = dirinfofile->size();
00140 if (infosz <= 0)
00141 return 0;
00142
00143 infoaddr = L4_PAGESIZE;
00144 err = L4Re::Env::env()->rm()->attach(&infoaddr, infosz,
00145 Rm::F::Search_addr | Rm::F::R,
00146 dirinfofile.get(), 0);
00147 if (err < 0)
00148 return 0;
00149
00150 char *p = reinterpret_cast<char *>(infoaddr) + _current_dir_pos;
00151 char *end = reinterpret_cast<char *>(infoaddr) + infosz;
00152
00153 char *current_dirinfo_entry = p;
00154 while (dest && p < end)
00155 {
00156 // parse lines of dirinfofile
00157 long len = 0;
00158 for (; p < end && *p >= '0' && *p <= '9'; ++p)
00159 {
00160 len *= 10;
00161 len += *p - '0';
00162 }
00163
00164 if (len == 0)
00165 break;
00166
00167 if (p == end)
00168 break;
00169
00170 if (*p != ':')
00171 break;
00172 p++; // skip colon
00173
00174 if (p + len >= end)
00175 break;
00176
00177 unsigned l = len + 1;
00178 if (l > sizeof(dest->d_name))
00179 l = sizeof(dest->d_name);
00180
00181 unsigned n = offsetof(struct dirent64, d_name) + l;
00182 n = (n + sizeof(long) - 1) & ~(sizeof(long) - 1);
00183
00184 if (n > dest_sz)
00185 break;
00186
00187 dest->d_ino = 1;
00188 dest->d_off = 0;
00189 memcpy(dest->d_name, p, l - 1);
00190 dest->d_name[l - 1] = 0;
00191 dest->d_reclen = n;
00192 dest->d_type = DT_UNKNOWN;
00193 ret += n;
00194 dest_sz -= n;
00195
00196 // next entry
00197 dest = reinterpret_cast<struct dirent64 *>
00198 (reinterpret_cast<unsigned long>(dest) + n);
00199
00200 // next infodirfile line
00201 p += len;
00202 while (p < end && *p && (*p == '\n' || *p == '\r'))
00203 p++;
00204
00205 current_dirinfo_entry = p;
00206 }
00207
00208 _current_dir_pos = current_dirinfo_entry - reinterpret_cast<char *>(infoaddr);
00209
00210 if (!ret) // hack since we should only reset this at open times
00211 _current_dir_pos = 0;
00212
00213 L4Re::Env::env()->rm()->detach(infoaddr, 0);
00214
00215 return ret;
00216 }

```

```

00217
00218 int
00219 Env_dir::get_ds(const char *path, L4Re::Unique_cap<L4Re::Dataspace> *ds) noexcept
00220 {
00221 Vfs::Path p(path);
00222 Vfs::Path first = p.strip_first();
00223
00224 if (first.empty())
00225 return -ENOENT;
00226
00227 L4::Cap<L4Re::Namespace>
00228 c = _env->get_cap<L4Re::Namespace>(first.path(), first.length());
00229
00230 if (!c.is_valid())
00231 return -ENOENT;
00232
00233 if (p.empty())
00234 {
00235 *ds = L4Re::Unique_cap<L4Re::Dataspace>(L4::cap_reinterpret_cast<L4Re::Dataspace>(c));
00236 return 0;
00237 }
00238
00239 auto file = L4Re::make_unique_cap<L4Re::Dataspace>(L4Re::virt_cap_alloc);
00240
00241 if (!file.is_valid())
00242 return -ENOMEM;
00243
00244 int err = c->query(p.path(), p.length(), file.get());
00245
00246 if (err < 0)
00247 return -ENOENT;
00248
00249 *ds = cxx::move(file);
00250 return err;
00251 }
00252
00253 int
00254 Env_dir::get_entry(const char *path, int /*flags*/, mode_t /*mode*/,
00255 Ref_ptr<L4Re::Vfs::File> *f) noexcept
00256 {
00257 if (!*path)
00258 {
00259 *f = cxx::ref_ptr(this);
00260 return 0;
00261 }
00262
00263 L4Re::Unique_cap<Dataspace> file;
00264 int err = get_ds(path, &file);
00265
00266 if (err < 0)
00267 return -ENOENT;
00268
00269 cxx::Ref_ptr<L4Re::Vfs::File> fi = cap_to_vfs_object(file.get(), &err);
00270 if (!fi)
00271 return err;
00272
00273 file.release();
00274 *f = cxx::move(fi);
00275 return 0;
00276 }
00277
00278 int
00279 Env_dir::faccessat(const char *path, int mode, int /*flags*/) noexcept
00280 {
00281 Vfs::Path p(path);
00282 Vfs::Path first = p.strip_first();
00283
00284 if (first.empty())
00285 return -ENOENT;
00286
00287 L4::Cap<L4Re::Namespace>
00288 c = _env->get_cap<L4Re::Namespace>(first.path(), first.length());
00289
00290 if (!c.is_valid())
00291 return -ENOENT;
00292
00293 if (p.empty())
00294 {
00295 if (mode & W_OK)
00296 return -EACCES;
00297
00298 return 0;
00299 }
00300
00301 auto tmpcap = L4Re::make_unique_cap<void>(L4Re::virt_cap_alloc);
00302
00303 if (!tmpcap.is_valid())

```

```

00304 return -ENOMEM;
00305
00306 if (c->query(p.path(), p.length(), tmpcap.get()))
00307 return -ENOENT;
00308
00309 if (mode & W_OK)
00310 return -EACCES;
00311
00312 return 0;
00313 }
00314
00315 bool
00316 Env_dir::check_type(Env::Cap_entry const *e, long protocol) noexcept
00317 {
00318 L4::Cap<L4::Meta> m(e->cap);
00319 return m->supports(protocol).label();
00320 }
00321
00322 int
00323 Env_dir::fstat(struct stat64 *b) const noexcept
00324 {
00325 b->st_dev = 1;
00326 b->st_ino = 1;
00327 b->st_mode = S_IRWXU | S_IFDIR;
00328 b->st_nlink = 0;
00329 b->st_uid = 0;
00330 b->st_gid = 0;
00331 b->st_rdev = 0;
00332 b->st_size = 0;
00333 b->st_blksize = 0;
00334 b->st_blocks = 0;
00335 b->st_atime = 0;
00336 b->st_mtime = 0;
00337 b->st_ctime = 0;
00338 return 0;
00339 }
00340
00341 ssize_t
00342 Env_dir::getdents(char *buf, size_t sz) noexcept
00343 {
00344 struct dirent64 *d = reinterpret_cast<struct dirent64 *>(buf);
00345 ssize_t ret = 0;
00346
00347 while (d
00348 && _current_cap_entry
00349 && _current_cap_entry->flags != ~0UL)
00350 {
00351 unsigned l = strlen(_current_cap_entry->name) + 1;
00352 if (l > sizeof(d->d_name))
00353 l = sizeof(d->d_name);
00354
00355 unsigned n = offsetof(struct dirent64, d_name) + 1;
00356 n = (n + sizeof(long) - 1) & ~(sizeof(long) - 1);
00357
00358 if (n <= sz)
00359 {
00360 d->d_ino = 1;
00361 d->d_off = 0;
00362 memcpy(d->d_name, _current_cap_entry->name, l);
00363 d->d_name[l - 1] = 0;
00364 d->d_reclen = n;
00365 if (check_type(_current_cap_entry, L4Re::Namespace::Protocol))
00366 d->d_type = DT_DIR;
00367 else if (check_type(_current_cap_entry, L4Re::Dataspace::Protocol))
00368 d->d_type = DT_REG;
00369 else
00370 d->d_type = DT_UNKNOWN;
00371 ret += n;
00372 sz -= n;
00373 d = reinterpret_cast<struct dirent64 *>
00374 (reinterpret_cast<unsigned long*>(d) + n);
00375 _current_cap_entry++;
00376 }
00377 else
00378 return ret;
00379 }
00380
00381 // bit of a hack because we should only (re)set this when opening the dir
00382 if (!ret)
00383 _current_cap_entry = _env->initial_caps();
00384
00385 return ret;
00386 }
00387
00388 }}

```

## 17.243 ro\_file.h

```

00001 /*
00002 * (c) 2010 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include <l4/l4re_vfs/backend>
00011
00012 namespace L4Re { namespace Core {
00013
00014 class Ro_file : public L4Re::Vfs::Be_file_pos
00015 {
00016 private:
00017 L4::Cap<L4Re::Dataspace> _ds;
00018 off64_t _size;
00019 char const *_addr;
00020
00021 public:
00022 explicit Ro_file(L4::Cap<L4Re::Dataspace> ds) noexcept
00023 : Be_file_pos(), _ds(ds), _addr(0)
00024 {
00025 _size = _ds->size();
00026 }
00027
00028 L4::Cap<L4Re::Dataspace> data_space() noexcept override { return _ds; }
00029
00030 int fstat(struct stat64 *buf) const noexcept override;
00031
00032 int ioctl(unsigned long, va_list) noexcept override;
00033
00034 off64_t size() const noexcept override { return _size; }
00035
00036 int get_status_flags() const noexcept override
00037 { return O_RDONLY; }
00038
00039 int set_status_flags(long) noexcept override
00040 { return 0; }
00041
00042 bool check_ready(Ready_type rt) noexcept override
00043 { return rt == Read; }
00044
00045 ~Ro_file() noexcept;
00046
00047 private:
00048 ssize_t read_single(const struct iovec*, off64_t) noexcept;
00049 ssize_t preadv(const struct iovec *, int, off64_t) noexcept override;
00050 ssize_t pwritev(const struct iovec *, int , off64_t) noexcept override;
00051 };
00052
00053
00054
00055 }
00056
00057 }
00058
00059 }
00060
00061 }
00062
00063
00064 }

```

## 17.244 ro\_file\_impl.h

```

00001 /*
00002 * (c) 2010 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00009 #include "ro_file.h"
00010
00011 #include <sys/ioctl.h>
00012
00013 #include <l4/re/env>
00014
00015 namespace L4Re { namespace Core {
00016
00017 Ro_file::~~Ro_file() noexcept
00018 {
00019 if (_addr)
00020 L4Re::Env::env()->rm()->detach(l4_addr_t(_addr), 0);
00021
00022 L4Re::virt_cap_alloc->release(_ds);
00023 }
00024
00025
00026
00027
00028
00029
00030
00031
00032
00033
00034
00035
00036
00037
00038
00039
00040
00041
00042
00043
00044
00045
00046
00047
00048
00049
00050
00051
00052
00053
00054
00055
00056
00057
00058
00059
00060
00061
00062
00063
00064
00065
00066
00067
00068
00069
00070
00071
00072
00073
00074
00075
00076
00077
00078
00079
00080
00081
00082
00083
00084
00085
00086
00087
00088
00089
00090
00091
00092
00093
00094
00095
00096
00097
00098
00099
00100
00101
00102
00103
00104
00105
00106
00107
00108
00109
00110
00111
00112
00113
00114
00115
00116
00117
00118
00119
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
00130
00131
00132
00133
00134
00135
00136
00137
00138
00139
00140
00141
00142
00143
00144
00145
00146
00147
00148
00149
00150
00151
00152
00153
00154
00155
00156
00157
00158
00159
00160
00161
00162
00163
00164
00165
00166
00167
00168
00169
00170
00171
00172
00173
00174
00175
00176
00177
00178
00179
00180
00181
00182
00183
00184
00185
00186
00187
00188
00189
00190
00191
00192
00193
00194
00195
00196
00197
00198
00199
00200
00201
00202
00203
00204
00205
00206
00207
00208
00209
00210
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232
00233
00234
00235
00236
00237
00238
00239
00240
00241
00242
00243
00244
00245
00246
00247
00248
00249
00250
00251
00252
00253
00254
00255
00256
00257
00258
00259
00260
00261
00262
00263
00264
00265
00266
00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000

```



```

00024
00025 int
00026 Ro_file::fstat(struct stat64 *buf) const noexcept
00027 {
00028 static int fake = 0;
00029
00030 memset(buf, 0, sizeof(*buf));
00031 buf->st_size = _size;
00032 buf->st_mode = S_IFREG | 0644;
00033 buf->st_dev = _ds.cap();
00034 buf->st_ino = ++fake;
00035 buf->st_blksize = L4_PAGESIZE;
00036 buf->st_blocks = 14_round_page(_size);
00037 return 0;
00038 }
00039
00040 ssize_t
00041 Ro_file::read_single(const struct iovec *vec, off64_t pos) noexcept
00042 {
00043 off64_t l = vec->iiov_len;
00044 if (_size - pos < l)
00045 l = _size - pos;
00046
00047 if (l > 0)
00048 {
00049 Vfs_config::memcpy(vec->iiov_base, _addr + pos, l);
00050 return l;
00051 }
00052
00053 return 0;
00054 }
00055
00056 ssize_t
00057 Ro_file::preadv(const struct iovec *vec, int cnt, off64_t offset) noexcept
00058 {
00059 if (!_addr)
00060 {
00061 void const *file = reinterpret_cast<void*>(L4_PAGESIZE);
00062 long err = L4Re::Env::env()->rm()->attach(&file, _size,
00063 Rm::F::Search_addr | Rm::F::R,
00064 _ds, 0);
00065
00066 if (err < 0)
00067 return err;
00068
00069 _addr = static_cast<char const *>(file);
00070 }
00071
00072 ssize_t l = 0;
00073
00074 while (cnt > 0)
00075 {
00076 ssize_t r = read_single(vec, offset);
00077 offset += r;
00078 l += r;
00079
00080 if (static_cast<size_t>(r) < vec->iiov_len)
00081 return l;
00082
00083 ++vec;
00084 --cnt;
00085 }
00086 return l;
00087 }
00088
00089 ssize_t
00090 Ro_file::pwritev(const struct iovec *, int, off64_t) noexcept
00091 {
00092 return -EROFS;
00093 }
00094
00095 int
00096 Ro_file::ioctl(unsigned long v, va_list args) noexcept
00097 {
00098 switch (v)
00099 {
00100 case FIONREAD: // return amount of data still available
00101 int *available = va_arg(args, int *);
00102 *available = _size - pos();
00103 return 0;
00104 };
00105 return -ENOTTY;
00106 }
00107
00108 {}

```

## 17.245 vcon\_stream.h

```

00001 /*
00002 * (c) 2010 Alexander Warg <warg@os.inf.tu-dresden.de>
00003 * economic rights: Technische Universität Dresden (Germany)
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/sys/capability>
00010 #include <l4/sys/vcon>
00011 #include <l4/sys/semaphore>
00012
00013 #include <l4/l4re_vfs/backend>
00014
00015 namespace L4Re { namespace Core {
00016
00017 class Vcon_stream : public L4Re::Vfs::Be_file_stream
00018 {
00019 private:
00020 L4::Cap<L4::Vcon> _s;
00021 L4::Cap<L4::Semaphore> _irq;
00022 unsigned _irq_bound;
00023
00024 public:
00025 explicit Vcon_stream(L4::Cap<L4::Vcon> s) noexcept;
00026
00027 ssize_t readv(const struct iovec*, int iovcnt) noexcept override;
00028 ssize_t writev(const struct iovec*, int iovcnt) noexcept override;
00029 int fstat(struct stat64 *buf) const noexcept override;
00030 int get_status_flags() const noexcept override { return O_RDWR; }
00031 int set_status_flags(long) noexcept override { return 0; }
00032 int ioctl(unsigned long request, va_list args) noexcept override;
00033
00034 ~Vcon_stream() noexcept {}
00035 void operator delete (void *) {}
00036 };
00037
00038 }}
```

## 17.246 vcon\_stream\_impl.h

```

00001 /*
00002 * (c) 2010 Alexander Warg <warg@os.inf.tu-dresden.de>
00003 * economic rights: Technische Universität Dresden (Germany)
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #include <l4/re/env>
00009 #include <l4/sys/factory>
00010 #include <l4/cxx/minmax>
00011
00012 #include "vcon_stream.h"
00013
00014 #include <limits.h>
00015 #include <termios.h>
00016 #include <unistd.h>
00017 #include <sys/ioctl.h>
00018 #include <sys/ttydefaults.h>
00019
00020 namespace L4Re { namespace Core {
00021 Vcon_stream::Vcon_stream(L4::Cap<L4::Vcon> s) noexcept
00022 : Be_file_stream(),
00023 _s(s), _irq(L4Re::virt_cap_alloc->alloc<L4::Semaphore>()), _irq_bound(false)
00024 {
00025 // [[maybe_unused]] int res =
00026 l4_error(L4Re::Env::env()->factory()->create(_irq));
00027 // (void)res; // handle errors!
00028 }
00029
00030 ssize_t
00031 Vcon_stream::readv(const struct iovec *iovec, int iovcnt) noexcept
00032 {
00033 if (iovcnt < 0)
00034 return -EINVAL;
00035
00036 if (!_irq_bound)
00037 {
00038 bool was_bound = __atomic_exchange_n(&_irq_bound, true, __ATOMIC_SEQ_CST);
00039 if (!was_bound)
```

```

00040 if (l4_error(_s->bind(0, _irq)) < 0)
00041 return -EIO;
00042 }
00043
00044 ssize_t bytes = 0;
00045 for (; iovcnt > 0; --iovcnt, ++iovec)
00046 {
00047 size_t len = cxx::min<size_t>(iovec->iov_len, SSIZE_MAX - bytes);
00048 if (len == 0)
00049 continue;
00050
00051 char *buf = static_cast<char *>(iovec->iov_base);
00052
00053 while (1)
00054 {
00055 size_t l = cxx::min<size_t>(L4_VCON_READ_SIZE, len);
00056 int ret = _s->read(buf, l);
00057
00058 if (ret > static_cast<int>(1))
00059 ret = 1;
00060
00061 if (ret < 0)
00062 return ret;
00063 else if (ret == 0)
00064 {
00065 if (bytes)
00066 return bytes;
00067
00068 ret = _s->read(buf, l);
00069 if (ret < 0)
00070 return ret;
00071 else if (ret == 0)
00072 {
00073 _irq->down();
00074 continue;
00075 }
00076 }
00077
00078 bytes += ret;
00079 len -= ret;
00080 buf += ret;
00081
00082 if (len == 0)
00083 break;
00084 }
00085 }
00086
00087 return bytes;
00088 }
00089
00090 ssize_t
00091 Vcon_stream::writev(const struct iovec *iovec, int iovcnt) noexcept
00092 {
00093 l4_msg_regs_t store;
00094 l4_msg_regs_t *mr = l4_utcb_mr();
00095
00096 if (iovcnt < 0)
00097 return -EINVAL;
00098
00099 Vfs_config::memcpy(&store, mr, sizeof(store));
00100
00101 ssize_t written = 0;
00102 while (iovcnt)
00103 {
00104 size_t sl = cxx::min<size_t>(iovec->iov_len, SSIZE_MAX - written);
00105 char const *b = static_cast<char const *>(iovec->iov_base);
00106
00107 for (; sl > L4_VCON_WRITE_SIZE;
00108 ; sl -= L4_VCON_WRITE_SIZE, b += L4_VCON_WRITE_SIZE,
00109 written += L4_VCON_WRITE_SIZE)
00110 _s->send(b, L4_VCON_WRITE_SIZE);
00111
00112 _s->send(b, sl);
00113
00114 written += sl;
00115
00116 ++iovec;
00117 --iovcnt;
00118 }
00119 Vfs_config::memcpy(mr, &store, sizeof(store));
00120 return written;
00121 }
00122
00123 int
00124 Vcon_stream::fstat(struct stat64 *buf) const noexcept
00125 {
00126 buf->st_size = 0;

```

```

00127 buf->st_mode = 0666;
00128 buf->st_dev = _s.cap();
00129 buf->st_ino = 0;
00130 return 0;
00131 }
00132
00133 int
00134 Vcon_stream::ioctl(unsigned long request, va_list args) noexcept
00135 {
00136 switch (request) {
00137 case TCGETS:
00138 {
00139 //vt100_tcgetattr(term, (struct termios *)argp);
00140
00141 struct termios *t = va_arg(args, struct termios *);
00142
00143 l4_vcon_attr_t l4a;
00144 if (!l4_error(_s->get_attr(&l4a)))
00145 {
00146 t->c_iflag = l4a.i_flags;
00147 t->c_oflag = l4a.o_flags; // output flags
00148 t->c_cflag = 0; // control flags
00149 t->c_lflag = l4a.l_flags; // local flags
00150 }
00151 else
00152 t->c_iflag = t->c_oflag = t->c_cflag = t->c_lflag = 0;
00153 #if 0
00154 //t->c_lflag |= ECHO; // if term->echo
00155 t->c_lflag |= ICANON; // if term->term_mode == VT100MODE_COOKED
00156 #endif
00157
00158 t->c_cc[VEOF] = CEOF;
00159 t->c_cc[VEOL] = _POSIX_VDISABLE;
00160 t->c_cc[VEOL2] = _POSIX_VDISABLE;
00161 t->c_cc[VERASE] = CERASE;
00162 t->c_cc[VWERASE] = CWERASE;
00163 t->c_cc[VKILL] = CKILL;
00164 t->c_cc[VREPRINT] = CREPRINT;
00165 t->c_cc[VINTR] = CINTR;
00166 t->c_cc[VQUIT] = _POSIX_VDISABLE;
00167 t->c_cc[VSUSP] = CSUSP;
00168 t->c_cc[VSTART] = CSTART;
00169 t->c_cc[VSTOP] = CSTOP;
00170 t->c_cc[VLNEXT] = CLNEXT;
00171 t->c_cc[VDISCARD] = CDISCARD;
00172 t->c_cc[VMIN] = CMIN;
00173 t->c_cc[VTIME] = 0;
00174
00175 }
00176
00177 return 0;
00178
00179 case TCSETS:
00180 case TCSETSW:
00181 case TCSETSF:
00182 {
00183 //vt100_tcsetattr(term, (struct termios *)argp);
00184 struct termios const *t = va_arg(args, struct termios const *);
00185
00186 // XXX: well, we're cheating, get this from the other side!
00187
00188 l4_vcon_attr_t l4a;
00189 l4a.i_flags = t->c_iflag;
00190 l4a.o_flags = t->c_oflag; // output flags
00191 l4a.l_flags = t->c_lflag; // local flags
00192 _s->set_attr(&l4a);
00193 }
00194 return 0;
00195
00196 default:
00197 break;
00198 };
00199 return -ENOTTY;
00200 }
00201
00202 }}

```

## 17.247 vfs\_impl.h

```

00001 /*
00002 * (c) 2008-2010 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00004 * Björn Döbel <doebel@os.inf.tu-dresden.de>

```

```

00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009
00010 #include "fd_store.h"
00011 #include "vcon_stream.h"
00012 #include "ns_fs.h"
00013
00014 #include <l4/bid_config.h>
00015 #include <l4/re/env>
00016 #include <l4/re/rm>
00017 #include <l4/re/dataspace>
00018 #include <l4/sys/assert.h>
00019 #include <l4/cxx/hlist>
00020 #include <l4/cxx/pair>
00021 #include <l4/cxx/std_alloc>
00022
00023 #include <l4/l4re_vfs/backend>
00024 #include <l4/re/shared_cap>
00025
00026 #include <unistd.h>
00027 #include <stdarg.h>
00028 #include <errno.h>
00029 #include <sys/uio.h>
00030
00031 #if 0
00032 #include <l4/sys/kdebug.h>
00033 static int debug_mmap = 1;
00034 #define DEBUG_LOG(level, dbg...) do { if (level) dbg } while (0)
00035 #else
00036 #define DEBUG_LOG(level, dbg...) do { } while (0)
00037 #endif
00038
00044 #define USE_BIG_ANON_DS
00045
00046 using L4Re::Rm;
00047
00048 namespace {
00049
00050 using cxx::Ref_ptr;
00051
00052 class Fd_store : public L4Re::Core::Fd_store
00053 {
00054 public:
00055 Fd_store() noexcept;
00056 };
00057
00058 // for internal Vcon_streams we want to have a placement new operator, so
00059 // inherit and add one
00060 class Std_stream : public L4Re::Core::Vcon_stream
00061 {
00062 public:
00063 Std_stream(L4::Cap<L4::Vcon> c) : L4Re::Core::Vcon_stream(c) {}
00064 };
00065
00066 Fd_store::Fd_store() noexcept
00067 {
00068 // use this strange way to prevent deletion of the stdio object
00069 // this depends on Fd_store to being a singleton !!!
00070 static char m[sizeof(Std_stream)] __attribute__((aligned(sizeof(long))));
00071 Std_stream *s = new (m) Std_stream(L4Re::Env::env()->log());
00072 // make sure that we never delete the static io stream thing
00073 s->add_ref();
00074 set(0, cxx::ref_ptr(s)); // stdin
00075 set(1, cxx::ref_ptr(s)); // stdout
00076 set(2, cxx::ref_ptr(s)); // stderr
00077 }
00078
00079 class Root_mount_tree : public L4Re::Vfs::Mount_tree
00080 {
00081 public:
00082 Root_mount_tree() : L4Re::Vfs::Mount_tree(0) {}
00083 void operator delete (void *) {}
00084 };
00085
00086 class Vfs : public L4Re::Vfs::Ops
00087 {
00088 private:
00089 bool _early_oom;
00090
00091 public:
00092 Vfs()
00093 : _early_oom(true), _root_mount(), _root(L4Re::Env::env())
00094 {
00095 _root_mount.add_ref();
00096 _root.add_ref();
00097 }

```

```

00097 _root_mount.mount(cxx::ref_ptr(&_root));
00098 _cwd = cxx::ref_ptr(&_root);
00099
00100 #if 0
00101 Ref_ptr<L4Re::Vfs::File> rom;
00102 _root.openat("rom", 0, 0, &rom);
00103
00104 _root_mount.create_tree("lib/foo", rom);
00105
00106 _root.openat("lib", 0, 0, &_cwd);
00107
00108 #endif
00109 }
00110
00111 int alloc_fd(Ref_ptr<L4Re::Vfs::File> const &f) noexcept override;
00112 Ref_ptr<L4Re::Vfs::File> free_fd(int fd) noexcept override;
00113 Ref_ptr<L4Re::Vfs::File> get_root() noexcept override;
00114 Ref_ptr<L4Re::Vfs::File> get_cwd() noexcept override;
00115 void set_cwd(Ref_ptr<L4Re::Vfs::File> const &dir) noexcept override;
00116 Ref_ptr<L4Re::Vfs::File> get_file(int fd) noexcept override;
00117 cxx::Pair<Ref_ptr<L4Re::Vfs::File>, int>
00118 set_fd(int fd, Ref_ptr<L4Re::Vfs::File> const &f = Ref_ptr<>::Nil) noexcept
00119 override;
00120
00121 int mmap2(void *start, size_t len, int prot, int flags, int fd,
00122 off_t offset, void **ptr) noexcept override;
00123
00124 int munmap(void *start, size_t len) noexcept override;
00125 int mremap(void *old, size_t old_sz, size_t new_sz, int flags,
00126 void **new_addr) noexcept override;
00127 int mprotect(const void *a, size_t sz, int prot) noexcept override;
00128 int msync(void *addr, size_t len, int flags) noexcept override;
00129 int madvise(void *addr, size_t len, int advice) noexcept override;
00130
00131 int register_file_system(L4Re::Vfs::File_system *f) noexcept override;
00132 int unregister_file_system(L4Re::Vfs::File_system *f) noexcept override;
00133 L4Re::Vfs::File_system *get_file_system(char const *fstype) noexcept override;
00134 L4Re::Vfs::File_system_list file_system_list() noexcept override;
00135
00136 int register_file_factory(cxx::Ref_ptr<L4Re::Vfs::File_factory> f) noexcept override;
00137 int unregister_file_factory(cxx::Ref_ptr<L4Re::Vfs::File_factory> f) noexcept override;
00138 Ref_ptr<L4Re::Vfs::File_factory> get_file_factory(int proto) noexcept override;
00139 Ref_ptr<L4Re::Vfs::File_factory> get_file_factory(char const *proto_name) noexcept override;
00140 int mount(char const *path, cxx::Ref_ptr<L4Re::Vfs::File> const &dir) noexcept override;
00141
00142 void operator delete (void *) {}
00143
00144 void *malloc(size_t size) noexcept override { return Vfs_config::malloc(size); }
00145 void free(void *m) noexcept override { Vfs_config::free(m); }
00146
00147 private:
00148 Root_mount_tree _root_mount;
00149 L4Re::Core::Env_dir _root;
00150 Ref_ptr<L4Re::Vfs::File> _cwd;
00151 Fd_store fds;
00152
00153 L4Re::Vfs::File_system *_fs_registry;
00154
00155 struct File_factory_item : cxx::H_list_item_t<File_factory_item>
00156 {
00157 cxx::Ref_ptr<L4Re::Vfs::File_factory> f;
00158 explicit File_factory_item(cxx::Ref_ptr<L4Re::Vfs::File_factory> const &f)
00159 : f(f) {};
00160
00161 File_factory_item() = default;
00162 File_factory_item(File_factory_item const &) = delete;
00163 File_factory_item &operator = (File_factory_item const &) = delete;
00164 };
00165
00166 cxx::H_list_t<File_factory_item> _file_factories;
00167
00168 l4_addr_t _anon_offset;
00169 L4Re::Shared_cap<L4Re::Dataspace> _anon_ds;
00170
00171 int alloc_ds(unsigned long size, L4Re::Shared_cap<L4Re::Dataspace> *ds);
00172 int alloc_anon_mem(l4_umword_t size, L4Re::Shared_cap<L4Re::Dataspace> *ds,
00173 l4_addr_t *offset);
00174
00175 void align_mmap_start_and_length(void **start, size_t *length);
00176 int munmap_regions(void *start, size_t len);
00177
00178 L4Re::Vfs::File_system *find_fs_from_type(char const *fstype) noexcept;
00179 };
00180
00181 static inline bool strequal(char const *a, char const *b)
00182 {
00183 for (;*a && *a == *b; ++a, ++b)

```

```

00184 ;
00185 return *a == *b;
00186 }
00187
00188 int
00189 Vfs::register_file_system(L4Re::Vfs::File_system *f) noexcept
00190 {
00191 using L4Re::Vfs::File_system;
00192
00193 if (!f)
00194 return -EINVAL;
00195
00196 for (File_system *c = _fs_registry; c; c = c->next())
00197 if (strequal(c->type(), f->type()))
00198 return -EEXIST;
00199
00200 f->next(_fs_registry);
00201 _fs_registry = f;
00202
00203 return 0;
00204 }
00205
00206 int
00207 Vfs::unregister_file_system(L4Re::Vfs::File_system *f) noexcept
00208 {
00209 using L4Re::Vfs::File_system;
00210
00211 if (!f)
00212 return -EINVAL;
00213
00214 File_system **p = &_fs_registry;
00215
00216 for (; *p; p = &(*p)->next())
00217 if (*p == f)
00218 {
00219 *p = f->next();
00220 f->next() = 0;
00221 return 0;
00222 }
00223
00224 return -ENOENT;
00225 }
00226
00227 L4Re::Vfs::File_system *
00228 Vfs::find_fs_from_type(char const *fstype) noexcept
00229 {
00230 L4Re::Vfs::File_system_list fsl(_fs_registry);
00231 for (L4Re::Vfs::File_system_list::Iterator c = fsl.begin();
00232 c != fsl.end(); ++c)
00233 if (strequal(c->type(), fstype))
00234 return *c;
00235 return 0;
00236 }
00237
00238 L4Re::Vfs::File_system_list
00239 Vfs::file_system_list() noexcept
00240 {
00241 return L4Re::Vfs::File_system_list(_fs_registry);
00242 }
00243
00244 L4Re::Vfs::File_system *
00245 Vfs::get_file_system(char const *fstype) noexcept
00246 {
00247 L4Re::Vfs::File_system *fs;
00248 if ((fs = find_fs_from_type(fstype)))
00249 return fs;
00250
00251 // Try to load a file system module dynamically
00252 int res = Vfs_config::load_module(fstype);
00253 if (res < 0)
00254 return 0;
00255
00256 // Try again
00257 return find_fs_from_type(fstype);
00258 }
00259
00260 int
00261 Vfs::register_file_factory(cxx::Ref_ptr<L4Re::Vfs::File_factory> f) noexcept
00262 {
00263 if (!f)
00264 return -EINVAL;
00265
00266 void *x = this->malloc(sizeof(File_factory_item));
00267 if (!x)
00268 return -ENOMEM;
00269
00270 auto ff = new (x, cxx::Nothrow()) File_factory_item(f);

```

```

00271 _file_factories.push_front(ff);
00272 return 0;
00273 }
00274
00275 int
00276 Vfs::unregister_file_factory(cxx::Ref_ptr<L4Re::Vfs::File_factory> f) noexcept
00277 {
00278 for (auto p: _file_factories)
00279 {
00280 if (p->f == f)
00281 {
00282 _file_factories.remove(p);
00283 p->~File_factory_item();
00284 this->free(p);
00285 return 0;
00286 }
00287 }
00288 return -ENOENT;
00289 }
00290
00291 Ref_ptr<L4Re::Vfs::File_factory>
00292 Vfs::get_file_factory(int proto) noexcept
00293 {
00294 for (auto p: _file_factories)
00295 if (p->f->proto() == proto)
00296 return p->f;
00297
00298 return Ref_ptr<L4Re::Vfs::File_factory>();
00299 }
00300
00301 Ref_ptr<L4Re::Vfs::File_factory>
00302 Vfs::get_file_factory(char const *proto_name) noexcept
00303 {
00304 for (auto p: _file_factories)
00305 {
00306 auto n = p->f->proto_name();
00307 if (n)
00308 {
00309 char const *a = n;
00310 char const *b = proto_name;
00311 for (; *a && *b && *a == *b; ++a, ++b)
00312 ;
00313
00314 if ((*a == 0) && (*b == 0))
00315 return p->f;
00316 }
00317 }
00318
00319 return Ref_ptr<L4Re::Vfs::File_factory>();
00320 }
00321
00322 int
00323 Vfs::alloc_fd(Ref_ptr<L4Re::Vfs::File> const &f) noexcept
00324 {
00325 int fd = fds.alloc();
00326 if (fd < 0)
00327 return -EMFILE;
00328
00329 if (f)
00330 fds.set(fd, f);
00331
00332 return fd;
00333 }
00334
00335 Ref_ptr<L4Re::Vfs::File>
00336 Vfs::free_fd(int fd) noexcept
00337 {
00338 Ref_ptr<L4Re::Vfs::File> f = fds.get(fd);
00339
00340 if (!f)
00341 return Ref_ptr<>::Nil;
00342
00343 fds.free(fd);
00344 return f;
00345 }
00346
00347
00348 Ref_ptr<L4Re::Vfs::File>
00349 Vfs::get_root() noexcept
00350 {
00351 return cxx::ref_ptr(&_root);
00352 }
00353
00354 Ref_ptr<L4Re::Vfs::File>
00355 Vfs::get_cwd() noexcept
00356 {
00357 return _cwd;

```



```

00358 }
00359
00360 void
00361 Vfs::set_cwd(Ref_ptr<L4Re::Vfs::File> const &dir) noexcept
00362 {
00363 // FIXME: check for is dir
00364 if (dir)
00365 _cwd = dir;
00366 }
00367
00368 Ref_ptr<L4Re::Vfs::File>
00369 Vfs::get_file(int fd) noexcept
00370 {
00371 return fds.get(fd);
00372 }
00373
00374 cxx::Pair<Ref_ptr<L4Re::Vfs::File>, int>
00375 Vfs::set_fd(int fd, Ref_ptr<L4Re::Vfs::File> const &f) noexcept
00376 {
00377 if (!fds.check_fd(fd))
00378 return cxx::pair(Ref_ptr<L4Re::Vfs::File>(Ref_ptr<>::Nil), EBADF);
00379
00380 Ref_ptr<L4Re::Vfs::File> old = fds.get(fd);
00381 fds.set(fd, f);
00382 return cxx::pair(old, 0);
00383 }
00384
00385 #define GET_FILE_DBG(fd, err) \
00386 Ref_ptr<L4Re::Vfs::File> fi = fds.get(fd); \
00387 if (!fi) \
00388 { \
00389 return -err; \
00390 }
00391
00392 #define GET_FILE(fd, err) \
00393 Ref_ptr<L4Re::Vfs::File> fi = fds.get(fd); \
00394 if (!fi) \
00395 return -err;
00396
00397 void
00398 Vfs::align_mmap_start_and_length(void **start, size_t *length)
00399 {
00400 l4_addr_t const s = reinterpret_cast<l4_addr_t>(*start);
00401 size_t const o = s & (L4_PAGESIZE - 1);
00402
00403 *start = reinterpret_cast<void*>(l4_trunc_page(s));
00404 *length = l4_round_page(*length + o);
00405 }
00406
00407 int
00408 Vfs::munmap_regions(void *start, size_t len)
00409 {
00410 using namespace L4;
00411 using namespace L4Re;
00412
00413 int err;
00414 Cap<Dataspace> ds;
00415 Cap<Rm> r = Env::env()->rm();
00416
00417 if (l4_addr_t(start) & (L4_PAGESIZE - 1))
00418 return -EINVAL;
00419
00420 align_mmap_start_and_length(&start, &len);
00421
00422 while (1)
00423 {
00424 DEBUG_LOG(debug_mmap, {
00425 outstring("DETACH: start = 0x");
00426 outhex32(l4_addr_t(start));
00427 outstring(" len = 0x");
00428 outhex32(len);
00429 outstring("\n");
00430 });
00431 err = r->detach(l4_addr_t(start), len, &ds, This_task);
00432 if (err < 0)
00433 return err;
00434
00435 switch (err & Rm::Detach_result_mask)
00436 {
00437 case Rm::Split_ds:
00438 if (ds.is_valid())
00439 L4Re::virt_cap_alloc->take(ds);
00440 return 0;
00441 case Rm::Detached_ds:
00442 if (ds.is_valid())
00443 L4Re::virt_cap_alloc->release(ds);
00444

```

```

00445 break;
00446 default:
00447 break;
00448 }
00449
00450 if (!(err & Rm::Detach_again))
00451 return 0;
00452 }
00453 }
00454
00455 int
00456 Vfs::munmap(void *start, size_t len) L4_NOTHROW
00457 {
00458 using namespace L4;
00459 using namespace L4Re;
00460
00461 int err = 0;
00462 Cap<Rm> r = Env::env()->rm();
00463
00464 // Fields for obtaining a list of areas for the calling process
00465 long area_cnt = -1; // No. of areas in this process
00466 Rm::Area const *area_array;
00467 bool matches_area = false; // true if unmap parameters match an area
00468
00469 // First check if there are any areas matching the munmap request. Those
00470 // might have been created by an mmap call using PROT_NONE as protection
00471 // modifier.
00472
00473 area_cnt = r->get_areas((l4_addr_t) start, &area_array);
00474
00475 // It is enough to check for the very first entry, since get_areas will
00476 // only return areas with a starting address equal or greater to <start>.
00477 // However, we intend to unmap at most the area starting exactly at
00478 // <start>.
00479 if (area_cnt > 0)
00480 {
00481 size_t area_size = area_array[0].end - area_array[0].start + 1;
00482
00483 // Only free the area if the munmap parameters describe it exactly.
00484 if (area_array[0].start == (l4_addr_t) start && area_size == len)
00485 {
00486 r->free_area((l4_addr_t) start);
00487 matches_area = true;
00488 }
00489 }
00490
00491 // After clearing possible area reservations from PROT_NONE mappings, clear
00492 // any regions in the address range specified. Note that errors shall be
00493 // suppressed if an area was freed but no regions were found.
00494 err = munmap_regions(start, len);
00495 if (err == -ENOENT && matches_area)
00496 return 0;
00497
00498 return err;
00499 }
00500
00501 int
00502 Vfs::alloc_ds(unsigned long size, L4Re::Shared_cap<L4Re::Dataspace> *ds)
00503 {
00504 *ds = L4Re::make_shared_cap<L4Re::Dataspace>(L4Re::virt_cap_alloc);
00505
00506 if (!ds->is_valid())
00507 return -ENOMEM;
00508
00509 int err;
00510 if ((err = Vfs_config::allocator()->alloc(size, ds->get())) < 0)
00511 return err;
00512
00513 DEBUG_LOG(debug_mmap, {
00514 outstring("ANON DS ALLOCATED: size=");
00515 outhex32(size);
00516 outstring(" cap = 0x");
00517 outhex32(ds->cap());
00518 outstring("\n");
00519 });
00520
00521 return 0;
00522 }
00523
00524 int
00525 Vfs::alloc_anon_mem(l4_umword_t size, L4Re::Shared_cap<L4Re::Dataspace> *ds,
00526 l4_addr_t *offset)
00527 {
00528 #if !defined(CONFIG_MMU)
00529 // Small values for !MMU systems. These platforms do not have much memory
00530 // typically and the memory must be instantly allocated.
00531 enum

```

```

00532 {
00533 ANON_MEM_DS_POOL_SIZE = 256UL « 10, // size of a pool dataspace used for anon memory
00534 ANON_MEM_MAX_SIZE = 32UL « 10, // chunk size that will be allocate a dataspace
00535 };
00536 #elif defined(USE_BIG_ANON_DS)
00537 enum
00538 {
00539 ANON_MEM_DS_POOL_SIZE = 256UL « 20, // size of a pool dataspace used for anon memory
00540 ANON_MEM_MAX_SIZE = 32UL « 20, // chunk size that will be allocate a dataspace
00541 };
00542 #else
00543 enum
00544 {
00545 ANON_MEM_DS_POOL_SIZE = 256UL « 20, // size of a pool dataspace used for anon memory
00546 ANON_MEM_MAX_SIZE = 0UL « 20, // chunk size that will be allocate a dataspace
00547 };
00548 #endif
00549
00550 if (size >= ANON_MEM_MAX_SIZE)
00551 {
00552 int err;
00553 if ((err = alloc_ds(size, ds)) < 0)
00554 return err;
00555
00556 *offset = 0;
00557
00558 if (!_early_oom)
00559 return err;
00560
00561 return (*ds)->allocate(0, size);
00562 }
00563
00564 if (!_anon_ds.is_valid() || _anon_offset + size >= ANON_MEM_DS_POOL_SIZE)
00565 {
00566 int err;
00567 if ((err = alloc_ds(ANON_MEM_DS_POOL_SIZE, ds)) < 0)
00568 return err;
00569
00570 _anon_offset = 0;
00571 _anon_ds = *ds;
00572 }
00573 else
00574 *ds = _anon_ds;
00575
00576 if (_early_oom)
00577 {
00578 if (int err = (*ds)->allocate(_anon_offset, size))
00579 return err;
00580 }
00581
00582 *offset = _anon_offset;
00583 _anon_offset += size;
00584 return 0;
00585 }
00586
00587 int
00588 Vfs::mmap2(void *start, size_t len, int prot, int flags, int fd, off_t page4k_offset,
00589 void **resptr) L4_NOTHROW
00590 {
00591 DEBUG_LOG(debug_mmap, {
00592 outstring("MMAP params: ");
00593 outstring("start = 0x");
00594 outhex32(l4_addr_t(start));
00595 outstring(", len = 0x");
00596 outhex32(len);
00597 outstring(", prot = 0x");
00598 outhex32(prot);
00599 outstring(", flags = 0x");
00600 outhex32(flags);
00601 outstring(", offset = 0x");
00602 outhex32(page4k_offset);
00603 outstring("\n");
00604 });
00605
00606 using namespace L4Re;
00607 off64_t offset = l4_trunc_page(page4k_offset « 12);
00608
00609 if (flags & MAP_FIXED)
00610 if (l4_addr_t(start) & (L4_PAGESIZE - 1))
00611 return -EINVAL;
00612
00613 align_mmap_start_and_length(&start, &len);
00614
00615 // special code to just reserve an area of the virtual address space
00616 // Same behavior should be exposed when mapping with PROT_NONE. Mind that
00617 // PROT_NONE can only be specified exclusively, since it is defined to 0x0.
00618 if ((flags & 0x1000000) || (prot == PROT_NONE))

```

```

00619 {
00620 int err;
00621 L4::Cap<Rm> r = Env::env()->rm();
00622 L4_addr_t area = reinterpret_cast<L4_addr_t>(start);
00623 err = r->reserve_area(&area, len, L4Re::Rm::F::Search_addr);
00624 if (err < 0)
00625 return err;
00626
00627 *resp_ptr = reinterpret_cast<void*>(area);
00628
00629 DEBUG_LOG(debug_mmap, {
00630 outstring(" MMAP reserved area: 0x");
00631 outhex32(area);
00632 outstring(" length= 0x");
00633 outhex32(len);
00634 outstring("\n");
00635 });
00636
00637 return 0;
00638 }
00639
00640 L4Re::Shared_cap<L4Re::Dataspace> ds;
00641 L4_addr_t anon_offset = 0;
00642 L4Re::Rm::Flags rm_flags(0);
00643
00644 if (flags & (MAP_ANONYMOUS | MAP_PRIVATE))
00645 {
00646 rm_flags |= L4Re::Rm::F::Detach_free;
00647
00648 int err = alloc_anon_mem(len, &ds, &anon_offset);
00649 if (err)
00650 return err;
00651
00652 DEBUG_LOG(debug_mmap, {
00653 outstring(" USE ANON MEM: 0x");
00654 outhex32(ds.cap());
00655 outstring(" offs = 0x");
00656 outhex32(anon_offset);
00657 outstring("\n");
00658 });
00659 }
00660
00661 char const *region_name = "[unknown]";
00662 L4_addr_t file_offset = 0;
00663 if (!(flags & MAP_ANONYMOUS))
00664 {
00665 Ref_ptr<L4Re::Vfs::File> fi = fds.get(fd);
00666 if (!fi)
00667 return -EBADF;
00668
00669 region_name = fi->path();
00670
00671 L4::Cap<L4Re::Dataspace> fds = fi->data_space();
00672
00673 if (!fds.is_valid())
00674 return -EINVAL;
00675
00676 if (len + offset > L4_round_page(fds->size()))
00677 return -EINVAL;
00678
00679 if (flags & MAP_PRIVATE)
00680 {
00681 DEBUG_LOG(debug_mmap, outstring("COW\n"));
00682 int err = ds->copy_in(anon_offset, fds, offset, len);
00683 file_offset = offset;
00684 if (err == -L4_EINVAL)
00685 {
00686 L4::Cap<Rm> r = Env::env()->rm();
00687 Rm::Unique_region<char*> src;
00688 Rm::Unique_region<char*> dst;
00689 err = r->attach(&src, len,
00690 L4Re::Rm::F::Search_addr | L4Re::Rm::F::R,
00691 fds, offset);
00692 if (err < 0)
00693 return err;
00694
00695 err = r->attach(&dst, len,
00696 L4Re::Rm::F::Search_addr | L4Re::Rm::F::RW,
00697 ds.get(), anon_offset);
00698 if (err < 0)
00699 return err;
00700
00701 memcpy(dst.get(), src.get(), len);
00702
00703 region_name = "[mmap-private]";
00704 file_offset = (unsigned long)dst.get();
00705 }

```

```

00706 else if (err)
00707 return err;
00708
00709 offset = anon_offset;
00710 }
00711 else
00712 {
00713 L4Re::virt_cap_alloc->take(fds);
00714 ds = L4Re::Shared_cap<L4Re::Dataspace>(fds, L4Re::virt_cap_alloc);
00715 }
00716 }
00717 else
00718 {
00719 offset = anon_offset;
00720 region_name = "[anon]";
00721 file_offset = offset;
00722 }
00723
00724
00725 if (!(flags & MAP_FIXED) && start == 0)
00726 start = reinterpret_cast<void*>(L4_PAGESIZE);
00727
00728 char *data = static_cast<char *>(start);
00729 L4::Cap<Rm> r = Env::env()->rm();
00730 l4_addr_t overmap_area = L4_INVALID_ADDR;
00731
00732 int err;
00733 if (flags & MAP_FIXED)
00734 {
00735 overmap_area = l4_addr_t(start);
00736
00737 err = r->reserve_area(&overmap_area, len);
00738 if (err < 0)
00739 overmap_area = L4_INVALID_ADDR;
00740
00741 rm_flags |= Rm::F::In_area;
00742
00743 // Make sure to remove old mappings residing at the respective address
00744 // range. If none exists, we are fine as well, allowing us to ignore
00745 // ENOENT here.
00746 err = munmap_regions(start, len);
00747 if (err && err != -ENOENT)
00748 return err;
00749 }
00750
00751 if (!(flags & MAP_FIXED))
00752 rm_flags |= Rm::F::Search_addr;
00753 if (prot & PROT_READ)
00754 rm_flags |= Rm::F::R;
00755 if (prot & PROT_WRITE)
00756 rm_flags |= Rm::F::W;
00757 if (prot & PROT_EXEC)
00758 rm_flags |= Rm::F::X;
00759
00760 err = r->attach(&data, len, rm_flags,
00761 L4::Ipc::make_cap(ds.get(), (prot & PROT_WRITE)
00762 ? L4_CAP_FPAGE_RW
00763 : L4_CAP_FPAGE_RO),
00764 offset, L4_PAGESHIFT, L4::Cap<L4::Task>::Invalid,
00765 region_name, file_offset);
00766
00767 DEBUG_LOG(debug_mmap, {
00768 outstring(" MAPPED: 0x");
00769 outhex32(ds.cap());
00770 outstring(" addr: 0x");
00771 outhex32(l4_addr_t(data));
00772 outstring(" bytes: 0x");
00773 outhex32(len);
00774 outstring(" offset: 0x");
00775 outhex32(offset);
00776 outstring(" err = ");
00777 outdec(err);
00778 outstring("\n");
00779 });
00780
00781
00782 if (overmap_area != L4_INVALID_ADDR)
00783 r->free_area(overmap_area);
00784
00785 if (err < 0)
00786 return err;
00787
00788 l4_assert (!(start && !data));
00789
00790 // release ownership of the attached DS
00791 ds.release();
00792 *resptr = data;

```

```

00793
00794 return 0;
00795 }
00796
00797 namespace {
00798 class Auto_area
00799 {
00800 public:
00801 L4::Cap<L4Re::Rm> r;
00802 l4_addr_t a;
00803
00804 explicit Auto_area(L4::Cap<L4Re::Rm> r, l4_addr_t a = L4_INVALID_ADDR)
00805 : r(r), a(a) {}
00806
00807 int reserve(l4_addr_t _a, l4_size_t sz, L4Re::Rm::Flags flags)
00808 {
00809 free();
00810 a = _a;
00811 int e = r->reserve_area(&a, sz, flags);
00812 if (e)
00813 a = L4_INVALID_ADDR;
00814 return e;
00815 }
00816
00817 void free()
00818 {
00819 if (is_valid())
00820 {
00821 r->free_area(a);
00822 a = L4_INVALID_ADDR;
00823 }
00824 }
00825
00826 bool is_valid() const { return a != L4_INVALID_ADDR; }
00827
00828 ~Auto_area() { free(); }
00829 };
00830 }
00831
00832 int
00833 Vfs::mremap(void *old_addr, size_t old_size, size_t new_size, int flags,
00834 void **new_addr) L4_NOTHROW
00835 {
00836 using namespace L4Re;
00837
00838 DEBUG_LOG(debug_mmap, {
00839 outstring("Mremap: addr = 0x");
00840 outhex32((l4_umword_t)old_addr);
00841 outstring(" old_size = 0x");
00842 outhex32(old_size);
00843 outstring(" new_size = 0x");
00844 outhex32(new_size);
00845 outstring("\n");
00846 });
00847
00848 if (flags & MREMAP_FIXED && !(flags & MREMAP_MAYMOVE))
00849 return -EINVAL;
00850
00851 l4_addr_t oa = l4_trunc_page(reinterpret_cast<l4_addr_t>(old_addr));
00852 if (oa != reinterpret_cast<l4_addr_t>(old_addr))
00853 return -EINVAL;
00854
00855 bool const fixed = flags & MREMAP_FIXED;
00856 bool const maymove = flags & MREMAP_MAYMOVE;
00857
00858 L4::Cap<Rm> r = Env::env()->rm();
00859
00860 // sanitize input parameters to multiples of pages
00861 old_size = l4_round_page(old_size);
00862 new_size = l4_round_page(new_size);
00863
00864 if (!fixed)
00865 {
00866 if (new_size < old_size)
00867 {
00868 *new_addr = old_addr;
00869 return munmap(reinterpret_cast<void*>(oa + new_size),
00870 old_size - new_size);
00871 }
00872
00873 if (new_size == old_size)
00874 {
00875 *new_addr = old_addr;
00876 return 0;
00877 }
00878 }
00879

```

```

00880 Auto_area old_area(r);
00881 int err = old_area.reserve(oa, old_size, L4Re::Rm::Flags(0));
00882 if (err < 0)
00883 return -EINVAL;
00884
00885 l4_addr_t pad_addr;
00886 Auto_area new_area(r);
00887 if (fixed)
00888 {
00889 l4_addr_t na = l4_trunc_page(reinterpret_cast<l4_addr_t>(*new_addr));
00890 if (na != reinterpret_cast<l4_addr_t>(*new_addr))
00891 return -EINVAL;
00892
00893 // check if the current virtual memory area can be expanded
00894 int err = new_area.reserve(na, new_size, L4Re::Rm::Flags(0));
00895 if (err < 0)
00896 return err;
00897
00898 pad_addr = na;
00899 // unmap all stuff and remap ours
00900 }
00901 else
00902 {
00903 l4_addr_t ta = oa + old_size;
00904 unsigned long ts = new_size - old_size;
00905 // check if the current virtual memory area can be expanded
00906 long err = new_area.reserve(ta, ts, L4Re::Rm::Flags(0));
00907 if (!maymove && err)
00908 return -ENOMEM;
00909
00910 L4Re::Rm::Offset toffs;
00911 L4Re::Rm::Flags tflags;
00912 L4::Cap<L4Re::Dataspace> tds;
00913
00914 err = r->find(&ta, &ts, &toffs, &tflags, &tds);
00915
00916 // there is enough space to expand the mapping in place
00917 if (err == -ENOENT || (err == 0 && (tflags & Rm::F::In_area)))
00918 {
00919 old_area.free(); // pad at the original address
00920 pad_addr = oa + old_size;
00921 *new_addr = old_addr;
00922 }
00923 else if (!maymove)
00924 return -ENOMEM;
00925 else
00926 {
00927 // search for a new area to remap
00928 err = new_area.reserve(0, new_size, Rm::F::Search_addr);
00929 if (err < 0)
00930 return -ENOMEM;
00931
00932 pad_addr = new_area.a + old_size;
00933 *new_addr = reinterpret_cast<void *>(new_area.a);
00934 }
00935 }
00936
00937 if (old_area.is_valid())
00938 {
00939 unsigned long size = old_size;
00940
00941 l4_addr_t a = old_area.a;
00942 unsigned long s = 1;
00943 L4Re::Rm::Offset o;
00944 L4Re::Rm::Flags f;
00945 L4::Cap<L4Re::Dataspace> ds;
00946
00947 while (r->find(&a, &s, &o, &f, &ds) >= 0 && !(f & Rm::F::In_area))
00948 {
00949 if (a < old_area.a)
00950 {
00951 auto d = old_area.a - a;
00952 a = old_area.a;
00953 s -= d;
00954 o += d;
00955 }
00956
00957 if (a + s > old_area.a + old_size)
00958 s = old_area.a + old_size - a;
00959
00960 l4_addr_t x = a - old_area.a + new_area.a;
00961
00962 int err = r->attach(&x, s, Rm::F::In_area | f,
00963 L4::Ipc::make_cap(ds, f.cap_rights()), o);
00964 if (err < 0)
00965 return err;
00966

```

```

00967 // count the new attached ds reference
00968 L4Re::virt_cap_alloc->take(ds);
00969
00970 err = r->detach(a, s, &ds, This_task,
00971 Rm::Detach_exact | Rm::Detach_keep);
00972 if (err < 0)
00973 return err;
00974
00975 switch (err & Rm::Detach_result_mask)
00976 {
00977 case Rm::Split_ds:
00978 // add a reference as we split up a mapping
00979 if (ds.is_valid())
00980 L4Re::virt_cap_alloc->take(ds);
00981 break;
00982 case Rm::Detached_ds:
00983 if (ds.is_valid())
00984 L4Re::virt_cap_alloc->release(ds);
00985 break;
00986 default:
00987 break;
00988 }
00989
00990 if (size <= s)
00991 break;
00992 a += s;
00993 size -= s;
00994 s = 1;
00995 }
00996
00997 old_area.free();
00998 }
00999
01000 if (old_size < new_size)
01001 {
01002 l4_addr_t const pad_sz = new_size - old_size;
01003 l4_addr_t toffs;
01004 L4Re::Shared_cap<L4Re::Dataspace> tds;
01005 int err = alloc_anon_mem(pad_sz, &tds, &toffs);
01006 if (err)
01007 return err;
01008
01009 // FIXME: must get the protection rights from the old
01010 // mapping and use the same here, for now just use RWX
01011 err = r->attach(&pad_addr, pad_sz,
01012 Rm::F::In_area | Rm::F::Detach_free | Rm::F::RWX,
01013 L4::Ipc::make_cap_rw(tds.get()), toffs);
01014 if (err < 0)
01015 return err;
01016
01017 // release ownership of tds, the region map is now the new owner
01018 tds.release();
01019 }
01020
01021 return 0;
01022 }
01023
01024 int
01025 Vfs::mprotect(const void * /* a */, size_t /* sz */, int prot) L4_NOTHROW
01026 {
01027 return (prot & PROT_WRITE) ? -1 : 0;
01028 }
01029
01030 int
01031 Vfs::msync(void *, size_t, int) L4_NOTHROW
01032 { return 0; }
01033
01034 int
01035 Vfs::madvise(void *, size_t, int) L4_NOTHROW
01036 { return 0; }
01037
01038 }
01039
01040 L4Re::Vfs::Ops *__rtld_l4re_env_posix_vfs_ops;
01041 extern void *l4re_env_posix_vfs_ops __attribute__((alias("__rtld_l4re_env_posix_vfs_ops"),
01042 visibility("default"))));
01043
01044 namespace {
01045 class Real_mount_tree : public L4Re::Vfs::Mount_tree
01046 {
01047 public:
01048 explicit Real_mount_tree(char *n) : Mount_tree(n) {}
01049
01050 void *operator new (size_t size)
01051 { return __rtld_l4re_env_posix_vfs_ops->malloc(size); }
01052
01053 void operator delete (void *mem)

```



```

01053 { __rtld_l4re_env_posix_vfs_ops->free(mem); }
01054 };
01055 }
01056
01057 int
01058 Vfs::mount(char const *path, cxx::Ref_ptr<L4Re::Vfs::File> const &dir) noexcept
01059 {
01060 using L4Re::Vfs::File;
01061 using L4Re::Vfs::Mount_tree;
01062 using L4Re::Vfs::Path;
01063
01064 cxx::Ref_ptr<Mount_tree> root = get_root()->mount_tree();
01065 if (!root)
01066 return -EINVAL;
01067
01068 cxx::Ref_ptr<Mount_tree> base;
01069 Path p = root->lookup(Path(path), &base);
01070
01071 while (!p.empty())
01072 {
01073 Path f = p.strip_first();
01074
01075 if (f.empty())
01076 return -EEXIST;
01077
01078 char *name = __rtld_l4re_env_posix_vfs_ops->strndup(f.path(), f.length());
01079 if (!name)
01080 return -ENOMEM;
01081
01082 auto nt = cxx::make_ref_obj<Real_mount_tree>(name);
01083 if (!nt)
01084 {
01085 __rtld_l4re_env_posix_vfs_ops->free(name);
01086 return -ENOMEM;
01087 }
01088
01089 base->add_child_node(nt);
01090 base = nt;
01091
01092 if (p.empty())
01093 {
01094 nt->mount(dir);
01095 return 0;
01096 }
01097 }
01098
01099 return -EINVAL;
01100 }
01101
01102 #undef DEBUG_LOG
01103 #undef GET_FILE_DBG
01104 #undef GET_FILE

```

## 17.248 vfs.h

```

00001 /*
00002 * (c) 2010 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include <14/sys/compiler.h>
00011
00012 #include <unistd.h>
00013 #include <stdarg.h>
00014 #include <fcntl.h>
00015 #include <sys/stat.h>
00016 #include <sys/time.h>
00017 #include <sys/mman.h>
00018 #include <sys/socket.h>
00019 #include <utime.h>
00020 #include <errno.h>
00021
00022 #ifndef AT_FDCWD
00023 # define AT_FDCWD -100
00024 #endif
00025
00026 #ifdef __cplusplus
00027 #include <14/sys/capability>

```

```

00029 #include <l4/re/cap_alloc>
00030 #include <l4/re/dataspace>
00031 #include <l4/cxx/pair>
00032 #include <l4/cxx/ref_ptr>
00033
00034 namespace L4Re {
00038 namespace Vfs {
00039
00040 class Mount_tree;
00041 class File;
00042
00052 class Generic_file
00053 {
00054 public:
00060 enum Ready_type : unsigned
00061 {
00062 Read = 0,
00063 Write,
00064 Exception
00065 };
00066
00067 virtual ~Generic_file() noexcept = 0;
00068
00080 virtual int unlock_all_locks() noexcept = 0;
00081
00090 virtual int fstat(struct stat64 *buf) const noexcept = 0;
00091
00097 virtual int fchmod(mode_t) noexcept = 0;
00098
00108 virtual int get_status_flags() const noexcept = 0;
00109
00125 virtual int set_status_flags(long flags) noexcept = 0;
00126
00127 virtual int utime(const struct utimbuf *) noexcept = 0;
00128 virtual int utimes(const struct timeval [2]) noexcept = 0;
00129 virtual ssize_t readlink(char *, size_t) = 0;
00130
00146 virtual bool check_ready(Ready_type rt) noexcept = 0;
00147 };
00148
00149 inline
00150 Generic_file::~Generic_file() noexcept
00151 {}
00152
00160 class Directory
00161 {
00162 public:
00163 virtual ~Directory() noexcept = 0;
00164
00178 virtual int faccessat(const char *path, int mode, int flags) noexcept = 0;
00179
00192 virtual int mkdir(const char *path, mode_t mode) noexcept = 0;
00193
00204 virtual int unlink(const char *path) noexcept = 0;
00205
00219 virtual int rename(const char *src_path, const char *dst_path) noexcept = 0;
00220
00234 virtual int link(const char *src_path, const char *dst_path) noexcept = 0;
00235
00248 virtual int symlink(const char *src_path, const char *dst_path) noexcept = 0;
00249
00260 virtual int rmdir(const char *path) noexcept = 0;
00261 virtual int openat(const char *path, int flags, mode_t mode,
00262 cxx::Ref_ptr<File> *f) noexcept = 0;
00263
00264 virtual ssize_t getdents(char *buf, size_t sizebytes) noexcept = 0;
00265
00266 virtual int fchmodat(const char *pathname,
00267 mode_t mode, int flags) noexcept = 0;
00268
00269 virtual int utimensat(const char *pathname,
00270 const struct timespec times[2], int flags) noexcept = 0;
00271
00275 virtual int get_entry(const char *, int, mode_t, cxx::Ref_ptr<File> *) noexcept = 0;
00276 };
00277
00278 inline
00279 Directory::~Directory() noexcept
00280 {}
00281
00287 class Regular_file
00288 {
00289 public:
00290 virtual ~Regular_file() noexcept = 0;
00291
00302 virtual L4::Cap<L4Re::Dataspace> data_space() noexcept = 0;
00303

```

```

00313 virtual ssize_t readv(const struct iovec*, int iovcnt) noexcept = 0;
00314
00325 virtual ssize_t writev(const struct iovec*, int iovcnt) noexcept = 0;
00326
00327 virtual ssize_t preadv(const struct iovec *iov, int iovcnt, off64_t offset) noexcept = 0;
00328 virtual ssize_t pwritev(const struct iovec *iov, int iovcnt, off64_t offset) noexcept = 0;
00329
00337 virtual off64_t lseek(off64_t, int) noexcept = 0;
00338
00339
00347 virtual int ftruncate(off64_t pos) noexcept = 0;
00348
00354 virtual int fsync() const noexcept = 0;
00355
00361 virtual int fdatasync() const noexcept = 0;
00362
00372 virtual int get_lock(struct flock64 *lock) noexcept = 0;
00373
00382 virtual int set_lock(struct flock64 *lock, bool wait) noexcept = 0;
00383 };
00384
00385 inline
00386 Regular_file::~Regular_file() noexcept
00387 {}
00388
00389 class Socket
00390 {
00391 public:
00392 virtual ~Socket() noexcept = 0;
00393 virtual int bind(sockaddr const *, socklen_t) noexcept = 0;
00394 virtual int connect(sockaddr const *, socklen_t) noexcept = 0;
00395 virtual ssize_t send(void const *, size_t, int) noexcept = 0;
00396 virtual ssize_t recv(void *, size_t, int) noexcept = 0;
00397 virtual ssize_t sendto(void const *, size_t, int, sockaddr const *, socklen_t) noexcept = 0;
00398 virtual ssize_t recvfrom(void *, size_t, int, sockaddr *, socklen_t *) noexcept = 0;
00399 virtual ssize_t sendmsg(msghdr const *, int) noexcept = 0;
00400 virtual ssize_t recvmsg(msghdr *, int) noexcept = 0;
00401 virtual int getsockopt(int level, int opt, void *, socklen_t *) noexcept = 0;
00402 virtual int setsockopt(int level, int opt, void const *, socklen_t) noexcept = 0;
00403 virtual int listen(int) noexcept = 0;
00404 virtual int accept(sockaddr *addr, socklen_t *) noexcept = 0;
00405 virtual int shutdown(int) noexcept = 0;
00406
00407 virtual int getsockname(sockaddr *, socklen_t *) noexcept = 0;
00408 virtual int getpeername(sockaddr *, socklen_t *) noexcept = 0;
00409 };
00410
00411 inline
00412 Socket::~Socket() noexcept
00413 {}
00414
00420 class Special_file
00421 {
00422 public:
00423 virtual ~Special_file() noexcept = 0;
00424
00435 virtual int ioctl(unsigned long cmd, va_list args) noexcept = 0;
00436 };
00437
00438 inline
00439 Special_file::~Special_file() noexcept
00440 {}
00441
00455 class File :
00456 public Generic_file,
00457 public Regular_file,
00458 public Directory,
00459 public Special_file,
00460 public Socket
00461 {
00462 friend class Mount_tree;
00463
00464 private:
00465 void operator = (File const &);
00466
00467 protected:
00468 File() noexcept : _ref_cnt(0) {}
00469 File(File const &)
00470 : Generic_file(), Regular_file(), Directory(), Special_file(), _ref_cnt(0)
00471 {}
00472
00473 public:
00474
00475 const char *get_mount(const char *path, cxx::Ref_ptr<File> *dir,
00476 cxx::Ref_ptr<Mount_tree> *mt = 0) noexcept;
00477
00478 int openat(const char *path, int flags, mode_t mode,

```

```

00479 cxx::Ref_ptr<File> *f) noexcept override;
00480
00481 void add_ref() noexcept { ++_ref_cnt; }
00482 int remove_ref() noexcept { return --_ref_cnt; }
00483
00484 virtual ~File() noexcept = 0;
00485
00486 cxx::Ref_ptr<Mount_tree> mount_tree() const noexcept
00487 { return _mount_tree; }
00488
00489 char const *path() const noexcept { return _path; }
00490
00491 private:
00492 int _ref_cnt;
00493 cxx::Ref_ptr<Mount_tree> _mount_tree;
00494 char _path[80] = "";
00495 };
00496
00497 inline
00498 File::~File() noexcept
00499 {}
00500
00501 class Path
00502 {
00503 private:
00504 char const *_p;
00505 unsigned _l;
00506
00507 public:
00508 Path() noexcept : _p(0), _l(0) {}
00509
00510 explicit Path(char const *p) noexcept : _p(p)
00511 { for (_l = 0; *p; ++p, ++_l) ; }
00512
00513 Path(char const *p, unsigned l) noexcept : _p(p), _l(l)
00514 {}
00515
00516 static bool __is_sep(char s) noexcept;
00517
00518 Path cmp_path(char const *prefix) const noexcept;
00519
00520 struct Invalid_ptr;
00521 operator Invalid_ptr const * () const
00522 { return reinterpret_cast<Invalid_ptr const *>(_p); }
00523
00524 unsigned length() const { return _l; }
00525 char const *path() const { return _p; }
00526
00527 bool empty() const { return _l == 0; }
00528
00529 bool is_sep(unsigned offset) const { return __is_sep(_p[offset]); }
00530
00531 bool strip_sep()
00532 {
00533 bool s = false;
00534 for (; __is_sep(_p) && _l; ++_p, --_l)
00535 s = true;
00536 return s;
00537 }
00538
00539 Path first() const
00540 {
00541 unsigned i;
00542 for (i = 0; i < _l && !is_sep(i); ++i)
00543 ;
00544 return Path(_p, i);
00545 }
00546
00547 Path strip_first()
00548 {
00549 Path r = first();
00550 _p += r.length();
00551 _l -= r.length();
00552 strip_sep();
00553 return r;
00554 }
00555 };
00556
00557 };
00558
00559 class Mount_tree
00560 {
00561 public:
00562 explicit Mount_tree(char *n) noexcept;
00571

```

```

00572 Path lookup(Path const &path, cxx::Ref_ptr<Mount_tree> *mt,
00573 cxx::Ref_ptr<Mount_tree> *mp = 0) noexcept;
00574
00575 Path find(Path const &p, cxx::Ref_ptr<Mount_tree> *t) noexcept;
00576
00577 cxx::Ref_ptr<File> mount() const
00578 { return _mount; }
00579
00580 void mount(cxx::Ref_ptr<File> const &m)
00581 {
00582 m->_mount_tree = cxx::ref_ptr(this);
00583 _mount = m;
00584 }
00585
00586 static int create_tree(cxx::Ref_ptr<Mount_tree> const &root,
00587 char const *path,
00588 cxx::Ref_ptr<File> const &dir) noexcept;
00589
00590 void add_child_node(cxx::Ref_ptr<Mount_tree> const &cld);
00591
00592 virtual ~Mount_tree() noexcept = 0;
00593
00594 void add_ref() noexcept { ++_ref_cnt; }
00595 int remove_ref() noexcept { return --_ref_cnt; }
00596
00597 private:
00598 friend class Real_mount_tree;
00599
00600 int _ref_cnt;
00601 char *_name;
00602 cxx::Ref_ptr<Mount_tree> _cld;
00603 cxx::Ref_ptr<Mount_tree> _sib;
00604 cxx::Ref_ptr<File> _mount;
00605 };
00606
00607 inline
00608 Mount_tree::~Mount_tree() noexcept
00609 {}
00610
00611 inline bool
00612 Path::__is_sep(char s) noexcept
00613 { return s == '/'; }
00614
00615 inline Path
00616 Path::cmp_path(char const *n) const noexcept
00617 {
00618 char const *p = _p;
00619 for (; *p && !__is_sep(*p) && *n; ++p, ++n)
00620 if (*p != *n)
00621 return Path();
00622 if (*n || (*p && !__is_sep(*p)))
00623 return Path();
00624 return Path(p, _l - (p - _p));
00625 }
00626
00627 inline
00628 Mount_tree::Mount_tree(char *n) noexcept
00629 : _ref_cnt(0), _name(n)
00630 {}
00631
00632 inline Path
00633 Mount_tree::find(Path const &p, cxx::Ref_ptr<Mount_tree> *t) noexcept
00634 {
00635 if (!_cld)
00636 return Path();
00637 for (cxx::Ref_ptr<Mount_tree> x = _cld; x; x = x->_sib)
00638 {
00639 Path const r = p.cmp_path(x->_name);
00640 if (r)
00641 {
00642 *t = x;
00643 return r;
00644 }
00645 }
00646 return Path();
00647 }
00648
00649 inline Path
00650 Mount_tree::lookup(Path const &path, cxx::Ref_ptr<Mount_tree> *mt,
00651 cxx::Ref_ptr<Mount_tree> *mp) noexcept
00652 {
00653 cxx::Ref_ptr<Mount_tree> x(this);
00654 Path p = path;

```

```

00659
00660 if (p.first().cmp_path("."))
00661 p.strip_first();
00662 Path last_mp = p;
00663
00664 if (mp)
00665 *mp = x;;
00666
00667 while (1)
00668 {
00669 Path r = x->find(p, &x);
00670
00671 if (!r)
00672 {
00673 if (mp)
00674 return last_mp;
00675
00676 if (mt)
00677 *mt = x;
00678
00679 return p;
00680 }
00681
00682 r.strip_sep();
00683
00684 if (mp && x->_mount)
00685 {
00686 last_mp = r;
00687 *mp = x;
00688 }
00689
00690 if (r.empty())
00691 {
00692 if (mt)
00693 *mt = x;
00694
00695 if (mp)
00696 return last_mp;
00697 else
00698 return r;
00699 }
00700
00701 p = r;
00702 }
00703 }
00704
00705 inline
00706 void
00707 Mount_tree::add_child_node(cxx::Ref_ptr<Mount_tree> const &cld)
00708 {
00709 cld->_sib = _cld;
00710 _cld = cld;
00711 }
00712
00713 inline
00714 const char *
00715 File::get_mount(const char *path, cxx::Ref_ptr<File> *dir,
00716 cxx::Ref_ptr<Mount_tree> *mt) noexcept
00717 {
00718 if (!_mount_tree)
00719 {
00720 *dir = cxx::ref_ptr(this);
00721 return path;
00722 }
00723
00724 cxx::Ref_ptr<Mount_tree> mp;
00725 Path p = _mount_tree->lookup(Path(path), mt, &mp);
00726 if (mp->mount())
00727 {
00728 *dir = mp->mount();
00729 return p.path();
00730 }
00731 else
00732 {
00733 *dir = cxx::ref_ptr(this);
00734 return path;
00735 }
00736 }
00737
00738 inline int
00739 File::openat(const char *path, int flags, mode_t mode,
00740 cxx::Ref_ptr<File> *f) noexcept
00741 {
00742 cxx::Ref_ptr<File> dir;
00743 cxx::Ref_ptr<Mount_tree> mt;
00744 char const *m_path = get_mount(path, &dir, &mt);

```

```

00746
00747 int res = dir->get_entry(m_path, flags, mode, f);
00748
00749 if (res < 0)
00750 return res;
00751
00752 if (!(*f)->_mount_tree && mt)
00753 (*f)->_mount_tree = mt;
00754
00755 // Debugging {
00756 size_t i = 0;
00757 for (; i < sizeof((*f)->_path) - 1 && path[i]; ++i)
00758 (*f)->_path[i] = path[i];
00759 (*f)->_path[i] = '\0';
00760 // } Debugging
00761
00762 return res;
00763 }
00764
00773 class Mman
00774 {
00775 public:
00777 virtual int mmap2(void *start, size_t len, int prot, int flags, int fd,
00778 off_t offset, void **ptr) noexcept = 0;
00779
00781 virtual int munmap(void *start, size_t len) noexcept = 0;
00782
00784 virtual int mremap(void *old, size_t old_sz, size_t new_sz, int flags,
00785 void **new_addr) noexcept = 0;
00786
00788 virtual int mprotect(const void *a, size_t sz, int prot) noexcept = 0;
00789
00791 virtual int msync(void *addr, size_t len, int flags) noexcept = 0;
00792
00794 virtual int madvise(void *addr, size_t len, int advice) noexcept = 0;
00795
00796 virtual ~Mman() noexcept = 0;
00797 };
00798
00799 inline
00800 Mman::~Mman() noexcept {}
00801
00802 class File_factory
00803 {
00804 private:
00805 int _ref_cnt = 0;
00806 int _proto = 0;
00807 char const *_proto_name = 0;
00808
00809 template<typename T> friend struct cxx::Default_ref_counter;
00810 void add_ref() noexcept { ++_ref_cnt; }
00811 int remove_ref() noexcept { return --_ref_cnt; }
00812
00813 public:
00814 explicit File_factory(int proto) : _proto(proto) {}
00815 explicit File_factory(char const *proto_name) : _proto_name(proto_name) {}
00816 File_factory(int proto, char const *proto_name)
00817 : _proto(proto), _proto_name(proto_name)
00818 {}
00819
00820 File_factory(File_factory const &) = delete;
00821 File_factory &operator = (File_factory const &) = delete;
00822
00823 char const *proto_name() const { return _proto_name; }
00824 int proto() const { return _proto; }
00825
00826 virtual ~File_factory() noexcept = 0;
00827 virtual cxx::Ref_ptr<File> create(L4::Cap<void> file) = 0;
00828 };
00829
00830 inline File_factory::~File_factory() noexcept {}
00831
00832 template<typename IFACE, typename IMPL>
00833 class File_factory_t : public File_factory
00834 {
00835 public:
00836 File_factory_t()
00837 : File_factory(IFACE::Protocol, L4::kobject_typeid<IFACE>()->name())
00838 {}
00839
00840 cxx::Ref_ptr<File> create(L4::Cap<void> file) override
00841 { return cxx::make_ref_obj<IMPL>(L4::cap_cast<IFACE>(file)); }
00842 };
00843
00857 class File_system
00858 {
00859 protected:

```

```

00860 File_system *_next;
00861
00862 public:
00863 File_system() noexcept : _next(0) {}
00864 virtual char const *type() const noexcept = 0;
00870
00887 virtual int mount(char const *source, unsigned long mountflags,
00888 void const *data, cxx::Ref_ptr<File> *dir) noexcept = 0;
00889
00890 virtual ~File_system() noexcept = 0;
00891
00896 File_system *next() const noexcept { return _next; }
00897 File_system *&next() noexcept { return _next; }
00898 void next(File_system *n) noexcept { _next = n; }
00899 };
00900
00901 inline
00902 File_system::~File_system() noexcept
00903 {}
00904
00905 class File_system_list
00906 {
00907 public:
00908 class Iterator
00909 {
00910 public:
00911 explicit constexpr Iterator(File_system *c = nullptr) : _c(c) {}
00912
00913 Iterator &operator++()
00914 {
00915 if (_c)
00916 _c = _c->next();
00917 return *this;
00918 }
00919
00920 bool operator==(Iterator const &other) const { return _c == other._c; }
00921 bool operator!=(Iterator const &other) const { return _c != other._c; }
00922 File_system *operator*() const { return _c; }
00923 File_system *operator->() const { return _c; }
00924
00925 private:
00926 File_system *_c;
00927 };
00928
00929 File_system_list(File_system *head) : _head(head) {}
00930
00931 constexpr Iterator begin() const
00932 { return Iterator(_head); }
00933
00934 constexpr Iterator end() const
00935 { return Iterator(); }
00936
00937 private:
00938 File_system *_head;
00939 };
00940
00946 class Fs
00947 {
00948 public:
00954 virtual cxx::Ref_ptr<File> get_file(int fd) noexcept = 0;
00955
00957 virtual cxx::Ref_ptr<File> get_root() noexcept = 0;
00958
00960 virtual cxx::Ref_ptr<File> get_cwd() noexcept { return get_root(); }
00961
00963 virtual void set_cwd(cxx::Ref_ptr<File> const &) noexcept {}
00964
00970 virtual int alloc_fd(cxx::Ref_ptr<File> const &f = cxx::Ref_ptr<>::Nil) noexcept = 0;
00971
00982 virtual cxx::Pair<cxx::Ref_ptr<File>, int>
00983 set_fd(int fd, cxx::Ref_ptr<File> const &f = cxx::Ref_ptr<>::Nil) noexcept = 0;
00984
00990 virtual cxx::Ref_ptr<File> free_fd(int fd) noexcept = 0;
00991
00999 virtual int mount(char const *path, cxx::Ref_ptr<File> const &dir) noexcept = 0;
01000
01008 virtual int register_file_system(File_system *f) noexcept = 0;
01009
01017 virtual int unregister_file_system(File_system *f) noexcept = 0;
01018
01026 virtual File_system *get_file_system(char const *fstype) noexcept = 0;
01027
01036 virtual File_system_list file_system_list() noexcept = 0;
01037
01041 int mount(char const *source, char const *target,
01042 char const *fstype, unsigned long mountflags,
01043 void const *data) noexcept;

```



```

01044
01045 virtual int register_file_factory(cxx::Ref_ptr<File_factory> f) noexcept = 0;
01046 virtual int unregister_file_factory(cxx::Ref_ptr<File_factory> f) noexcept = 0;
01047 virtual cxx::Ref_ptr<File_factory> get_file_factory(int proto) noexcept = 0;
01048 virtual cxx::Ref_ptr<File_factory> get_file_factory(char const *proto_name) noexcept = 0;
01049
01050 virtual ~Fs() = 0;
01051
01052 private:
01053 int mount_one(char const *source, char const *target,
01054 File_system *fs, unsigned long mountflags,
01055 void const *data) noexcept;
01056 };
01057
01058 inline int
01059 Fs::mount_one(char const *source, char const *target,
01060 File_system *fs, unsigned long mountflags,
01061 void const *data) noexcept
01062 {
01063 cxx::Ref_ptr<File> dir;
01064 int res = fs->mount(source, mountflags, data, &dir);
01065
01066 if (res < 0)
01067 return res;
01068
01069 return mount(target, dir);
01070 }
01071
01072 inline int
01073 Fs::mount(char const *source, char const *target,
01074 char const *fstype, unsigned long mountflags,
01075 void const *data) noexcept
01076 {
01077 if (fstype[0] == 'a'
01078 && fstype[1] == 'u'
01079 && fstype[2] == 't'
01080 && fstype[3] == 'o'
01081 && fstype[4] == 0)
01082 {
01083 File_system_list fsl = file_system_list();
01084 for (File_system_list::Iterator c = fsl.begin(); c != fsl.end(); ++c)
01085 if (mount_one(source, target, *c, mountflags, data) == 0)
01086 return 0;
01087
01088 return -ENODEV;
01089 }
01090
01091 File_system *fs = get_file_system(fstype);
01092
01093 if (!fs)
01094 return -ENODEV;
01095
01096 return mount_one(source, target, fs, mountflags, data);
01097 }
01098
01099 inline
01100 Fs::~Fs()
01101 {}
01102
01109 class Ops : public Mman, public Fs
01110 {
01111 public:
01112 virtual void *malloc(size_t bytes) noexcept = 0;
01113 virtual void free(void *mem) noexcept = 0;
01114 virtual ~Ops() noexcept = 0;
01115
01116 char *strndup(char const *str, unsigned l) noexcept
01117 {
01118 unsigned len;
01119 for (len = 0; str[len] && len < l; ++len)
01120 ;
01121
01122 if (len == 0)
01123 return nullptr;
01124
01125 ++len;
01126
01127 char *b = static_cast<char *>(this->malloc(len));
01128 if (b == nullptr)
01129 return nullptr;
01130
01131 char *r = b;
01132 for (; len - 1 > 0 && *str; --len, ++b, ++str)
01133 *b = *str;
01134
01135 *b = 0;
01136 return r;

```

```

01137 }
01138
01139 };
01140
01141 inline
01142 Ops::~Ops() noexcept
01143 {}
01144
01145 }}
01146
01147 #endif
01148

```

## 17.249 virtio-net

```

00001 // vi:ft=cpp
00002 /* SPDX-License-Identifier: MIT */
00003 /*
00004 * Copyright (C) 2022, 2024 Kernkonzept GmbH.
00005 * Author(s): Stephan Gerhold <stephan.gerhold@kernkonzept.com>
00006 */
00007 #pragma once
00008
00009 #include <cstring>
00010 #include <functional>
00011
00012 #include <l4/cxx/exceptions>
00013 #include <l4/cxx/minmax>
00014 #include <l4/re/dataspace>
00015 #include <l4/re/env>
00016 #include <l4/re/error_helper>
00017 #include <l4/re/util/unique_cap>
00018 #include <l4/sys/consts.h>
00019
00020 #include <l4/l4virtio/client/l4virtio>
00021 #include <l4/l4virtio/l4virtio>
00022 #include <l4/l4virtio/virtio_net.h>
00023 #include <l4/l4virtio/virtqueue>
00024
00025 namespace L4virtio { namespace Driver {
00026
00030 class Virtio_net_device : public L4virtio::Driver::Device
00031 {
00032 public:
00037 struct Packet
00038 {
00039 l4virtio_net_header_t hdr;
00040 l4_uint8_t data[1500 + 14]; /* MTU + Ethernet header */
00041 };
00042
00047 int rx_queue_size() const
00048 { return max_queue_size(0); }
00049
00054 int tx_queue_size() const
00055 { return max_queue_size(1); }
00056
00066 void setup_device(L4::Cap<L4virtio::Device> srvcap)
00067 {
00068 // Contact device.
00069 driver_connect(srvcap, false);
00070
00071 if (_config->device != L4VIRTIO_ID_NET)
00072 L4Re::chksys(-L4_ENODEV, "Device is not a network device.");
00073
00074 if (_config->num_queues < 2)
00075 L4Re::chksys(-L4_EINVAL, "Invalid number of queues reported.");
00076
00077 auto rxqsz = rx_queue_size();
00078 auto txqsz = tx_queue_size();
00079
00080 // Allocate memory for RX/TX queue and RX/TX packet buffers
00081 auto rxqoff = 0;
00082 auto txqoff = l4_round_size(rxqoff + rxqsz * _rxq.total_size(rxqsz),
00083 L4virtio::Virtqueue::Desc_align);
00084 auto rxpktoff = l4_round_size(txqoff + txqsz * _txq.total_size(txqsz),
00085 L4virtio::Virtqueue::Desc_align);
00086 auto txpktoff = rxpktoff + rxqsz * sizeof(Packet);
00087 auto totalsz = txpktoff + txqsz * sizeof(Packet);
00088
00089 _queue_ds = L4Re::chkcapi(L4Re::Util::make_unique_cap<L4Re::Dataspace>(),
00090 "Allocate queue dataspace capability");
00091 auto *e = L4Re::Env::env();
00092 L4Re::chksys(e->mem_alloc()->alloc(totalsz, _queue_ds.get()),

```

```

00093 L4Re::Mem_alloc::Continuous
00094 | L4Re::Mem_alloc::Pinned),
00095 "Allocate memory for virtio structures");
00096
00097 L4Re::chksys(e->rm()->attach(&_queue_region, totalsz,
00098 L4Re::Rm::F::Search_addr | L4Re::Rm::F::RW,
00099 L4::Ipc::make_cap_rw(_queue_ds.get()), 0,
00100 L4_PAGESHIFT),
00101 "Attach dataspace for virtio structures");
00102
00103 l4_uint64_t devaddr;
00104 L4Re::chksys(register_ds(_queue_ds.get(), 0, totalsz, &devaddr),
00105 "Register queue dataspace with device");
00106
00107 _rxq.init_queue(rxqsz, _queue_region.get() + rxqoff);
00108 _txq.init_queue(txqsz, _queue_region.get() + txqoff);
00109
00110 config_queue(0, rxqsz, devaddr + rxqoff,
00111 devaddr + rxqoff + _rxq.avail_offset(),
00112 devaddr + rxqoff + _rxq.used_offset());
00113 config_queue(1, txqsz, devaddr + txqoff,
00114 devaddr + txqoff + _txq.avail_offset(),
00115 devaddr + txqoff + _txq.used_offset());
00116
00117 _rxpkts = reinterpret_cast<Packet*>(_queue_region.get() + rxpktoff);
00118 _txpkts = reinterpret_cast<Packet*>(_queue_region.get() + txpktoff);
00119
00120 // Prepare descriptors to save work later
00121 for (l4_uint16_t descno = 0; descno < rxqsz; ++descno)
00122 {
00123 auto &desc = _rxq.desc(descno);
00124 desc.addr = L4virtio::Ptr<void>(devaddr + rxpktoff +
00125 descno * sizeof(Packet));
00126 desc.len = sizeof(Packet);
00127 desc.flags.write() = 1;
00128 }
00129 for (l4_uint16_t descno = 0; descno < txqsz; ++descno)
00130 {
00131 auto &desc = _txq.desc(descno);
00132 desc.addr = L4virtio::Ptr<void>(devaddr + txpktoff +
00133 descno * sizeof(Packet));
00134 desc.len = sizeof(Packet);
00135 }
00136
00137 // Setup notification IRQ
00138 _driver_notification_irq =
00139 L4Re::chkcap(L4Re::Util::make_unique_cap<L4::Irq>()),
00140 "Allocate notification capability");
00141
00142 L4Re::chksys(l4_error(e->factory()->create(_driver_notification_irq.get())),
00143 "Create irq for notifications from device");
00144
00145 L4Re::chksys(_device->bind(0, _driver_notification_irq.get()),
00146 "Bind driver notification interrupt");
00147
00148 // Finish handshake with device
00149 l4virtio_set_feature(_config->driver_features_map,
00150 L4VIRTIO_FEATURE_VERSION_1);
00151 l4virtio_set_feature(_config->driver_features_map, L4VIRTIO_NET_F_MAC);
00152 driver_acknowledge();
00153 }
00154
00158 l4virtio_net_config_t const &device_config() const
00159 {
00160 return *_config->device_config<l4virtio_net_config_t>();
00161 }
00162
00169 int bind_rx_notification_irq(L4::Cap<L4::Thread> thread, l4_umword_t label)
00170 {
00171 return l4_error(_driver_notification_irq->bind_thread(thread, label));
00172 }
00173
00180 Packet &rx_pkt(l4_uint16_t descno)
00181 {
00182 if (descno >= _rxq.num())
00183 throw L4::Bounds_error("Invalid used descriptor number in RX queue");
00184 return _rxpkts[descno];
00185 }
00186
00202 l4_uint16_t wait_rx(l4_uint32_t *len = nullptr)
00203 {
00204 l4_uint16_t descno;
00205 // Wait until used descriptor becomes available.
00206 for (;;)
00207 {
00208 descno = _rxq.find_next_used(len);
00209 if (descno != Virtqueue::Eoq)

```

```

00210 break;
00211
00212 L4Re::chksys(_driver_notification_irq->receive(), "Wait for RX");
00213 }
00214
00215 if (len)
00216 // Ensure that the length provided by the device in wait_for_next_used()
00217 // is not larger than the buffer and subtract the length of the header.
00218 *len = cxx::min(*len - sizeof(_rxpkts[0].hdr), sizeof(_rxpkts[0].data));
00219 return descno;
00220 }
00221
00230 void finish_rx(l4_uint16_t descno)
00231 {
00232 _rxq.free_descriptor(descno, descno);
00233 }
00234
00238 void queue_rx()
00239 {
00240 l4_uint16_t descno;
00241 while ((descno = _rxq.alloc_descriptor()) != Virtqueue::Eoq)
00242 _rxq.enqueue_descriptor(descno);
00243 notify(_rxq);
00244 }
00245
00260 bool tx(std::function<l4_uint32_t(Packet&)> prepare)
00261 {
00262 auto descno = _txq.alloc_descriptor();
00263 if (descno == Virtqueue::Eoq)
00264 {
00265 // Try again after cleaning old descriptors that have already been used
00266 free_used_tx_descriptors();
00267 descno = _txq.alloc_descriptor();
00268 if (descno == Virtqueue::Eoq)
00269 return false;
00270 }
00271
00272 auto &pkt = _txpkts[descno];
00273 auto &desc = _txq.desc(descno);
00274 desc.len = sizeof(pkt.hdr) + prepare(pkt);
00275 send(_txq, descno);
00276 return true;
00277 }
00278
00279 private:
00280 void free_used_tx_descriptors()
00281 {
00282 l4_uint16_t used;
00283 while ((used = _txq.find_next_used()) != Virtqueue::Eoq)
00284 {
00285 if (used >= _txq.num())
00286 throw L4::Bounds_error("Invalid used descriptor number in TX queue");
00287 _txq.free_descriptor(used, used);
00288 }
00289 }
00290
00291 private:
00292 L4Re::Util::Unique_cap<L4Re::Dataspace> _queue_ds;
00293 L4Re::Rm::Unique_region<l4_uint8_t *> _queue_region;
00294 L4Re::Util::Unique_cap<L4::Irq> _driver_notification_irq;
00295 L4virtio::Driver::Virtqueue _rxq, _txq;
00296 Packet *_rxpkts, *_txpkts;
00297 };
00298
00299 } }

```

## 17.250 l4virtio

```

00001 // vi:ft=c++
00002 /* SPDX-License-Identifier: MIT */
00003 /*
00004 * Copyright (C) 2015-2020, 2022, 2024 Kernkonzept GmbH.
00005 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00006 *
00007 */
00008 #pragma once
00009
00010 #include <l4/sys/factory>
00011 #include <l4/sys/semaphore>
00012 #include <l4/re/dataspace>
00013 #include <l4/re/env>
00014 #include <l4/re/util/unique_cap>
00015 #include <l4/re/util/object_registry>

```

```

00016 #include <l4/re/error_helper>
00017
00018 #include <l4/util/atomic.h>
00019 #include <l4/util/bitops.h>
00020 #include <l4/l4virtio/l4virtio>
00021 #include <l4/l4virtio/virtqueue>
00022 #include <l4/sys/consts.h>
00023
00024 #include <cstring>
00025
00026 namespace L4virtio { namespace Driver {
00027
00031 class Device
00032 {
00033 public:
00056 void driver_connect(L4::Cap<L4virtio::Device> srvcap, bool manage_notify = true)
00057 {
00058 _device = srvcap;
00059
00060 _next_devaddr = L4_SUPERPAGESIZE;
00061
00062 auto *e = L4Re::Env::env();
00063
00064 // Set up the virtio configuration page.
00065
00066 _config_cap = L4Re::chkcapi(L4Re::Util::make_unique_cap<L4Re::Dataspace>(),
00067 "Allocate config dataspace capability");
00068
00069 l4_addr_t ds_offset;
00070 L4Re::chksys(_device->device_config(_config_cap.get(), &ds_offset),
00071 "Request virtio config page");
00072
00073 if (ds_offset & ~L4_PAGEMASK)
00074 L4Re::chksys(-L4_EINVAL, "Virtio config page is page aligned.");
00075
00076 L4Re::chksys(e->rm()->attach(&_config, L4_PAGESIZE,
00077 L4Re::Rm::F::Search_addr | L4Re::Rm::F::RW,
00078 L4::Ipc::make_cap_rw(_config_cap.get(), ds_offset,
00079 L4_PAGESHIFT),
00080 "Attach config dataspace");
00081
00082 if (memcmp(&_config->magic, "virt", 4) != 0)
00083 L4Re::chksys(-L4_ENODEV, "Device config has wrong magic value");
00084
00085 if (_config->version != 2)
00086 L4Re::chksys(-L4_ENODEV, "Invalid virtio version, must be 2");
00087
00088 _device->set_status(0); // reset
00089 int status = L4VIRTIO_STATUS_ACKNOWLEDGE;
00090 _device->set_status(status);
00091
00092 status |= L4VIRTIO_STATUS_DRIVER;
00093 _device->set_status(status);
00094
00095 if (_config->fail_state())
00096 L4Re::chksys(-L4_EIO, "Device failure during initialisation.");
00097
00098 // Set up the interrupt used to notify the device about events.
00099 // (only supporting one interrupt with index 0 at the moment)
00100
00101 _host_irq = L4Re::chkcapi(L4Re::Util::make_unique_cap<L4::Irq>(),
00102 "Allocate host IRQ capability");
00103
00104 L4Re::chksys(_device->device_notification_irq(0, _host_irq.get()),
00105 "Request device notification interrupt.");
00106
00107 // Set up the interrupt to get notifications from the device.
00108 // (only supporting one interrupt with index 0 at the moment)
00109 if (manage_notify)
00110 {
00111 _driver_notification =
00112 L4Re::chkcapi(L4Re::Util::make_unique_cap<L4::Semaphore>(),
00113 "Allocate notification capability");
00114
00115 L4Re::chksys(l4_error(e->factory()->create(_driver_notification.get())),
00116 "Create semaphore for notifications from device");
00117
00118 L4Re::chksys(_device->bind(0, _driver_notification.get()),
00119 "Bind driver notification interrupt");
00120 }
00121 }
00122
00129 int bind_notification_irq(unsigned index, L4::Cap<L4::Triggerable> irq) const
00130 { return l4_error(_device->bind(index, irq)); }
00131
00133 bool fail_state() const { return _config->fail_state(); }
00134

```

```

00145 bool feature_negotiated(unsigned int feat) const
00146 { return l4virtio_get_feature(_config->driver_features_map, feat); }
00147
00156 int driver_acknowledge()
00157 {
00158 if (!l4virtio_get_feature(_config->dev_features_map,
00159 L4VIRTIO_FEATURE_VERSION_1))
00160 L4Re::chksys(-L4_ENODEV,
00161 "Require Virtio 1.0 device; Legacy device not supported.");
00162
00163 _config->driver_features_map[0] &= _config->dev_features_map[0];
00164 _config->driver_features_map[1] &= _config->dev_features_map[1];
00165
00166 _device->set_status(_config->status | L4VIRTIO_STATUS_FEATURES_OK);
00167
00168 if (!(_config->status & L4VIRTIO_STATUS_FEATURES_OK))
00169 L4Re::chksys(-L4_EINVAL, "Negotiation of device features.");
00170
00171 _device->set_status(_config->status | L4VIRTIO_STATUS_DRIVER_OK);
00172
00173 if (_config->fail_state())
00174 return -L4_EIO;
00175
00176 return L4_EOK;
00177 }
00178
00196 int register_ds(L4::Cap<L4Re::Dataspace> ds, l4_umword_t offset,
00197 l4_umword_t size, l4_uint64_t *devaddr)
00198 {
00199 *devaddr = next_device_address(size);
00200 return _device->register_ds(L4::Ipc::make_cap_rw(ds), *devaddr, offset, size);
00201 }
00202
00212 int config_queue(int num, unsigned size, l4_uint64_t desc_addr,
00213 l4_uint64_t avail_addr, l4_uint64_t used_addr)
00214 {
00215 auto *queueconf = &_amp;config->queues()[num];
00216 queueconf->num = size;
00217 queueconf->desc_addr = desc_addr;
00218 queueconf->avail_addr = avail_addr;
00219 queueconf->used_addr = used_addr;
00220 queueconf->ready = 1;
00221
00222 return _device->config_queue(num);
00223 }
00224
00230 int max_queue_size(int num) const
00231 {
00232 return _config->queues()[num].num_max;
00233 }
00234
00247 int send_and_wait(Virtqueue &queue, l4_uint16_t descno)
00248 {
00249 send(queue, descno);
00250
00251 // wait for a reply, we assume that no other
00252 // request will get in the way.
00253 auto head = wait_for_next_used(queue);
00254
00255 if (head < 0)
00256 return head;
00257
00258 return (head == descno) ? L4_EOK : -L4_EINVAL;
00259 }
00260
00268 int wait(int index) const
00269 {
00270 if (index != 0)
00271 return -L4_EEXIST;
00272
00273 return l4_ipc_error(_driver_notification->down(), l4_utcb());
00274 }
00275
00291 int wait_for_next_used(Virtqueue &queue, l4_uint32_t *len = nullptr) const
00292 {
00293 while (true)
00294 {
00295 int err = wait(0);
00296
00297 if (err < 0)
00298 return err;
00299
00300 auto head = queue.find_next_used(len);
00301 if (head != Virtqueue::Eoq) // spurious interrupt?
00302 return head;
00303 }
00304 }

```

```

00305
00312 void send(Virtqueue &queue, l4_uint16_t descno)
00313 {
00314 queue.enqueue_descriptor(descno);
00315 notify(queue);
00316 }
00317
00318 void notify(Virtqueue &queue)
00319 {
00320 if (!queue.no_notify_host())
00321 _host_irq->trigger();
00322 }
00323
00324 private:
00325 l4_uint64_t next_device_address(l4_umword_t size)
00326 {
00327 l4_umword_t ret;
00328 size = l4_round_page(size);
00329 do
00330 {
00331 ret = _next_devaddr;
00332 if (l4_umword_t(~0) - ret < size)
00333 L4Re::chksys(-L4_ENOMEM, "Out of device address space.");
00334 }
00335 while (!l4util_cmpxchg(&_next_devaddr, ret, ret + size));
00336 return ret;
00337 }
00338
00339 protected:
00340 L4::Cap<L4virtio::Device> _device;
00341 L4Re::Rm::Unique_region<L4virtio::Device::Config_hdr *> _config;
00342 l4_umword_t _next_devaddr;
00343 L4Re::Util::Unique_cap<L4::Semaphore> _driver_notification;
00344
00345 private:
00346 L4Re::Util::Unique_cap<L4::Irq> _host_irq;
00347 L4Re::Util::Unique_cap<L4Re::Dataspace> _config_cap;
00348 };
00349
00350 }

```

## 17.251 l4virtio

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /* SPDX-License-Identifier: MIT */
00003 /*
00004 * Copyright (C) 2013-2024 Kernkonzept GmbH.
00005 * Author(s): Alexander Warg <alexander.warg@kernkonzept.com>
00006 * Matthias Lange <matthias.lange@kernkonzept.com>
00007 */
00008
00009 #pragma once
00010
00011 #include "virtio.h"
00012 #include <l4/sys/capability>
00013 #include <l4/sys/cxx/ipc_client>
00014 #include <l4/re/dataspace>
00015 #include <l4/sys/irq>
00016 #include <l4/cxx/utis>
00017
00018 namespace L4virtio {
00019 class Device :
00020 public L4::Kobject_t<Device, L4::Icu, L4VIRTIO_PROTOCOL,
00021 L4::Type_info::Demand_t<1> >
00022 {
00023 public:
00024 typedef l4virtio_config_queue_t Config_queue;
00025 struct Config_hdr : l4virtio_config_hdr_t
00026 {
00027 Config_queue *queues() const
00028 { return l4virtio_config_queues(this); }
00029
00030 template <typename T>
00031 T *device_config() const
00032 {
00033 return static_cast<T*>(l4virtio_device_config(this));
00034 }
00035 };
00036
00037 int config_queue(unsigned num, L4::Cap<L4::Triggerable> out_notify,
00038 L4::Cap<L4::Triggerable> in_notify,
00039 l4_timeout_s to = L4_IPC_TIMEOUT_NEVER)
00040 {

```

```

00060 return send_cmd(L4VIRTIO_CMD_CFG_QUEUE | num,
00061 out_notify, in_notify, to);
00062 }
00063
00064 int notify_queue(unsigned num, L4::Cap<L4::Triggerable> out_notify,
00065 L4::Cap<L4::Triggerable> in_notify,
00066 l4_timeout_s to = L4_IPC_TIMEOUT_NEVER)
00067 {
00068 return send_cmd(L4VIRTIO_CMD_NOTIFY_QUEUE | num,
00069 out_notify, in_notify, to);
00070 }
00071
00080 bool fail_state() const
00081 {
00082 auto cfg_status = cxx::access_once(&status);
00083 return cfg_status
00084 & (L4VIRTIO_STATUS_FAILED | L4VIRTIO_STATUS_DEVICE_NEEDS_RESET);
00085 }
00086
00087 int set_status(unsigned new_status, L4::Cap<L4::Triggerable> out_notify,
00088 L4::Cap<L4::Triggerable> in_notify,
00089 l4_timeout_s to = L4_IPC_TIMEOUT_NEVER)
00090 {
00091 return send_cmd(L4VIRTIO_CMD_SET_STATUS | new_status,
00092 out_notify, in_notify, to);
00093 }
00094
00095 int cfg_changed(unsigned reg, L4::Cap<L4::Triggerable> out_notify,
00096 L4::Cap<L4::Triggerable> in_notify,
00097 l4_timeout_s to = L4_IPC_TIMEOUT_NEVER)
00098 {
00099 return send_cmd(L4VIRTIO_CMD_CFG_CHANGED | reg,
00100 out_notify, in_notify, to);
00101 }
00102
00103 int send_cmd(unsigned command, L4::Cap<L4::Triggerable> out_notify,
00104 L4::Cap<L4::Triggerable> in_notify,
00105 l4_timeout_s to = L4_IPC_TIMEOUT_NEVER)
00106 {
00107 cxx::write_now(&cmd, command);
00108
00109 if (out_notify)
00110 out_notify->trigger();
00111
00112 auto utcb = l4_utcb();
00113 auto ipc_to = l4_timeout(L4_IPC_TIMEOUT_0, to);
00114
00115 do
00116 {
00117 if (in_notify)
00118 if (l4_ipc_error(l4_ipc_receive(in_notify.cap(), utcb, ipc_to),
00119 utcb) == L4_IPC_RETIMEOUT)
00120 break;
00121 }
00122 while (cxx::access_once(&cmd));
00123
00124 return cxx::access_once(&cmd) ? -L4_EBUSY : L4_EOK;
00125 }
00126 };
00127
00137 L4_INLINE_RPC_OP(L4VIRTIO_OP_SET_STATUS, long,
00138 set_status, (unsigned status));
00139
00155 L4_INLINE_RPC_OP(L4VIRTIO_OP_CONFIG_QUEUE, long,
00156 config_queue, (unsigned queue));
00157
00181 L4_INLINE_RPC_OP(L4VIRTIO_OP_REGISTER_DS, long,
00182 register_ds, (L4::Ipc::Cap<L4Re::Dataspace> ds_cap,
00183 l4_uint64_t base, l4_umword_t offset,
00184 l4_umword_t size));
00185
00194 L4_INLINE_RPC_OP(L4VIRTIO_OP_DEVICE_CONFIG, long, device_config,
00195 (L4::Ipc::Out<L4::Cap<L4Re::Dataspace> > config_ds,
00196 l4_addr_t *ds_offset));
00197
00216 L4_INLINE_RPC_OP(L4VIRTIO_OP_GET_DEVICE_IRQ, long, device_notification_irq,
00217 (unsigned index, L4::Ipc::Out<L4::Cap<L4::Triggerable> > irq));
00218
00219 typedef L4::Typeid::Rpcs<set_status_t, config_queue_t, register_ds_t,
00220 device_config_t, device_notification_irq_t>
00221 Rpcs;
00222 };
00223 };
00224
00225 }

```



## 17.252 l4virtio

```

00001 // vi:ft=cpp
00002 /* SPDX-License-Identifier: MIT */
00003 /*
00004 * Copyright (C) 2014-2024 Kernkonzept GmbH.
00005 * Author(s): Alexander Warg <alexander.warg@kernkonzept.com>
00006 * Manuel von Oltersdorff-Kalettkka <manuel.kalettkka@kernkonzept.com>
00007 */
00008 #pragma once
00009
00010 #include <algorithm>
00011 #include <limits.h>
00012 #include <memory>
00013 #include <vector>
00014
00015 #include <l4/re/dataspace>
00016 #include <l4/re/util/debug>
00017 #include <l4/re/env>
00018 #include <l4/re/error_helper>
00019 #include <l4/re/rm>
00020 #include <l4/re/util/cap_alloc>
00021 #include <l4/re/util/shared_cap>
00022 #include <l4/re/util/unique_cap>
00023
00024 #include <l4/sys/types.h>
00025 #include <l4/re/util/meta>
00026
00027 #include <l4/cxx/bitfield>
00028 #include <l4/cxx/utils>
00029 #include <l4/cxx/unique_ptr>
00030
00031 #include <l4/sys/cxx/ipc_legacy>
00032
00033 #include "../l4virtio"
00034 #include "virtio"
00035
00036 namespace L4virtio {
00037 namespace Svr {
00038
00039 class Dev_config
00040 {
00041 public:
00042 typedef Dev_status Status;
00043 typedef Dev_features Features;
00044 private:
00045 typedef L4Re::Rm::Unique_region<l4virtio_config_hdr_t*> Cfg_region;
00046 typedef L4Re::Util::Shared_cap<L4Re::Dataspace> Cfg_cap;
00047
00048 l4_uint32_t _vendor, _device, _qoffset, _nqueues;
00049 l4_uint32_t _host_features[sizeof(l4virtio_config_hdr_t::dev_features_map)
00050 / sizeof(l4_uint32_t)];
00051
00052 Cfg_cap _ds;
00053 Cfg_region _config;
00054 l4_addr_t _ds_offset = 0;
00055
00056 Status _status{0}; // status shadow, can be trusted by the device model
00057
00058 static l4_uint32_t align(l4_uint32_t x)
00059 { return (x + 0xfU) & ~0xfU; }
00060
00061 void attach_n_init_cfg(Cfg_cap const &cfg, l4_addr_t offset)
00062 {
00063 L4Re::chksys(L4Re::Env::env()->rm()->attach(&_config, L4_PAGESIZE,
00064 L4Re::Rm::F::Search_addr | L4Re::Rm::F::RW,
00065 L4::Ipc::make_cap_rw(cfg.get()),
00066 offset),
00067 "Attach config space to local address space.");
00068
00069 _config->generation = 0;
00070 memset(_config->driver_features_map, 0, sizeof(_config->driver_features_map));
00071 memset(_host_features, 0, sizeof(_host_features));
00072 set_host_feature(L4VIRTIO_FEATURE_VERSION_1);
00073 reset_hdr();
00074
00075 _ds = cfg;
00076 _ds_offset = offset;
00077 }
00078
00079 protected:
00080 void volatile *get_priv_config() const
00081 {
00082 return l4virtio_device_config(_config.get());
00083 }
00084 }

```

```

00097
00098 public:
00099
00112 Dev_config(l4_uint32_t vendor, l4_uint32_t device,
00113 unsigned cfg_size, l4_uint32_t num_queues = 0)
00114 : _vendor(vendor), _device(device),
00115 _qoffset(0x100 + align(cfg_size)),
00116 _nqueues(num_queues)
00117 {
00118 using L4Re::Dataspace;
00119 using L4Re::chkcapi;
00120 using L4Re::chksys;
00121
00122 if (sizeof(l4virtio_config_queue_t) * _nqueues + _qoffset > L4_PAGESIZE)
00123 {
00124 // too many queues does not fit into our page
00125 _qoffset = 0;
00126 _nqueues = 0;
00127 }
00128
00129 auto cfg = chkcapi(L4Re::Util::make_shared_cap<Dataspace>());
00130 chksys(L4Re::Env::env()->mem_alloc()->alloc(L4_PAGESIZE, cfg.get()));
00131
00132 attach_n_init_cfg(cfg, 0);
00133 }
00134
00146 Dev_config(Cfg_cap const &cfg, l4_addr_t cfg_offset,
00147 l4_uint32_t vendor, l4_uint32_t device,
00148 unsigned cfg_size, l4_uint32_t num_queues = 0)
00149 : _vendor(vendor), _device(device),
00150 _qoffset(0x100 + align(cfg_size)),
00151 _nqueues(num_queues)
00152 {
00153 if (sizeof(l4virtio_config_queue_t) * _nqueues + _qoffset > L4_PAGESIZE)
00154 {
00155 // too many queues does not fit into our page
00156 _qoffset = 0;
00157 _nqueues = 0;
00158 }
00159
00160 attach_n_init_cfg(cfg, cfg_offset);
00161 }
00162
00163 void set_host_feature(unsigned feature)
00164 { l4virtio_set_feature(_host_features, feature); }
00165
00166 void clear_host_feature(unsigned feature)
00167 { l4virtio_clear_feature(_host_features, feature); }
00168
00169 bool get_host_feature(unsigned feature)
00170 { return l4virtio_get_feature(_host_features, feature); }
00171
00172 bool get_guest_feature(unsigned feature)
00173 { return l4virtio_get_feature(_config->driver_features_map, feature); }
00174
00175 l4_uint32_t &host_features(unsigned idx)
00176 { return _host_features[idx]; }
00177
00178 l4_uint32_t host_features(unsigned idx) const
00179 { return _host_features[idx]; }
00180
00184 l4_uint32_t num_queues() const
00185 { return _nqueues; }
00186
00198 l4_uint32_t guest_features(unsigned idx) const
00199 { return _config->driver_features_map[idx]; }
00200
00212 l4_uint32_t negotiated_features(unsigned idx) const
00213 { return _config->driver_features_map[idx] & _host_features[idx]; }
00214
00222 Status status() const { return _status; }
00223
00230 l4_uint32_t get_cmd() const
00231 {
00232 return hdr()->cmd;
00233 }
00234
00241 void reset_cmd()
00242 {
00243 const_cast<l4_uint32_t volatile &>(hdr()->cmd) = 0;
00244 }
00245
00253 void set_status(Status status)
00254 {
00255 _status = status;
00256 const_cast<l4_uint32_t volatile &>(hdr()->status) = status.raw;
00257 }

```

```

00258
00265 void add_irq_status(l4_uint32_t status)
00266 {
00267 const_cast<l4_uint32_t volatile &>(hdr()->irq_status) |= status;
00268 }
00269
00276 void set_device_needs_reset()
00277 {
00278 _status.device_needs_reset() = 1;
00279 const_cast<l4_uint32_t volatile &>(hdr()->status) = _status.raw;
00280 }
00281
00286 bool change_queue_config(l4_uint32_t num_queues)
00287 {
00288 if (sizeof(l4virtio_config_queue_t) * num_queues + _qoffset > L4_PAGESIZE)
00289 // too many queues does not fit into our page
00290 return false;
00291
00292 _nqueues = num_queues;
00293 reset_hdr(true);
00294 return true;
00295 }
00296
00303 l4virtio_config_queue_t volatile const *qconfig(unsigned index) const
00304 {
00305 if (L4_UNLIKELY(_qoffset < sizeof (l4virtio_config_hdr_t)))
00306 return 0;
00307
00308 if (L4_UNLIKELY(index >= _nqueues))
00309 return 0;
00310
00311 return reinterpret_cast<l4virtio_config_queue_t const *>
00312 (reinterpret_cast<char *>(_config.get()) + _qoffset) + index;
00313 }
00314
00318 void reset_hdr(bool inc_generation = false) const
00319 {
00320 _config->magic = L4VIRTIO_MAGIC;
00321 _config->version = 2;
00322 _config->device = _device;
00323 _config->vendor = _vendor;
00324 _config->status = 0;
00325 _config->irq_status = 0;
00326 _config->num_queues = _nqueues;
00327 _config->queues_offset = _qoffset;
00328
00329 memcpy(_config->dev_features_map, _host_features,
00330 sizeof(_config->dev_features_map));
00331 wmb();
00332 if (inc_generation)
00333 ++_config->generation;
00334 }
00335
00345 bool reset_queue(unsigned index, unsigned num_max,
00346 bool inc_generation = false) const
00347 {
00348 l4virtio_config_queue_t volatile *qc;
00349 // this function is allowed to write to the device config
00350 qc = const_cast<l4virtio_config_queue_t volatile *>(qconfig(index));
00351 if (L4_UNLIKELY(qc == 0))
00352 return false;
00353
00354 qc->num_max = num_max;
00355 qc->num = 0;
00356 qc->ready = 0;
00357 wmb();
00358 if (inc_generation)
00359 ++_config->generation;
00360
00361 return true;
00362 }
00363
00368 l4virtio_config_hdr_t const volatile *hdr() const
00369 { return _config.get(); }
00370
00375 L4::Cap<L4Re::Dataspace> ds() const { return _ds.get(); }
00376
00381 l4_addr_t ds_offset() const
00382 { return _ds_offset; }
00383 };
00384
00385
00386 template<typename PRIV_CONFIG>
00387 class Dev_config_t : public Dev_config
00388 {
00389 public:

```

```

00391 typedef PRIV_CONFIG Priv_config;
00392
00404 Dev_config_t(l4_uint32_t vendor, l4_uint32_t device,
00405 l4_uint32_t num_queues = 0)
00406 : Dev_config(vendor, device, sizeof(PRIV_CONFIG), num_queues)
00407 {}
00408
00419 Dev_config_t(L4Re::Util::Shared_cap<L4Re::Dataspace> const &cfg,
00420 l4_addr_t cfg_offset, l4_uint32_t vendor, l4_uint32_t device,
00421 l4_uint32_t num_queues = 0)
00422 : Dev_config(cfg, cfg_offset, vendor, device, sizeof(PRIV_CONFIG),
00423 num_queues)
00424 {}
00425
00435 Priv_config volatile *priv_config() const
00436 {
00437 return static_cast<Priv_config volatile *>(get_priv_config());
00438 }
00439
00440 };
00441
00442 struct No_custom_data {};
00443
00449 template <typename DATA>
00450 class Driver_mem_region_t : public DATA
00451 {
00452 public:
00453 struct Flags
00454 {
00455 Flags() = default;
00456 explicit Flags(l4_uint32_t raw) : raw(raw) {}
00457
00458 l4_uint32_t raw;
00459 CXX_BITFIELD_MEMBER(0, 0, rw, raw);
00460 };
00461
00462 private:
00464 typedef L4Re::Util::Unique_cap<L4Re::Dataspace> Ds_cap;
00465
00466 l4_uint64_t _drv_base;
00467 l4_uint64_t _trans_offset;
00468 l4_umword_t _size;
00469 Flags _flags;
00470
00471 Ds_cap _ds;
00472 l4_addr_t _ds_offset;
00473
00475 L4Re::Rm::Unique_region<l4_addr_t> _local_base;
00476
00477 template<typename T>
00478 T _local(l4_uint64_t addr) const
00479 {
00480 return reinterpret_cast<T>(addr - _trans_offset);
00481 }
00482
00483 public:
00485 Driver_mem_region_t() : _size(0) {}
00486
00498 Driver_mem_region_t(l4_uint64_t drv_base, l4_umword_t size,
00499 l4_addr_t offset, Ds_cap &ds)
00500 : _drv_base(l4_trunc_page(drv_base)), _size(0), _flags(0),
00501 _ds_offset(l4_trunc_page(offset))
00502 {
00503 using L4Re::chksys;
00504 using L4Re::Env;
00505
00506 L4Re::Dataspace::Stats ds_info = L4Re::Dataspace::Stats();
00507 // Sometimes we work with dataspace that do not implement all dataspace
00508 // methods and return an error instead. An example of such a dataspace is
00509 // io's Vi::System_bus. We detect this case when the info method returns
00510 // -L4_ENOSYS and simply assume the dataspace is good for us.
00511 long err = ds->info(&ds_info);
00512 if (err >= 0)
00513 {
00514 l4_addr_t ds_size = l4_round_page(ds_info.size);
00515
00516 if (ds_size < L4_PAGESIZE)
00517 chksys(-L4_EINVAL, "DS too small");
00518
00519 if (_ds_offset >= ds_size)
00520 chksys(-L4_ERANGE, "offset larger than DS size");
00521
00522 size = l4_round_page(size);
00523 if (size > ds_size)
00524 chksys(-L4_EINVAL, "size larger than DS size");
00525
00526 if (_ds_offset > ds_size - size)

```

```

00527 chksys(-L4_EINVAL, "invalid offset or size");
00528
00529 // overflow check
00530 if ((ULONG_MAX - size) < _drv_base)
00531 chksys(-L4_EINVAL, "invalid size");
00532
00533 _flags.rw() = (ds_info.flags & L4Re::Dataspace::F::W).raw != 0;
00534 }
00535 else if (err == -L4_ENOSYS)
00536 {
00537 _flags.rw() = true;
00538 }
00539 else
00540 {
00541 chksys(err, "getting data-space infos");
00542 }
00543
00544 auto f = L4Re::Rm::F::Search_addr | L4Re::Rm::F::R;
00545 if (_flags.rw())
00546 f |= L4Re::Rm::F::W;
00547
00548 // use a big alignment to save PT/TLB entries and kernel memory resources!
00549 chksys(Env::env()->rm()->attach(&_local_base, size, f,
00550 L4::Ipc::make_cap(ds.get(), _flags.rw()
00551 ? L4_CAP_FPAGE_RW
00552 : L4_CAP_FPAGE_RO),
00553 _ds_offset, L4_SUPERPAGESHIFT));
00554
00555 _size = size;
00556 _ds = cxx::move(ds);
00557 _trans_offset = _drv_base - _local_base.get();
00558 }
00559
00560 bool is_writable() const { return _flags.rw(); }
00561
00562 Flags flags() const { return _flags; }
00563
00564 bool empty() const
00565 { return _size == 0; }
00566
00567 l4_uint64_t drv_base() const { return _drv_base; }
00568
00569 l4_addr_t local_base() const { return _local_base.get(); }
00570
00571 l4_umword_t size() const { return _size; }
00572
00573 l4_addr_t ds_offset() const { return _ds_offset; }
00574
00575 L4::Cap<L4Re::Dataspace> ds() const { return _ds.get(); }
00576
00577 bool contains(l4_uint64_t base, l4_umword_t size) const
00578 {
00579 if (base < _drv_base)
00580 return false;
00581
00582 if (base > _drv_base + _size - 1)
00583 return false;
00584
00585 if (size > _size)
00586 return false;
00587
00588 if (base - _drv_base > _size - size)
00589 return false;
00590
00591 return true;
00592 }
00593
00594 template<typename T>
00595 T *local(Ptr<T> p) const
00596 { return _local<T*>(p.get()); }
00597 };
00598
00599 typedef Driver_mem_region_t<No_custom_data> Driver_mem_region;
00600
00601 template <typename DATA>
00602 class Driver_mem_list_t
00603 {
00604 public:
00605 typedef Driver_mem_region_t<DATA> Mem_region;
00606
00607 private:
00608 cxx::unique_ptr<Mem_region[]> _l;
00609 unsigned _max;
00610 unsigned _free;
00611
00612 public:
00613 typedef L4Re::Util::Unique_cap<L4Re::Dataspace> Ds_cap;

```

```

00642
00644 Driver_mem_list_t() : _max(0), _free(0) {}
00645
00650 void init(unsigned max)
00651 {
00652 _l = cxx::make_unique<Driver_mem_region_t<DATA>[]>(max);
00653 _max = max;
00654 _free = 0;
00655 }
00656
00658 bool full() const
00659 { return _free == _max; }
00660
00669 Mem_region const *add(l4_uint64_t drv_base, l4_umword_t size,
00670 l4_addr_t offset, Ds_cap &&ds)
00671 {
00672 if (full())
00673 L4Re::chksys(-L4_ENOMEM);
00674
00675 _l[_free++] = Mem_region(drv_base, size, offset, cxx::move(ds));
00676 return &_l[_free - 1];
00677 }
00678
00683 void remove(Mem_region const *r)
00684 {
00685 if (r < &_l[0] || r >= &_l[_free])
00686 L4Re::chksys(-L4_ERANGE);
00687
00688 unsigned idx = r - &_l[0];
00689
00690 for (unsigned i = idx + 1; i < _free - 1; ++i)
00691 _l[i] = cxx::move(_l[i + 1]);
00692
00693 _l[--_free] = Mem_region();
00694 }
00695
00703 Mem_region *find(l4_uint64_t base, l4_umword_t size) const
00704 {
00705 return _find(base, size);
00706 }
00707
00717 void load_desc(Virtqueue::Desc const &desc, Request_processor const *p,
00718 Virtqueue::Desc const **table) const
00719 {
00720 Mem_region const *r = find(desc.addr.get(), desc.len);
00721 if (L4_UNLIKELY(!r))
00722 throw Bad_descriptor(p, Bad_descriptor::Bad_address);
00723
00724 *table = static_cast<Virtqueue::Desc const *>(r->local(desc.addr));
00725 }
00726
00737 void load_desc(Virtqueue::Desc const &desc, Request_processor const *p,
00738 Mem_region const **data) const
00739 {
00740 Mem_region const *r = find(desc.addr.get(), desc.len);
00741 if (L4_UNLIKELY(!r))
00742 throw Bad_descriptor(p, Bad_descriptor::Bad_address);
00743
00744 *data = r;
00745 }
00746
00763 template<typename ARG>
00764 void load_desc(Virtqueue::Desc const &desc, Request_processor const *p,
00765 ARG *data) const
00766 {
00767 Mem_region *r = find(desc.addr.get(), desc.len);
00768 if (L4_UNLIKELY(!r))
00769 throw Bad_descriptor(p, Bad_descriptor::Bad_address);
00770
00771 *data = ARG(r, desc, p);
00772 }
00773
00774 Mem_region *begin() { return &_l[0]; }
00775 Mem_region const *begin() const { return &_l[0]; }
00776
00777 Mem_region *end() { return &_l[_free]; }
00778 Mem_region const *end() const { return &_l[_free]; }
00779
00780 private:
00781 Mem_region *_find(l4_uint64_t base, l4_umword_t size) const
00782 {
00783 for (unsigned i = 0; i < _free; ++i)
00784 if (_l[i].contains(base, size))
00785 return &_l[i];
00786 return 0;
00787 }
00788

```

```

00789
00790 };
00791
00792 typedef Driver_mem_list_t<No_custom_data> Driver_mem_list;
00793
00800 template<typename DATA>
00801 class Device_t
00802 {
00803 public:
00804 typedef Driver_mem_list_t<DATA> Mem_list;
00805
00806 protected:
00807 Mem_list _mem_info;
00808
00809 private:
00810 Dev_config *_device_config;
00811
00812 using Ds_vector = std::vector<L4::Cap<L4Re::Dataspace>;
00814 std::shared_ptr<Ds_vector const> _trusted_ds_caps;
00815
00817 bool _trusted_ds_validation_enabled = false;
00818
00819 public:
00820 L4_RPC_LEGACY_DISPATCH(L4virtio::Device);
00821 template<typename IOS> int virtio_dispatch(unsigned r, IOS &ios)
00822 { return dispatch(r, ios); }
00823
00825 virtual void reset() = 0;
00826
00828 virtual bool check_features()
00829 { return true; }
00830
00832 virtual bool check_queues() = 0;
00833
00835 virtual int reconfig_queue(unsigned idx) = 0;
00836
00838 virtual void cfg_changed(unsigned /* reg */) {};
00839
00841 virtual void register_single_driver_irq()
00842 { L4Re::chksys(-L4_ENOSYS, "Legacy single IRQ interface not implemented."); }
00843
00845 virtual void trigger_driver_config_irq() = 0;
00846
00848 virtual L4::Cap<L4::Irq> device_notify_irq() const
00849 {
00850 L4Re::chksys(-L4_ENOSYS, "Legacy single IRQ interface not implemented.");
00851 return L4::Cap<L4::Irq>();
00852 }
00853
00860 virtual void register_driver_irq(unsigned idx)
00861 {
00862 if (idx != 0)
00863 L4Re::chksys(-L4_ENOSYS, "Multi IRQ interface not implemented.");
00864
00865 register_single_driver_irq();
00866 }
00867
00874 virtual L4::Cap<L4::Irq> device_notify_irq(unsigned idx)
00875 {
00876 if (idx != 0)
00877 L4Re::chksys(-L4_ENOSYS, "Multi IRQ interface not implemented.");
00878
00879 return device_notify_irq();
00880 }
00881
00883 virtual unsigned num_events_supported() const
00884 { return 1; }
00885
00886 virtual L4::Ipc_svr::Server_iface *server_iface() const = 0;
00887
00891 Device_t(Dev_config *dev_config)
00892 : _device_config(dev_config)
00893 {}
00894
00898 Mem_list const *mem_info() const
00899 { return &_mem_info; };
00900
00901 long op_set_status(L4virtio::Device::Rights, unsigned status)
00902 { return _set_status(status); }
00903
00904 long op_config_queue(L4virtio::Device::Rights, unsigned queue)
00905 {
00906 Dev_config::Status status = _device_config->status();
00907 if (status.fail_state() || !status.acked() || !status.driver())
00908 return -L4_EIO;
00909
00910 return reconfig_queue(queue);

```

```

00911 }
00912
00913 long op_register_ds(L4virtio::Device::Rights,
00914 L4::Ipc::Snd_fpage ds_cap_fp, l4_uint64_t ds_base,
00915 l4_umword_t offset, l4_umword_t sz)
00916 {
00917 L4Re::Util::Dbg()
00918 .printf("Registering dataspace from 0x%llx with %lu KiB, offset 0x%lx\n",
00919 ds_base, sz » 10, offset);
00920
00921 _check_n_init_shm(ds_cap_fp, ds_base, sz, offset);
00922
00923 return 0;
00924 }
00925
00926 long op_device_config(L4virtio::Device::Rights,
00927 L4::Ipc::Cap<L4Re::Dataspace> &config_ds,
00928 l4_addr_t &ds_offset)
00929 {
00930 L4Re::Util::Dbg()
00931 .printf("register client: host IRQ: %lx config DS: %lx\n",
00932 device_notify_irq().cap(), _device_config->ds().cap());
00933
00934 config_ds = L4::Ipc::make_cap(_device_config->ds(), L4_CAP_FPAGE_RW);
00935 ds_offset = _device_config->ds_offset();
00936 return 0;
00937 }
00938
00939 long op_device_notification_irq(L4virtio::Device::Rights,
00940 unsigned idx,
00941 L4::Ipc::Cap<L4::Triggerable> &irq)
00942 {
00943 auto cap = device_notify_irq(idx);
00944
00945 if (!cap.is_valid())
00946 return -L4_EINVAL;
00947
00948 irq = L4::Ipc::make_cap(cap, L4_CAP_FPAGE_RO);
00949 return L4_EOK;
00950 }
00951
00952 int op_bind(L4::Icu::Rights, l4_umword_t idx, L4::Ipc::Snd_fpage irq_cap_fp)
00953 {
00954 if (idx >= num_events_supported())
00955 return -L4_ERANGE;
00956
00957 if (!irq_cap_fp.cap_received())
00958 return -L4_EINVAL;
00959
00960 register_driver_irq(idx);
00961
00962 return L4_EOK;
00963 }
00964
00965 int op_unbind(L4::Icu::Rights, l4_umword_t, L4::Ipc::Snd_fpage)
00966 {
00967 return -L4_ENOSYS;
00968 }
00969
00970 int op_info(L4::Icu::Rights, L4::Icu::_Info &info)
00971 {
00972 info.features = 0;
00973 info.nr_irqs = num_events_supported();
00974 info.nr_msis = 0;
00975
00976 return L4_EOK;
00977 }
00978
00979 int op_msi_info(L4::Icu::Rights, l4_umword_t, l4_uint64_t, l4_icu_msi_info_t &)
00980 { return -L4_ENOSYS; }
00981
00982 int op_mask(L4::Icu::Rights, l4_umword_t)
00983 { return -L4_ENOSYS; }
00984
00985 int op_unmask(L4::Icu::Rights, l4_umword_t)
00986 { return -L4_ENOREPLY; }
00987
00988 int op_set_mode(L4::Icu::Rights, l4_umword_t, l4_umword_t)
00989 { return -L4_ENOSYS; }
00990
01002 void reset_queue_config(unsigned idx, unsigned num_max,
01003 bool inc_generation = false)
01004 {
01005 _device_config->reset_queue(idx, num_max, inc_generation);
01006 }
01007
01012 void init_mem_info(unsigned num)

```



```

01013 {
01014 _mem_info.init(num);
01015 }
01016
01024 void device_error()
01025 {
01026 reset();
01027 _device_config->set_device_needs_reset();
01028
01029 // the device MUST NOT notify the driver before DRIVER_OK.
01030 if (_device_config->status().driver_ok())
01031 trigger_driver_config_irq();
01032 }
01033
01047 bool setup_queue(Virtqueue *q, unsigned qn, unsigned num_max)
01048 {
01049 l4virtio_config_queue_t volatile const *qc;
01050 qc = _device_config->qconfig(qn);
01051 if (L4_UNLIKELY(qc == 0))
01052 return false;
01053
01054 if (!qc->ready)
01055 {
01056 q->disable();
01057 return true;
01058 }
01059
01060 // read to local variables before check
01061 l4_uint32_t num = qc->num;
01062 l4_uint64_t desc = qc->desc_addr;
01063 l4_uint64_t avail = qc->avail_addr;
01064 l4_uint64_t used = qc->used_addr;
01065
01066 if (0)
01067 printf("%p: setup queue: num=0x%x max_num=0x%x desc=0x%llx avail=0x%llx used=0x%llx\n",
01068 this, num, num_max, desc, avail, used);
01069
01070 if (!num || num > num_max)
01071 return false;
01072
01073 // num must be power of two
01074 if (num & (num - 1))
01075 return false;
01076
01077 if (desc & 0xf)
01078 return false;
01079
01080 if (avail & 0x1)
01081 return false;
01082
01083 if (used & 0x3)
01084 return false;
01085
01086 auto const *desc_info = _mem_info.find(desc, Virtqueue::desc_size(num));
01087 if (L4_UNLIKELY(!desc_info))
01088 return false;
01089
01090 auto const *avail_info = _mem_info.find(avail, Virtqueue::avail_size(num));
01091 if (L4_UNLIKELY(!avail_info))
01092 return false;
01093
01094 auto const *used_info = _mem_info.find(used, Virtqueue::used_size(num));
01095 if (L4_UNLIKELY(!used_info || !used_info->is_writable()))
01096 return false;
01097
01098 L4Re::Util::Dbg()
01099 .printf("shm=[%llx-%llx] local=[%lx-%lx] desc=[%llx-%llx] (%p-%p)\n",
01100 desc_info->drv_base(), desc_info->drv_base() + desc_info->size() - 1,
01101 desc_info->local_base(),
01102 desc_info->local_base() + desc_info->size() - 1,
01103 desc, desc + Virtqueue::desc_size(num),
01104 desc_info->local(Ptr<char>(desc)),
01105 desc_info->local(Ptr<char>(desc)) + Virtqueue::desc_size(num));
01106
01107 L4Re::Util::Dbg()
01108 .printf("shm=[%llx-%llx] local=[%lx-%lx] avail=[%llx-%llx] (%p-%p)\n",
01109 avail_info->drv_base(), avail_info->drv_base() + avail_info->size() - 1,
01110 avail_info->local_base(),
01111 avail_info->local_base() + avail_info->size() - 1,
01112 avail, avail + Virtqueue::avail_size(num),
01113 avail_info->local(Ptr<char>(avail)),
01114 avail_info->local(Ptr<char>(avail)) + Virtqueue::avail_size(num));
01115
01116 L4Re::Util::Dbg()
01117 .printf("shm=[%llx-%llx] local=[%lx-%lx] used=[%llx-%llx] (%p-%p)\n",
01118 used_info->drv_base(), used_info->drv_base() + used_info->size() - 1,
01119 used_info->local_base(),

```

```

01120 used_info->local_base() + used_info->size() - 1,
01121 used, used + Virtqueue::used_size(num),
01122 used_info->local(Ptr<char>(used)),
01123 used_info->local(Ptr<char>(used)) + Virtqueue::used_size(num));
01124
01125 q->setup(num, desc_info->local(Ptr<void>(desc)),
01126 avail_info->local(Ptr<void>(avail)),
01127 used_info->local(Ptr<void>(used)));
01128 return true;
01129 }
01130
01131 void check_n_init_shm(L4Re::Util::Unique_cap<L4Re::Dataspace> &shm,
01132 l4_uint64_t base, l4_umword_t size, l4_addr_t offset)
01133 {
01134 if (_mem_info.full())
01135 L4Re::chksys(-L4_ENOMEM);
01136
01137 auto const *i = _mem_info.add(base, size, offset, cxx::move(shm));
01138 L4Re::Util::Dbg()
01139 .printf("PORT[%p]: DMA guest [%llx-%llx] local [%lx-%lx] offset %lx\n",
01140 this, i->drv_base(), i->drv_base() + i->size() - 1,
01141 i->local_base(),
01142 i->local_base() + i->size() - 1,
01143 i->ds_offset());
01144 }
01145
01154 bool handle_mem_cmd_write()
01155 {
01156 l4_uint32_t cmd = _device_config->get_cmd();
01157 if (L4_LIKELY(!(cmd & L4VIRTIO_CMD_MASK)))
01158 return false;
01159
01160 switch (cmd & L4VIRTIO_CMD_MASK)
01161 {
01162 case L4VIRTIO_CMD_SET_STATUS:
01163 _set_status(cmd & ~L4VIRTIO_CMD_MASK);
01164 break;
01165
01166 case L4VIRTIO_CMD_CFG_QUEUE:
01167 reconfig_queue(cmd & ~L4VIRTIO_CMD_MASK);
01168 break;
01169
01170 case L4VIRTIO_CMD_CFG_CHANGED:
01171 cfg_changed(cmd & ~L4VIRTIO_CMD_MASK);
01172 break;
01173
01174 default:
01175 // unknown command
01176 break;
01177 }
01178
01179 _device_config->reset_cmd();
01180
01181 return true;
01182 }
01183
01187 void enable_trusted_ds_validation()
01188 {
01189 _trusted_ds_validation_enabled = true;
01190 }
01191
01197 void
01198 add_trusted_dataspaces(std::shared_ptr<Ds_vector const> ds)
01199 {
01200 _trusted_ds_caps = ds;
01201 }
01202
01203
01204 private:
01216 long validate_ds(L4::Cap<L4Re::Dataspace> ds)
01217 {
01218 if (!_trusted_ds_caps)
01219 return -L4_EINVAL;
01220 if (std::any_of(_trusted_ds_caps->cbegin(), _trusted_ds_caps->cend(),
01221 [&ds] (L4::Cap<L4Re::Dataspace> cap)
01222 {
01223 return L4Re::Env::env()->task()
01224 ->cap_equal(ds, cap).label() == 1;
01225 })
01226)
01227 {
01228 return L4_EOK;
01229 }
01230 return -L4_EINVAL;
01231 }
01232
01233 void _check_n_init_shm(L4::Ipc::Snd_fpage shm_cap_fp,

```

```

01234 l4_uint64_t base, l4_umword_t size, l4_addr_t offset)
01235 {
01236 if (!shm_cap_fp.cap_received())
01237 L4Re::chksys(-L4_EINVAL);
01238
01239 L4Re::Util::Unique_cap<L4Re::Dataspace> ds(
01240 L4Re::chkcap(server_iface()->template rcv_cap<L4Re::Dataspace>(0)));
01241 L4Re::chksys(server_iface()->realloc_rcv_cap(0));
01242
01243 if (_trusted_ds_validation_enabled)
01244 L4Re::chksys(validate_ds(ds.get()), "Validating the dataspace.");
01245
01246 check_n_init_shm(cxx::move(ds), base, size, offset);
01247 }
01248
01249 bool check_features_internal()
01250 {
01251 static_assert(sizeof(l4virtio_config_hdr_t::driver_features_map)
01252 == sizeof(l4virtio_config_hdr_t::dev_features_map),
01253 "Driver and device feature maps must be of the same size");
01254
01255 // From the Virtio 1.0 specification 6.1 Driver Requirements and 6.2 Device
01256 // Requirements: A driver MUST accept VIRTIO_F_VERSION_1 if it is offered.
01257 // A device MUST offer VIRTIO_F_VERSION_1. A device MAY fail to operate
01258 // further if VIRTIO_F_VERSION_1 is not accepted.
01259 //
01260 // The L4virtio implementation does not support legacy interfaces so we
01261 // fail here if the Virtio 1.0 feature was not accepted.
01262 if (!_device_config->get_guest_feature(L4VIRTIO_FEATURE_VERSION_1))
01263 return false;
01264
01265 for (auto i = 0u;
01266 i < sizeof(l4virtio_config_hdr_t::driver_features_map)
01267 / sizeof(l4virtio_config_hdr_t::driver_features_map[0]);
01268 i++)
01269 {
01270 // Driver must not accept features that were not offered by device
01271 if (_device_config->guest_features(i)
01272 & ~_device_config->host_features(i))
01273 return false;
01274 }
01275 return check_features();
01276 }
01277
01299 void check_and_update_status(Dev_config::Status status)
01300 {
01301 // snapshot of current status
01302 Dev_config::Status current_status = _device_config->status();
01303
01304 // handle reset
01305 if (!status.raw)
01306 {
01307 _device_config->set_status(status);
01308 return;
01309 }
01310
01311 // Do no further processing in case of driver or device failure. If FAILED
01312 // or DEVICE_NEEDS_RESET are set only these fail_state bits will be set in
01313 // addition to the current status bits already set.
01314 if (current_status.fail_state() || status.fail_state())
01315 {
01316 if (current_status.fail_state() != status.fail_state())
01317 {
01318 current_status.fail_state() =
01319 current_status.fail_state() | status.fail_state();
01320 _device_config->set_status(current_status);
01321 }
01322 return;
01323 }
01324
01325 // Enforce init sequence ACKNOWLEDGE, DRIVER, FEATURES_OK, DRIVER_OK.
01326 // We do not enforce that only one additional new bit is set per call.
01327 if ((!status.acked() && status.driver())
01328 || (!status.driver() && status.features_ok())
01329 || (!status.features_ok() && status.driver_ok()))
01330 {
01331 current_status.device_needs_reset() = 1;
01332 _device_config->set_status(current_status);
01333 return;
01334 }
01335
01336 // only check feature compatibility before DRIVER_OK is set
01337 if (status.features_ok() && !status.driver_ok()
01338 && !check_features_internal())
01339 status.features_ok() = 0;
01340
01341 // Note that if FEATURES_OK and DRIVER_OK are both updated to being set

```

```

01342 // at the same time the above check_features_internal() is skipped; this is
01343 // considered undefined behaviour but it is not prevented.
01344 if (status.running() && !check_queues())
01345 {
01346 current_status.device_needs_reset() = 1;
01347 _device_config->set_status(current_status);
01348 return;
01349 }
01350
01351 _device_config->set_status(status);
01352 }
01353
01354 int _set_status(unsigned new_status)
01355 {
01356 if (new_status == 0)
01357 {
01358 L4Re::Util::Dbg().printf("Resetting device\n");
01359 reset();
01360 _device_config->reset_hdr(true);
01361 }
01362
01363 Dev_config::Status status(new_status);
01364 check_and_update_status(status);
01365
01366 return 0;
01367 }
01368
01369 };
01370
01371 typedef Device_t<No_custom_data> Device;
01372
01373 } // namespace Svr
01374
01375 }

```

## 17.253 virtio

```

00001 // vi:ft=cpp
00002 /* SPDX-License-Identifier: MIT */
00003 /*
00004 * Copyright (C) 2014-2020, 2023-2024 Kernkonzept GmbH.
00005 * Author(s): Alexander Warg <alexander.warg@kernkonzept.com>
00006 *
00007 */
00008 #pragma once
00009
00010 #include <l4/sys/types.h>
00011 #include <l4/cxx/bitfield>
00012 #include <l4/cxx/minmax>
00013 #include <l4/cxx/utils>
00014
00015 #include <limits.h>
00016 #include <string.h>
00017 #include <stdio.h>
00018
00019 #include "../virtqueue"
00020
00021 namespace L4virtio {
00022 namespace Svr {
00023
00024 struct Dev_status
00025 {
00026 unsigned char raw;
00027 Dev_status() = default;
00028
00029 explicit Dev_status(l4_uint32_t v) : raw(v) {}
00030
00031 CXX_BITFIELD_MEMBER(0, 0, acked, raw);
00032 CXX_BITFIELD_MEMBER(1, 1, driver, raw);
00033 CXX_BITFIELD_MEMBER(2, 2, driver_ok, raw);
00034 CXX_BITFIELD_MEMBER(3, 3, features_ok, raw);
00035 CXX_BITFIELD_MEMBER(6, 7, fail_state, raw);
00036 CXX_BITFIELD_MEMBER(6, 6, device_needs_reset, raw);
00037 CXX_BITFIELD_MEMBER(7, 7, failed, raw);
00038
00039 bool running() const
00040 {
00041 return (raw == 0xf);
00042 }
00043 };
00044
00045 struct Dev_features
00046 {

```

```

00068 l4_uint32_t raw;
00069 Dev_features() = default;
00070
00072 explicit Dev_features(l4_uint32_t v) : raw(v) {}
00073
00074 CXX_BITFIELD_MEMBER(28, 28, ring_indirect_desc, raw);
00075 CXX_BITFIELD_MEMBER(29, 29, ring_event_idx, raw);
00076 };
00077
00078
00087 class Virtqueue : public L4virtio::Virtqueue
00088 {
00089 public:
00093 class Head_desc
00094 {
00095 friend class Virtqueue;
00096 private:
00097 Virtqueue::Desc const *_d;
00098 Head_desc(Virtqueue *r, unsigned i) : _d(r->desc(i)) {}
00099
00100 public:
00102 Head_desc() : _d(0) {}
00103
00105 bool valid() const { return _d; }
00106
00108 explicit operator bool () const
00109 { return valid(); }
00110
00112 Desc const *desc() const
00113 { return _d; }
00114 };
00115
00116 struct Request : Head_desc
00117 {
00118 Virtqueue *ring = nullptr;
00119 Request() = default;
00120 private:
00121 friend class Virtqueue;
00122 Request(Virtqueue *r, unsigned i) : Head_desc(r, i), ring(r) {}
00123 };
00124
00125
00136 Request next_avail()
00137 {
00138 if (L4_LIKELY(_current_avail != _avail->idx))
00139 {
00140 rmb();
00141 unsigned head = _current_avail & _idx_mask;
00142 ++_current_avail;
00143 return Request(this, _avail->ring[head]);
00144 }
00145 return Request();
00146 }
00147
00160 void rewind_avail(Head_desc const &d)
00161 {
00162 unsigned head_idx = d._d - _desc;
00163 // Calculate the distance between _current_avail and head_idx, taking into
00164 // account that _current_avail might have wrapped around with respect to
00165 // _idx_mask in the meantime.
00166 _current_avail -= (_current_avail - head_idx) & _idx_mask;
00167 }
00168
00175 bool desc_avail() const
00176 {
00177 return _current_avail != _avail->idx;
00178 }
00179
00190 void consumed(Head_desc const &r, l4_uint32_t len = 0)
00191 {
00192 l4_uint16_t i = _used->idx & _idx_mask;
00193 _used->ring[i] = Used_elem(r._d - _desc, len);
00194 wmb();
00195 ++_used->idx;
00196 }
00197
00212 template<typename ITER>
00213 void consumed(ITER const &begin, ITER const &end)
00214 {
00215 l4_uint16_t added = 0;
00216 l4_uint16_t idx = _used->idx;
00217
00218 for (auto elem = begin ; elem != end; ++elem, ++added)
00219 _used->ring[(idx + added) & _idx_mask]
00220 = Used_elem(elem->first._d - _desc, elem->second);
00221
00222 wmb();

```

```

00223 __used->idx += added;
00224 }
00225
00239 template<typename QUEUE_OBSERVER>
00240 void finish(Head_desc &d, QUEUE_OBSERVER *o, l4_uint32_t len = 0)
00241 {
00242 consumed(d, len);
00243 o->notify_queue(this);
00244 d._d = 0;
00245 }
00246
00261 template<typename ITER, typename QUEUE_OBSERVER>
00262 void finish(ITER const &begin, ITER const &end, QUEUE_OBSERVER *o)
00263 {
00264 consumed(begin, end);
00265 o->notify_queue(this);
00266 }
00267
00273 void disable_notify()
00274 {
00275 if (L4_LIKELY(ready()))
00276 __used->flags.no_notify() = 1;
00277 }
00278
00284 void enable_notify()
00285 {
00286 if (L4_LIKELY(ready()))
00287 __used->flags.no_notify() = 0;
00288 }
00289
00298 Desc const *desc(unsigned idx) const
00299 { return __desc + idx; }
00300
00301 };
00302
00306 struct Data_buffer
00307 {
00308 char *pos;
00309 l4_uint32_t left;
00310
00311 Data_buffer() = default;
00312
00322 template<typename T>
00323 explicit Data_buffer(T *p)
00324 : pos(reinterpret_cast<char *>(p)), left(sizeof(T))
00325 {}
00326
00336 template<typename T>
00337 void set(T *p)
00338 {
00339 pos = reinterpret_cast<char *>(p);
00340 left = sizeof(T);
00341 }
00342
00354 l4_uint32_t copy_to(Data_buffer *dst, l4_uint32_t max = UINT_MAX)
00355 {
00356 unsigned long bytes = cxx::min(cxx::min(left, dst->left), max);
00357 memcpy(dst->pos, pos, bytes);
00358 left -= bytes;
00359 pos += bytes;
00360 dst->left -= bytes;
00361 dst->pos += bytes;
00362 return bytes;
00363 }
00364
00375 l4_uint32_t skip(l4_uint32_t bytes)
00376 {
00377 unsigned long b = cxx::min(left, bytes);
00378 left -= b;
00379 pos += b;
00380 return b;
00381 }
00382
00388 bool done() const
00389 { return left == 0; }
00390 };
00391
00392 class Request_processor;
00393
00397 struct Bad_descriptor
00398 {
00400 enum Error
00401 {
00402 Bad_address,
00403 Bad_rights,
00404 Bad_flags,
00405 Bad_next,

```

```

00406 Bad_size
00407 };
00408
00410 Request_processor const *proc;
00411
00412 // The error code
00413 Error error;
00414
00421 Bad_descriptor(Request_processor const *proc, Error e)
00422 : proc(proc), error(e)
00423 {}
00424
00430 char const *message() const
00431 {
00432 static char const *const err[] =
00433 {
00434 [Bad_address] = "Descriptor address cannot be translated",
00435 [Bad_rights] = "Insufficient memory access rights",
00436 [Bad_flags] = "Invalid descriptor flags",
00437 [Bad_next] = "The descriptor's `next` index is invalid",
00438 [Bad_size] = "Invalid size of the memory block"
00439 };
00440
00441 if (error >= (sizeof(err) / sizeof(err[0])) || !err[error])
00442 return "Unknown error";
00443
00444 return err[error];
00445 }
00446 };
00447
00448
00472 class Request_processor
00473 {
00474 private:
00476 Virtqueue::Desc const *_table;
00477
00479 Virtqueue::Desc _current;
00480
00482 l4_uint16_t _num;
00483
00484 public:
00500 template<typename DESC_MAN, typename ...ARGS>
00501 void start(DESC_MAN *dm, Virtqueue *ring, Virtqueue::Head_desc const &request, ARGS... args)
00502 {
00503 _current = cxx::access_once(request.desc());
00504
00505 if (_current.flags.indirect())
00506 {
00507 dm->load_desc(_current, this, &_table);
00508 _num = _current.len / sizeof(Virtqueue::Desc);
00509 if (L4_UNLIKELY(!_num))
00510 throw Bad_descriptor(this, Bad_descriptor::Bad_size);
00511
00512 _current = cxx::access_once(_table);
00513 }
00514 else
00515 {
00516 _table = ring->desc(0);
00517 _num = ring->num();
00518 }
00519
00520 dm->load_desc(_current, this, cxx::forward<ARGS>(args)...);
00521 }
00522
00533 template<typename DESC_MAN, typename ...ARGS>
00534 Virtqueue::Request const &start(DESC_MAN *dm, Virtqueue::Request const &request, ARGS... args)
00535 {
00536 start(dm, request.ring, request, cxx::forward<ARGS>(args)...);
00537 return request;
00538 }
00539
00545 Virtqueue::Desc::Flags current_flags() const
00546 { return _current.flags; }
00547
00553 bool has_more() const
00554 { return _current.flags.next(); }
00555
00569 template<typename DESC_MAN, typename ...ARGS>
00570 bool next(DESC_MAN *dm, ARGS... args)
00571 {
00572 if (!_current.flags.next())
00573 return false;
00574
00575 if (L4_UNLIKELY(_current.next >= _num))
00576 throw Bad_descriptor(this, Bad_descriptor::Bad_next);
00577
00578 _current = cxx::access_once(_table + _current.next);

```

```

00579
00580 if (0) // we ignore this for performance reasons
00581 if (L4_UNLIKELY(_current.flags.indirect()))
00582 throw Bad_descriptor(this, Bad_descriptor::Bad_flags);
00583
00584 // must throw an exception in case of a bad descriptor
00585 dm->load_desc(_current, this, cxx::forward<ARGS>(args)...);
00586 return true;
00587 }
00588 };
00589
00590 }
00591 }

```

## 17.254 virtio-block

```

00001 // vi:ft=cpp
00002 /* SPDX-License-Identifier: MIT */
00003 /*
00004 * Copyright (C) 2015-2022, 2024 Kernkonzept GmbH.
00005 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00006 * Manuel von Oltersdorff-Kalettka <manuel.kalettka@kernkonzept.com>
00007 *
00008 */
00009 #pragma once
00010
00011 #include <l4/sys/factory>
00012 #include <l4/sys/semaphore>
00013 #include <l4/re/dataspace>
00014 #include <l4/re/env>
00015 #include <l4/re/util/unique_cap>
00016 #include <l4/re/util/object_registry>
00017 #include <l4/re/error_helper>
00018
00019 #include <l4/util/atomic.h>
00020 #include <l4/util/bitops.h>
00021 #include <l4/l4virtio/client/l4virtio>
00022 #include <l4/l4virtio/l4virtio>
00023 #include <l4/l4virtio/virtqueue>
00024 #include <l4/l4virtio/virtio_block.h>
00025 #include <l4/sys/consts.h>
00026
00027 #include <cstring>
00028 #include <vector>
00029 #include <functional>
00030
00031 namespace L4virtio { namespace Driver {
00032
00036 class Block_device : public Device
00037 {
00038 public:
00039 typedef std::function<void(unsigned char)> Callback;
00040
00041 private:
00042 enum { Header_size = sizeof(l4virtio_block_header_t) };
00043
00044 struct Request
00045 {
00046 l4_uint16_t tail;
00047 Callback callback;
00048
00049 Request() : tail(Virtqueue::Eq), callback(0) {}
00050 };
00051
00052 public:
00056 class Handle
00057 {
00058 {
00059 friend Block_device;
00060 l4_uint16_t head;
00061
00062 explicit Handle(l4_uint16_t descno) : head(descno) {}
00063
00064 public:
00065 Handle() : head(Virtqueue::Eq) {}
00066 bool valid() const { return head != Virtqueue::Eq; }
00067 };
00068
00092 void setup_device(L4::Cap<L4virtio::Device> srvcap, l4_size_t usermem,
00093 void **userdata, Ptr<void> &user_devaddr,
00094 L4::Cap<L4Re::Dataspace> qds = L4::Cap<L4Re::Dataspace>(),
00095 l4_uint32_t fmask0 = -1U, l4_uint32_t fmask1 = -1U)
00096 {
00097 // Contact device.

```



```

00098 driver_connect(srvcap);
00099
00100 if (_config->device != L4VIRTIO_ID_BLOCK)
00101 L4Re::chksys(-L4_ENODEV, "Device is not a block device.");
00102
00103 if (_config->num_queues != 1)
00104 L4Re::chksys(-L4_EINVAL, "Invalid number of queues reported.");
00105
00106 // Memory is shared in one large dataspace which contains queues,
00107 // space for header/status and additional user-defined memory.
00108 unsigned queuesz = max_queue_size(0);
00109 l4_size_t totalsz = l4_round_page(usermem);
00110
00111 l4_uint64_t const header_offset =
00112 l4_round_size(_queue.total_size(queuesz),
00113 l4util_bsr(aligned(l4virtio_block_header_t)));
00114 l4_uint64_t const status_offset = header_offset + queuesz * Header_size;
00115 l4_uint64_t const usermem_offset = l4_round_page(status_offset + queuesz);
00116
00117 // reserve space for one header/status per descriptor
00118 // TODO Should be reduced to 1/3 but this way no freelist is needed.
00119 totalsz += usermem_offset;
00120
00121 auto *e = L4Re::Env::env();
00122 if (!qds.is_valid())
00123 {
00124 _ds = L4Re::chkcap(L4Re::Util::make_unique_cap<L4Re::Dataspace>(),
00125 "Allocate queue dataspace capability");
00126 L4Re::chksys(e->mem_alloc()->alloc(totalsz, _ds.get(),
00127 L4Re::Mem_alloc::Continuous
00128 | L4Re::Mem_alloc::Pinned),
00129 "Allocate memory for virtio structures");
00130 _queue_ds = _ds.get();
00131 }
00132 else
00133 {
00134 if (qds->size() < totalsz)
00135 L4Re::chksys(-L4_EINVAL, "External queue dataspace too small.");
00136 _queue_ds = qds;
00137 }
00138
00139 // Now sort out which region goes where in the dataspace.
00140 L4Re::chksys(e->rm()->attach(&_queue_region, totalsz,
00141 L4Re::Rm::F::Search_addr | L4Re::Rm::F::RW,
00142 L4::Ipc::make_cap_rw(_queue_ds), 0,
00143 L4_PAGESHIFT),
00144 "Attach dataspace for virtio structures");
00145
00146 l4_uint64_t devaddr;
00147 L4Re::chksys(register_ds(_queue_ds, 0, totalsz, &devaddr),
00148 "Register queue dataspace with device");
00149
00150 _queue.init_queue(queuesz, _queue_region.get());
00151
00152 config_queue(0, queuesz, devaddr, devaddr + _queue.avail_offset(),
00153 devaddr + _queue.used_offset());
00154
00155 _header_addr = devaddr + header_offset;
00156 _headers = reinterpret_cast<l4virtio_block_header_t *>(_queue_region.get()
00157 + header_offset);
00158
00159 _status_addr = devaddr + status_offset;
00160 _status = _queue_region.get() + status_offset;
00161
00162 user_devaddr = Ptr<void>(devaddr + usermem_offset);
00163 if (userdata)
00164 *userdata = _queue_region.get() + usermem_offset;
00165
00166 // setup the callback mechanism
00167 _pending.assign(queuesz, Request());
00168
00169 // Finish handshake with device.
00170 _config->driver_features_map[0] = fmask0;
00171 _config->driver_features_map[1] = fmask1;
00172 driver_acknowledge();
00173 }
00174
00178 l4virtio_block_config_t const &device_config() const
00179 {
00180 return *_config->device_config<l4virtio_block_config_t>();
00181 }
00182
00191 Handle start_request(l4_uint64_t sector, l4_uint32_t type,
00192 Callback callback)
00193 {
00194 l4_uint16_t descno = _queue.alloc_descriptor();
00195 if (descno == Virtqueue::Eoq)

```

```

00196 return Handle(Virtqueue::Eq);
00197
00198 L4virtio::Virtqueue::Desc &desc = _queue.desc(descno);
00199 Request &req = _pending[descno];
00200
00201 // setup the header
00202 L4virtio_block_header_t &head = _headers[descno];
00203 head.type = type;
00204 head.ioprio = 0;
00205 head.sector = sector;
00206
00207 // and put it in the descriptor
00208 desc.addr = Ptr<void>(_header_addr + descno * Header_size);
00209 desc.len = Header_size;
00210 desc.flags.raw = 0; // no write, no indirect
00211
00212 req.tail = descno;
00213 req.callback = callback;
00214
00215 return Handle(descno);
00216 }
00217
00229 int add_block(Handle handle, Ptr<void> addr, l4_uint32_t size)
00230 {
00231 l4_uint16_t descno = _queue.alloc_descriptor();
00232 if (descno == Virtqueue::Eq)
00233 return -L4_EAGAIN;
00234
00235 Request &req = _pending[handle.head];
00236 L4virtio::Virtqueue::Desc &desc = _queue.desc(descno);
00237 L4virtio::Virtqueue::Desc &prev = _queue.desc(req.tail);
00238
00239 prev.next = descno;
00240 prev.flags.next() = true;
00241
00242 desc.addr = addr;
00243 desc.len = size;
00244 desc.flags.raw = 0;
00245 if (_headers[handle.head].type > 0) // write or flush request
00246 desc.flags.write() = true;
00247
00248 req.tail = descno;
00249
00250 return L4_EOK;
00251 }
00252
00265 int send_request(Handle handle)
00266 {
00267 // add the status bit
00268 auto descno = _queue.alloc_descriptor();
00269 if (descno == Virtqueue::Eq)
00270 return -L4_EAGAIN;
00271
00272 Request &req = _pending[handle.head];
00273 L4virtio::Virtqueue::Desc &desc = _queue.desc(descno);
00274 L4virtio::Virtqueue::Desc &prev = _queue.desc(req.tail);
00275
00276 prev.next = descno;
00277 prev.flags.next() = true;
00278
00279 desc.addr = Ptr<void>(_status_addr + descno);
00280 desc.len = 1;
00281 desc.flags.raw = 0;
00282 desc.flags.write() = true;
00283
00284 req.tail = descno;
00285
00286 send(_queue, handle.head);
00287
00288 return L4_EOK;
00289 }
00290
00306 int process_request(Handle handle)
00307 {
00308 // add the status bit
00309 auto descno = _queue.alloc_descriptor();
00310 if (descno == Virtqueue::Eq)
00311 return -L4_EAGAIN;
00312
00313 L4virtio::Virtqueue::Desc &desc = _queue.desc(descno);
00314 L4virtio::Virtqueue::Desc &prev = _queue.desc(_pending[handle.head].tail);
00315
00316 prev.next = descno;
00317 prev.flags.next() = true;
00318
00319 desc.addr = Ptr<void>(_status_addr + descno);
00320 desc.len = 1;

```

```

00321 desc.flags.raw = 0;
00322 desc.flags.write() = true;
00323
00324 _pending[handle.head].tail = descno;
00325
00326 int ret = send_and_wait(_queue, handle.head);
00327 unsigned char status = _status[descno];
00328 free_request(handle);
00329
00330 if (ret < 0)
00331 return ret;
00332
00333 switch (status)
00334 {
00335 case L4VIRTIO_BLOCK_S_OK: return L4_EOK;
00336 case L4VIRTIO_BLOCK_S_IOERR: return -L4_EIO;
00337 case L4VIRTIO_BLOCK_S_UNSUPP: return -L4_ENOSYS;
00338 }
00339
00340 return -L4_EINVAL;
00341 }
00342
00343 void free_request(Handle handle)
00344 {
00345 if (handle.head != Virtqueue::Eoq
00346 && _pending[handle.head].tail != Virtqueue::Eoq)
00347 _queue.free_descriptor(handle.head, _pending[handle.head].tail);
00348 _pending[handle.head].tail = Virtqueue::Eoq;
00349 }
00350
00357 void process_used_queue()
00358 {
00359 for (l4_uint16_t descno = _queue.find_next_used();
00360 descno != Virtqueue::Eoq;
00361 descno = _queue.find_next_used()
00362)
00363 {
00364 if (descno >= _queue.num() || _pending[descno].tail == Virtqueue::Eoq)
00365 L4Re::chksys(-L4_ENOSYS, "Bad descriptor number");
00366
00367 unsigned char status = _status[descno];
00368 free_request(Handle(descno));
00369
00370 if (_pending[descno].callback)
00371 _pending[descno].callback(status);
00372 }
00373 }
00374
00375 protected:
00376 L4Re::Util::Unique_cap<L4Re::Dataspace> _ds;
00377 L4::Cap<L4Re::Dataspace> _queue_ds;
00378
00379 private:
00380 L4Re::Rm::Unique_region<unsigned char *> _queue_region;
00381 l4virtio_block_header_t *_headers;
00382 unsigned char *_status;
00383 l4_uint64_t _header_addr;
00384 l4_uint64_t _status_addr;
00385 Virtqueue _queue;
00386 std::vector<Request> _pending;
00387 };
00388
00389 } }

```

## 17.255 virtio-block

```

00001 // vi:ft=cpp
00002 /* SPDX-License-Identifier: MIT */
00003 /*
00004 * Copyright (C) 2017-2021, 2024 Kernkonzept GmbH.
00005 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00006 *
00007 */
00008 #pragma once
00009
00010 #include <l4/cxx/unique_ptr>
00011 #include <l4/re/util/unique_cap>
00012
00013 #include <climits>
00014
00015 #include <l4/l4virtio/virtio.h>
00016 #include <l4/l4virtio/virtio_block.h>
00017 #include <l4/l4virtio/server/l4virtio>

```

```

00018 #include <l4/sys/cxx/ipc_epiface>
00019
00020 namespace L4virtio { namespace Svr {
00021
00022 template <typename Ds_data> class Block_dev_base;
00023
00027 template<typename Ds_data>
00028 class Block_request
00029 {
00030 friend class Block_dev_base<Ds_data>;
00031 enum { Header_size = sizeof(l4virtio_block_header_t) };
00032
00033 public:
00034 struct Data_block
00035 {
00037 Driver_mem_region_t<Ds_data> *mem;
00039 void *addr;
00041 l4_uint32_t len;
00042
00043 Data_block() = default;
00044
00045 Data_block(Driver_mem_region_t<Ds_data> *m, Virtqueue::Desc const &desc,
00046 Request_processor const *)
00047 : mem(m), addr(m->local(desc.addr)), len(desc.len)
00048 {}
00049 };
00050
00051
00052
00063 unsigned data_size() const
00064 {
00065 Request_processor rp;
00066 Data_block data;
00067
00068 rp.start(_mem_list, _request, &data);
00069
00070 unsigned total = data.len;
00071
00072 try
00073 {
00074 while (rp.has_more())
00075 {
00076 rp.next(_mem_list, &data);
00077 total += data.len;
00078 }
00079 }
00080 catch (Bad_descriptor const &e)
00081 {
00082 // need to convert the exception because e contains a raw pointer to rp
00083 throw L4::Runtime_error(-L4_EIO, "bad virtio descriptor");
00084 }
00085
00086 if (total < Header_size + 1)
00087 throw L4::Runtime_error(-L4_EIO, "virtio request too short");
00088
00089 return total - Header_size - 1;
00090 }
00091
00095 bool has_more()
00096 {
00097 // peek into the remaining data
00098 while (_data.len == 0 && _rp.has_more())
00099 _rp.next(_mem_list, &_data);
00100
00101 // there always must be one byte left for status
00102 return (_data.len > 1 || _rp.has_more());
00103 }
00104
00113 Data_block next_block()
00114 {
00115 Data_block out;
00116
00117 if (_data.len == 0)
00118 {
00119 if (!_rp.has_more())
00120 throw L4::Runtime_error(-L4_EEXIST,
00121 "No more data blocks in virtio request");
00122
00123 if (_todo_blocks == 0)
00124 throw Bad_descriptor(&_rp, Bad_descriptor::Bad_size);
00125 --_todo_blocks;
00126
00127 _rp.next(_mem_list, &_data);
00128 }
00129
00130 if (_data.len > _max_block_size)
00131 throw Bad_descriptor(&_rp, Bad_descriptor::Bad_size);

```

```

00132
00133 out = _data;
00134
00135 if (!_rp.has_more())
00136 {
00137 --(out.len);
00138 _data.len = 1;
00139 _data.addr = static_cast<char *>(_data.addr) + out.len;
00140 }
00141 else
00142 _data.len = 0; // is consumed
00143
00144 return out;
00145 }
00146
00148 l4virtio_block_header_t const &header() const
00149 { return _header; }
00150
00151 private:
00152 Block_request(Virtqueue::Request req, Driver_mem_list_t<Ds_data> *mem_list,
00153 unsigned max_blocks, l4_uint32_t max_block_size)
00154 : _mem_list(mem_list),
00155 _request(req),
00156 _todo_blocks(max_blocks),
00157 _max_block_size(max_block_size)
00158 {
00159 // read header which should be in the first block
00160 _rp.start(mem_list, _request, &_data);
00161 --_todo_blocks;
00162
00163 if (_data.len < Header_size)
00164 throw Bad_descriptor(&_rp, Bad_descriptor::Bad_size);
00165
00166 _header = *(static_cast<l4virtio_block_header_t *>(_data.addr));
00167
00168 _data.addr = static_cast<char *>(_data.addr) + Header_size;
00169 _data.len -= Header_size;
00170
00171 // if there is no space for status bit we cannot really recover
00172 if (!_rp.has_more() && _data.len == 0)
00173 throw Bad_descriptor(&_rp, Bad_descriptor::Bad_size);
00174 }
00175
00176 int release_request(Virtqueue *queue, l4_uint8_t status, unsigned sz)
00177 {
00178 // write back status
00179 // If there was an error on the way or the status byte is in its
00180 // own block, fast-forward to the last block.
00181 if (!_rp.has_more())
00182 {
00183 while (_rp.next(_mem_list, &_data) && _todo_blocks > 0)
00184 --_todo_blocks;
00185
00186 if (_todo_blocks > 0 && _data.len > 0)
00187 *(static_cast<l4_uint8_t *>(_data.addr) + _data.len - 1) = status;
00188 else
00189 return -L4_EIO; // too many data blocks
00190 }
00191 else if (_data.len > 0)
00192 *(static_cast<l4_uint8_t *>(_data.addr)) = status;
00193 else
00194 return -L4_EIO; // no space for final status byte
00195
00196 // now release the head
00197 queue->consumed(_request, sz);
00198
00199 return L4_EOK;
00200 }
00201
00207 Driver_mem_list_t<Ds_data> *_mem_list;
00209 l4virtio_block_header_t _header;
00211 Request_processor _rp;
00213 Data_block _data;
00214
00216 Virtqueue::Request _request;
00218 unsigned _todo_blocks;
00220 l4_uint32_t _max_block_size;
00221 };
00222
00223 struct Block_features : public Dev_config::Features
00224 {
00225 Block_features() = default;
00226 Block_features(l4_uint32_t raw) : Dev_config::Features(raw) {}
00227
00229 CXX_BITFIELD_MEMBER(1, 1, size_max, raw);
00231 CXX_BITFIELD_MEMBER(2, 2, seg_max, raw);
00233 CXX_BITFIELD_MEMBER(4, 4, geometry, raw);

```

```

00235 CXX_BITFIELD_MEMBER(5, 5, ro, raw);
00237 CXX_BITFIELD_MEMBER(6, 6, blk_size, raw);
00239 CXX_BITFIELD_MEMBER(9, 9, flush, raw);
00241 CXX_BITFIELD_MEMBER(10, 10, topology, raw);
00243 CXX_BITFIELD_MEMBER(11, 11, config_wce, raw);
00245 CXX_BITFIELD_MEMBER(12, 12, mq, raw);
00247 CXX_BITFIELD_MEMBER(13, 13, discard, raw);
00249 CXX_BITFIELD_MEMBER(14, 14, write_zeroes, raw);
00250 };
00251
00252
00258 template <typename Ds_data>
00259 class Block_dev_base : public L4virtio::Svr::Device_t<Ds_data>
00260 {
00261 private:
00262 L4Re::Util::Unique_cap<L4::Irq> _kick_guest_irq;
00263 Virtqueue _queue;
00264 unsigned _vq_max;
00265 l4_uint32_t _max_block_size = UINT_MAX;
00266 Dev_config_t<l4virtio_block_config_t> _dev_config;
00267
00268 public:
00269 typedef Block_request<Ds_data> Request;
00270
00271 protected:
00272 Block_features negotiated_features() const
00273 { return _dev_config.negotiated_features(0); }
00274
00275 Block_features device_features() const
00276 { return _dev_config.host_features(0); }
00277
00278 void set_device_features(Block_features df)
00279 { _dev_config.host_features(0) = df.raw; }
00280
00290 void set_size_max(l4_uint32_t sz)
00291 {
00292 _dev_config.priv_config()->size_max = sz;
00293 Block_features df = device_features();
00294 df.size_max() = true;
00295 set_device_features(df);
00296
00297 _max_block_size = sz;
00298 }
00299
00304 void set_seg_max(l4_uint32_t sz)
00305 {
00306 _dev_config.priv_config()->seg_max = sz;
00307 Block_features df = device_features();
00308 df.seg_max() = true;
00309 set_device_features(df);
00310 }
00311
00315 void set_geometry(l4_uint16_t cylinders, l4_uint8_t heads, l4_uint8_t sectors)
00316 {
00317 l4virtio_block_config_t volatile *pc = _dev_config.priv_config();
00318 pc->geometry.cylinders = cylinders;
00319 pc->geometry.heads = heads;
00320 pc->geometry.sectors = sectors;
00321 Block_features df = device_features();
00322 df.geometry() = true;
00323 set_device_features(df);
00324 }
00325
00332 void set_blk_size(l4_uint32_t sz)
00333 {
00334 _dev_config.priv_config()->blk_size = sz;
00335 Block_features df = device_features();
00336 df.blk_size() = true;
00337 set_device_features(df);
00338 }
00339
00348 void set_topology(l4_uint8_t physical_block_exp,
00349 l4_uint8_t alignment_offset,
00350 l4_uint32_t min_io_size,
00351 l4_uint32_t opt_io_size)
00352 {
00353 l4virtio_block_config_t volatile *pc = _dev_config.priv_config();
00354 pc->topology.physical_block_exp = physical_block_exp;
00355 pc->topology.alignment_offset = alignment_offset;
00356 pc->topology.min_io_size = min_io_size;
00357 pc->topology.opt_io_size = opt_io_size;
00358 Block_features df = device_features();
00359 df.topology() = true;
00360 set_device_features(df);
00361 }
00362
00364 void set_flush()

```

```

00365 {
00366 Block_features df = device_features();
00367 df.flush() = true;
00368 set_device_features(df);
00369 }
00370
00375 void set_config_wce(l4_uint8_t writeback)
00376 {
00377 l4virtio_block_config_t volatile *pc = _dev_config.priv_config();
00378 pc->writeback = writeback;
00379 Block_features df = device_features();
00380 df.config_wce() = true;
00381 set_device_features(df);
00382 }
00383
00388 l4_uint8_t get_writeback()
00389 {
00390 l4virtio_block_config_t volatile *pc = _dev_config.priv_config();
00391 return pc->writeback;
00392 }
00393
00402 void set_discard(l4_uint32_t max_discard_sectors, l4_uint32_t max_discard_seg,
00403 l4_uint32_t discard_sector_alignment)
00404 {
00405 l4virtio_block_config_t volatile *pc = _dev_config.priv_config();
00406 pc->max_discard_sectors = max_discard_sectors;
00407 pc->max_discard_seg = max_discard_seg;
00408 pc->discard_sector_alignment = discard_sector_alignment;
00409 Block_features df = device_features();
00410 df.discard() = true;
00411 set_device_features(df);
00412 }
00413
00422 void set_write_zeroes(l4_uint32_t max_write_zeroes_sectors,
00423 l4_uint32_t max_write_zeroes_seg,
00424 l4_uint8_t write_zeroes_may_unmap)
00425 {
00426 l4virtio_block_config_t volatile *pc = _dev_config.priv_config();
00427 pc->max_write_zeroes_sectors = max_write_zeroes_sectors;
00428 pc->max_write_zeroes_seg = max_write_zeroes_seg;
00429 pc->write_zeroes_may_unmap = write_zeroes_may_unmap;
00430 Block_features df = device_features();
00431 df.write_zeroes() = true;
00432 set_device_features(df);
00433 }
00434
00435 public:
00444 Block_dev_base(l4_uint32_t vendor, unsigned queue_size, l4_uint64_t capacity,
00445 bool read_only)
00446 : L4virtio::Svr::Device_t<Ds_data>(&_dev_config),
00447 _vq_max(queue_size),
00448 _dev_config(vendor, L4VIRTIO_ID_BLOCK, 1)
00449 {
00450 this->reset_queue_config(0, queue_size);
00451
00452 Block_features df(0);
00453 df.ring_indirect_desc() = true;
00454 df.ro() = read_only;
00455 set_device_features(df);
00456
00457 _dev_config.set_host_feature(L4VIRTIO_FEATURE_VERSION_1);
00458
00459 _dev_config.priv_config()->capacity = capacity;
00460 }
00461
00465 virtual void reset_device() = 0;
00466
00470 virtual bool queue_stopped() = 0;
00471
00483 void finalize_request(cxx::unique_ptr<Request> req, unsigned sz,
00484 l4_uint8_t status = L4VIRTIO_BLOCK_S_OK)
00485 {
00486 if (_dev_config.status().fail_state() || !_queue.ready())
00487 return;
00488
00489 if (req->release_request(&_queue, status, sz) < 0)
00490 this->device_error();
00491
00492 if (_queue.no_notify_guest())
00493 return;
00494
00495 _dev_config.add_irq_status(L4VIRTIO_IRQ_STATUS_VRING);
00496 _kick_guest_irq->trigger();
00497
00498 // Request can be dropped here.
00499 }
00500

```

```

00501 int reconfig_queue(unsigned idx) override
00502 {
00503 if (idx == 0 && this->setup_queue(&_queue, 0, _vq_max))
00504 return 0;
00505
00506 return -L4_EINVAL;
00507 }
00508
00509 void reset() override
00510 {
00511 _queue.disable();
00512 _dev_config.reset_queue(0, _vq_max);
00513 _dev_config.reset_hdr();
00514 reset_device();
00515 }
00516
00517 protected:
00518 bool check_for_new_requests()
00519 {
00520 if (!_queue.ready() || queue_stopped())
00521 return false;
00522
00523 if (_dev_config.status().fail_state())
00524 return false;
00525
00526 return _queue.desc_avail();
00527 }
00528
00530 cxx::unique_ptr<Request> get_request()
00531 {
00532 cxx::unique_ptr<Request> req;
00533
00534 if (!_queue.ready() || queue_stopped())
00535 return req;
00536
00537 if (_dev_config.status().fail_state())
00538 return req;
00539
00540 auto r = _queue.next_avail();
00541 if (!r)
00542 return req;
00543
00544 try
00545 {
00546 cxx::unique_ptr<Request> cur{
00547 new Request(r, &(this->_mem_info), _vq_max, _max_block_size)};
00548
00549 req = cxx::move(cur);
00550 }
00551 catch (Bad_descriptor const &e)
00552 {
00553 this->device_error();
00554 return req;
00555 }
00556
00557 return req;
00558 }
00559
00560 private:
00561 void register_single_driver_irq() override
00562 {
00563 _kick_guest_irq = L4Re::Util::Unique_cap<L4::Irq>(
00564 L4Re::chkcap(this->server_iface()->template rcv_cap<L4::Irq>(0)));
00565
00566 L4Re::chksys(this->server_iface()->realloc_rcv_cap(0));
00567 }
00568
00569 void trigger_driver_config_irq() override
00570 {
00571 _dev_config.add_irq_status(L4VIRTIO_IRQ_STATUS_CONFIG);
00572 _kick_guest_irq->trigger();
00573 }
00574
00575 bool check_queues() override
00576 {
00577 if (!_queue.ready())
00578 {
00579 reset();
00580 return false;
00581 }
00582
00583 return true;
00584 }
00585 };
00586
00587 template <typename Ds_data>
00588 struct Block_dev

```



```

00589 : Block_dev_base<Ds_data>,
00590 L4::Epiface_t<Block_dev<Ds_data>, L4virtio::Device>
00591 {
00592 private:
00593 class Irq_object : public L4::Irqep_t<Irq_object>
00594 {
00595 public:
00596 Irq_object(Block_dev<Ds_data> *parent) : _parent(parent) {}
00597
00598 void handle_irq()
00599 {
00600 _parent->kick();
00601 }
00602
00603 private:
00604 Block_dev<Ds_data> *_parent;
00605 };
00606 Irq_object _irq_handler;
00607
00608 protected:
00609 L4::Epiface *irq_iface()
00610 { return &_amp;_irq_handler; }
00611
00612 public:
00613 Block_dev(l4_uint32_t vendor, unsigned queue_size, l4_uint64_t capacity,
00614 bool read_only)
00615 : Block_dev_base<Ds_data>(vendor, queue_size, capacity, read_only),
00616 _irq_handler(this)
00617 {}
00618
00619 L4::Cap<void> register_obj(L4::Registry_iface *registry,
00620 char const *service = 0)
00621 {
00622 L4Re::chkcap(registry->register_irq_obj(this->irq_iface()));
00623 L4::Cap<void> ret;
00624 if (service)
00625 ret = registry->register_obj(this, service);
00626 else
00627 ret = registry->register_obj(this);
00628 L4Re::chkcap(ret);
00629
00630 return ret;
00631 }
00632
00633 L4::Cap<void> register_obj(L4::Registry_iface *registry,
00634 L4::Cap<L4::Rcv_endpoint> ep)
00635 {
00636 L4Re::chkcap(registry->register_irq_obj(this->irq_iface()));
00637
00638 return L4Re::chkcap(registry->register_obj(this, ep));
00639 }
00640
00641 typedef Block_request<Ds_data> Request;
00642 virtual bool process_request(cxx::unique_ptr<Request> &&req) = 0;
00643
00644 protected:
00645 L4::Ipc_svr::Server_iface *server_iface() const override
00646 {
00647 return this->L4::Epiface::server_iface();
00648 }
00649
00650 void kick()
00651 {
00652 for (;;)
00653 {
00654 auto req = this->get_request();
00655 if (!req)
00656 return;
00657 if (!this->process_request(cxx::move(req)))
00658 return;
00659 }
00660 }
00661
00662 private:
00663 L4::Cap<L4::Irq> device_notify_irq() const override
00664 {
00665 return L4::cap_cast<L4::Irq>(_irq_handler.obj_cap());
00666 }
00667 };
00668
00669 }
```

## 17.256 virtio-console

```

00001 // vi:ft=cpp
00002 /* SPDX-License-Identifier: MIT */
00003 /*
00004 * Copyright (C) 2019-2024 Kernkonzept GmbH.
00005 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00006 * Phillip Raffeck <phillip.raffeck@kernkonzept.com>
00007 * Steffen Liebergeld <steffen.liebergeld@kernkonzept.com>
00008 * Jan Klötzke <jan.kloetzke@kernkonzept.com>
00009 */
00010 #pragma once
00011
00012 #include <l4/l4virtio/server/l4virtio>
00013 #include <l4/re/error_helper>
00014
00015 namespace L4virtio { namespace Svr { namespace Console {
00016
00017 struct Features : Dev_config::Features
00018 {
00019 Features() = default;
00020 explicit Features(l4_uint32_t raw) : Dev_config::Features(raw) {}
00021 CXX_BITFIELD_MEMBER(0, 0, console_size, raw);
00022 CXX_BITFIELD_MEMBER(1, 1, console_multiport, raw);
00023 CXX_BITFIELD_MEMBER(2, 2, emerg_write, raw);
00024 };
00025
00026 struct Control_message
00027 {
00028 enum Events
00029 {
00030 Device_ready = 0,
00031 Device_add = 1,
00032 Device_remove = 2,
00033 Port_ready = 3,
00034 Console_port = 4,
00035 Resize = 5,
00036 Port_open = 6,
00037 Port_name = 7,
00038 };
00039 l4_uint32_t id;
00040 l4_uint16_t event;
00041 l4_uint16_t value;
00042 Control_message() {}
00043 Control_message(l4_uint32_t i, l4_uint16_t e, l4_uint16_t v)
00044 : id(i), event(e), value(v) {}
00045 };
00046
00047 struct Control_request
00048 {
00049 Control_message *msg;
00050 l4_uint32_t len;
00051 Driver_mem_region *mem;
00052 };
00053
00054 struct Port
00055 {
00056 enum Port_status
00057 {
00058 Port_disabled = 0,
00059 Port_added,
00060 Port_ready,
00061 Port_open,
00062 Port_failed,
00063 Port_num_states,
00064 };
00065 enum { Control_queue_size = 0x10 };
00066 Virtqueue tx;
00067 Virtqueue rx;
00068 Port_status status;
00069 Port_status reported_status;
00070 unsigned vq_max;
00071 Port() : status(Port_disabled), vq_max(Control_queue_size) {}
00072 Port(Port const &) = delete;
00073 Port &operator = (Port const &) = delete;
00074 virtual ~Port() = default;
00075 bool is_open() const
00076 { return status == Port_open; }

```

Generated for L4Re by Doxygen

```

00281 } __attribute__((packed));
00282
00283 public:
00293 explicit Virtio_con(unsigned max_ports, bool enable_multiport)
00294 : L4virtio::Svr::Device(&_dev_config),
00295 _num_ports(enable_multiport ? max_ports : 1),
00296 _dev_config(L4VIRTIO_VENDOR_KK, L4VIRTIO_ID_CONSOLE,
00297 enable_multiport ? max_ports * 2 + 2 : 2)
00298 {
00299 if (_num_ports < 1)
00300 L4Re::chksys(-L4_EINVAL, "At least one port is required.");
00301
00302 Features hf(0);
00303
00304 hf.console_multiport() = enable_multiport;
00305
00306 _dev_config.host_features(0) = hf.raw;
00307
00308 if (enable_multiport)
00309 _dev_config.priv_config()->max_nr_ports = _num_ports;
00310 _dev_config.reset_hdr();
00311 }
00312
00313 void reset_queue_configs()
00314 {
00315 for (unsigned q = 0; q < _dev_config.num_queues(); ++q)
00316 reset_queue_config(q, max_queue_size(q));
00317 }
00318
00319 int reconfig_queue(unsigned index) override
00320 {
00321 if (index >= _dev_config.num_queues())
00322 return -L4_ERANGE;
00323
00324 if (setup_queue(get_queue(index), index, max_queue_size(index)))
00325 return 0;
00326
00327 return -L4_EINVAL;
00328 }
00329
00334 bool multiport_enabled() const
00335 {
00336 return _negotiated_features.console_multiport()
00337 && _dev_config.num_queues() > Ctrl_rx;
00338 }
00339
00340 bool ctrl_queue_ready() const
00341 { return _ctrl_port.is_open(); }
00342
00343 bool check_features(void) override
00344 {
00345 _negotiated_features = Features(_dev_config.negotiated_features(0));
00346 return true;
00347 }
00348
00349 bool check_queues() override
00350 {
00351 // NOTE
00352 // The VIRTIO specification states:
00353 // "The port 0 receive and transmit queues always exist"
00354 // The linux driver however does not setup port 0 if the multiport feature
00355 // is negotiated.
00356 // We just go along with the linux driver and do not expect port 0 to be up,
00357 // if the multiport feature is negotiated.
00358
00359 if (multiport_enabled())
00360 // If MULTIPORT was negotiated, ctrl queues should be set up.
00361 return _ctrl_port.queues_ready();
00362
00363 // If MULTIPORT was not negotiated, port 0 should be set up.
00364 port(0)->status = Port::Port_open;
00365 return port(0)->queues_ready();
00366 }
00367
00379 int port_add(unsigned idx)
00380 {
00381 Port *p = port(idx);
00382
00383 if (p->status != Port::Port_disabled)
00384 return -L4_EPERM;
00385
00386 p->status = Port::Port_added;
00387 port_report_status(idx);
00388
00389 return L4_EOK;
00390 }
00391

```

```

00403 int port_remove(unsigned idx)
00404 {
00405 Port *p = port(idx);
00406
00407 if (p->status == Port::Port_disabled)
00408 return -L4_EPERM;
00409
00410 p->status = Port::Port_disabled;
00411 port_report_status(idx);
00412
00413 return L4_EOK;
00414 }
00415
00428 int port_open(unsigned idx, bool open)
00429 {
00430 Port *p = port(idx);
00431
00432 if ((open && p->status != Port::Port_ready)
00433 || (!open && p->status != Port::Port_open))
00434 return -L4_EPERM;
00435
00436 p->status = open ? Port::Port_open : Port::Port_ready;
00437 port_report_status(idx);
00438
00439 return L4_EOK;
00440 }
00441
00455 int port_name(unsigned idx, char const *name)
00456 {
00457 Port *p = port(idx);
00458
00459 if (p->status == Port::Port_disabled)
00460 return -L4_EPERM;
00461
00462 return send_control_message(idx, Control_message::Port_name, 0, name);
00463 }
00464
00487 int send_control_message(l4_uint32_t idx, l4_uint16_t event,
00488 l4_uint16_t value = 0, const char *name = 0)
00489 {
00490 if (!ctrl_queue_ready())
00491 return -L4_ENODEV;
00492
00493 Virtqueue *q = &_ctrl_port.rx;
00494 if (!q->ready())
00495 return -L4_ENODEV;
00496
00497 Virtqueue::Request r = q->next_avail();
00498 if (!r)
00499 return -L4_EBUSY;
00500
00501 Request_processor rp;
00502 Control_request req;
00503 rp.start(this, r, &req);
00504
00505 if (req.len < sizeof(Control_message))
00506 return -L4_ENOMEM;
00507
00508 Control_message msg(idx, event, value);
00509
00510 memcpy(req.msg, &msg, sizeof(msg));
00511
00512 if (event == Control_message::Port_name && name)
00513 {
00514 size_t name_len = cxx::min(req.len - sizeof(msg), strlen(name));
00515 memcpy(reinterpret_cast<char*>(req.msg) + sizeof(msg), name, name_len);
00516 q->finish(r, this, sizeof(msg) + name_len);
00517 }
00518 else
00519 q->finish(r, this, sizeof(msg));
00520
00521 return L4_EOK;
00522 }
00523
00536 int handle_control_message()
00537 {
00538 // Report port state transitions if that failed in the past...
00539 if (_report_port_state)
00540 {
00541 _report_port_state = false;
00542
00543 for (unsigned i = 0; i < _num_ports; ++i)
00544 if (!port_report_status(i))
00545 _report_port_state = true;
00546 }
00547
00548 Virtqueue *q = &_ctrl_port.tx;

```

```

00549 if (!q->ready())
00550 return -L4_ENODEV;
00551
00552 int ret = L4_EOK;
00553 Virtqueue::Request r;
00554 while ((r = q->next_avail()))
00555 {
00556 Request_processor rp;
00557 Control_request req;
00558
00559 rp.start(this, r, &req);
00560
00561 Control_message msg;
00562 if (req.len < sizeof(msg))
00563 {
00564 // Just ignore malformed input.
00565 q->finish(r, this);
00566 ret = -L4_EINVAL;
00567 continue;
00568 }
00569
00570 memcpy(&msg, req.msg, sizeof(msg));
00571 q->finish(r, this);
00572
00573 if (_ctrl_port.status == Port::Port_disabled)
00574 {
00575 // When the control queue is disabled, only device ready is accepted.
00576 if (msg.event == Control_message::Device_ready)
00577 {
00578 if (msg.value)
00579 _ctrl_port.status = Port::Port_open;
00580 }
00581
00582 process_device_ready(msg.value);
00583 continue;
00584 }
00585
00586 if (!ctrl_queue_ready())
00587 continue;
00588
00589 // Ignore invalid port ids
00590 if (msg.id >= max_ports())
00591 break;
00592
00593 switch (msg.event)
00594 {
00595 case Control_message::Port_ready:
00596 process_port_ready(msg.id, msg.value);
00597 break;
00598 case Control_message::Port_open:
00599 process_port_open(msg.id, msg.value);
00600 break;
00601 default:
00602 ret = -L4_EINVAL;
00603 break;
00604 }
00605 }
00606
00607 return ret;
00608 }
00609
00611 void load_desc(L4virtio::Virtqueue::Desc const &desc,
00612 Request_processor const *proc,
00613 L4virtio::Virtqueue::Desc const **table)
00614 {
00615 this->_mem_info.load_desc(desc, proc, table);
00616 }
00617
00619 void load_desc(L4virtio::Virtqueue::Desc const &desc,
00620 Request_processor const *proc,
00621 Control_request *data)
00622 {
00623 auto *region = this->_mem_info.find(desc.addr.get(), desc.len);
00624 if (L4_UNLIKELY(!region))
00625 throw Bad_descriptor(proc, Bad_descriptor::Bad_address);
00626
00627 data->msg = reinterpret_cast<Control_message *>(region->local(desc.addr));
00628 data->len = desc.len;
00629 data->mem = region;
00630 }
00631
00632 void reset() override
00633 {
00634 for (unsigned p = 0; p < _num_ports; ++p)
00635 port(p)->reset();
00636
00637 _ctrl_port.reset();

```

```

00638 reset_queue_configs();
00639 _dev_config.reset_hdr();
00640 _negotiated_features = Features(0);
00641 _report_port_state = false;
00642
00643 reset_device();
00644 }
00645
00652 virtual void reset_device() {}
00653
00663 virtual void notify_queue(Virtqueue *queue) = 0;
00664
00672 virtual Port *port(unsigned port) = 0;
00673 virtual Port const *port(unsigned port) const = 0;
00674
00685 virtual void process_device_ready(l4_uint16_t value) = 0;
00686
00698 virtual void process_port_ready(l4_uint32_t id, l4_uint16_t value)
00699 {
00700 Port *p = port(id);
00701
00702 switch (p->status)
00703 {
00704 case Port::Port_added:
00705 case Port::Port_ready:
00706 p->status = value ? Port::Port_ready : Port::Port_failed;
00707 break;
00708 case Port::Port_open:
00709 if (!value)
00710 p->status = Port::Port_failed;
00711 break;
00712 default:
00713 // invalid state for PORT_READY message
00714 break;
00715 }
00716 }
00717
00728 virtual void process_port_open(l4_uint32_t id, l4_uint16_t value) = 0;
00729
00730 unsigned max_ports() const
00731 { return _num_ports; }
00732
00733 private:
00734 bool is_control_queue(unsigned q) const
00735 { return q == Ctrl_rx || q == Ctrl_tx; }
00736
00737 unsigned queue_to_port(unsigned q) const
00738 { return (q == 0 || q == 1) ? 0 : (q / 2) - 1; }
00739
00748 unsigned max_queue_size(unsigned q) const
00749 {
00750 if (is_control_queue(q))
00751 return _ctrl_port.vq_max;
00752
00753 return port(queue_to_port(q))->vq_max;
00754 }
00755
00764 Virtqueue *get_queue(unsigned q)
00765 {
00766 Port *p;
00767 if (is_control_queue(q))
00768 p = &_ctrl_port;
00769 else
00770 p = port(queue_to_port(q));
00771
00772 if (q & 1)
00773 return &p->tx;
00774 else
00775 return &p->rx;
00776 }
00777
00788 bool port_report_status(unsigned idx)
00789 {
00790 Port *p = port(idx);
00791 while (p->status != p->reported_status)
00792 {
00793 auto const &trans
00794 = Port::state_transitions[p->reported_status][p->status];
00795
00796 if (trans.event >= 0
00797 && send_control_message(idx, trans.event, trans.value) < 0)
00798 {
00799 _report_port_state = true;
00800 return false;
00801 }
00802
00803 p->reported_status = trans.next;

```

```

00804 }
00805
00806 return true;
00807 }
00808
00809 unsigned _num_ports;
00810 bool _report_port_state = false;
00811
00812 protected:
00813 Dev_config_t<Serial_config_space> _dev_config;
00814 Port _ctrl_port;
00815 Features _negotiated_features{0};
00816 };
00817
00818 }}} // name space

```

## 17.257 virtio-console-device

```

00001 // vi:ft=cpp
00002 /* SPDX-License-Identifier: MIT */
00003 /*
00004 * Copyright (C) 2019-2024 Kernkonzept GmbH.
00005 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00006 * Phillip Raffeck <phillip.raffeck@kernkonzept.com>
00007 * Steffen Liebergeld <steffen.liebergeld@kernkonzept.com>
00008 * Jan Klötzke <jan.kloetzke@kernkonzept.com>
00009 */
00010 #pragma once
00011
00012 #include <l4/cxx/bitmap>
00013 #include <l4/cxx/static_vector>
00014 #include <l4/l4virtio/server/l4virtio>
00015 #include <l4/l4virtio/server/virtio-console>
00016 #include <l4/re/error_helper>
00017
00018 namespace L4virtio { namespace Svr { namespace Console {
00019
00020 struct Device_port : public Port
00021 {
00022 public:
00023 struct Buffer : Data_buffer
00024 {
00025 public:
00026 Buffer() = default;
00027 Buffer(Driver_mem_region const *r,
00028 Virtqueue::Desc const &d,
00029 Request_processor const *)
00030 {
00031 pos = static_cast<char *>(r->local(d.addr));
00032 left = d.len;
00033 }
00034 };
00035
00036 Request_processor rp;
00037 Virtqueue::Request request;
00038 Buffer src;
00039
00040 bool poll_in_req = true;
00041 bool poll_out_req = true;
00042
00043 void reset() override
00044 {
00045 Port::reset();
00046 request = Virtqueue::Request();
00047 poll_in_req = true;
00048 poll_out_req = true;
00049 }
00050 };
00051
00052 class Device
00053 : public Virtio_console
00054 {
00055 public:
00056 class Irq_object : public L4::Irqep_t<Irq_object>
00057 {
00058 public:
00059 Irq_object(Device *parent) : _parent(parent) {}
00060
00061 void handle_irq() { _parent->kick(); }
00062
00063 private:
00064 Device *_parent;
00065 };
00066
00067 protected:
00068 L4::Epiface *irq_iface()

```



```

00134 { return &_irq_handler; }
00135
00136 public:
00145 explicit Device(unsigned vq_max)
00146 : Virtio_con(1, false),
00147 _irq_handler(this),
00148 _ports(cxx::make_unique<Device_port[]>(1))
00149 {
00150 _ports[0].vq_max = vq_max;
00151 reset_queue_configs();
00152 }
00153
00163 explicit Device(unsigned vq_max, unsigned ports)
00164 : Virtio_con(ports, true),
00165 _irq_handler(this),
00166 _ports(cxx::make_unique<Device_port[]>(ports))
00167 {
00168 for (unsigned i = 0; i < ports; ++i)
00169 _ports[i].vq_max = vq_max;
00170 reset_queue_configs();
00171 }
00172
00182 explicit Device(cxx::static_vector<unsigned> const &vq_max_nums)
00183 : Virtio_con(vq_max_nums.size(), true),
00184 _irq_handler(this),
00185 _ports(cxx::make_unique<Device_port[]>(max_ports()))
00186 {
00187 for (unsigned i = 0; i < vq_max_nums.size(); ++i)
00188 _ports[i].vq_max = vq_max_nums[i];
00189 reset_queue_configs();
00190 }
00191
00192 void register_single_driver_irq() override
00193 {
00194 _kick_driver_irq = L4Re::Util::Unique_cap<L4::Irq>(
00195 L4Re::chkcap(server_iface()->rcv_cap<L4::Irq>(0)));
00196 L4Re::chksys(server_iface()->realloc_rcv_cap(0));
00197 }
00198
00199 L4::Cap<L4::Irq> device_notify_irq() const override
00200 { return _irq_handler.obj_cap(); }
00201
00202 void notify_queue(Virtqueue *queue) override
00203 {
00204 if (queue->no_notify_guest())
00205 return;
00206
00207 _dev_config.add_irq_status(L4VIRTIO_IRQ_STATUS_VRING);
00208 _kick_driver_irq->trigger();
00209 }
00210
00214 virtual void rx_data_available(unsigned port) = 0;
00215
00219 virtual void tx_space_available(unsigned port) = 0;
00220
00224 virtual bool queues_stopped()
00225 { return false; }
00226
00227 void trigger_driver_config_irq() override
00228 {
00229 _dev_config.add_irq_status(L4VIRTIO_IRQ_STATUS_CONFIG);
00230 _kick_driver_irq->trigger();
00231 }
00232
00233 void kick()
00234 {
00235 if (queues_stopped())
00236 return;
00237
00238 // We're not interested in logging any errors, just ignore return value.
00239 handle_control_message();
00240
00241 for (unsigned i = 0; i < max_ports(); ++i)
00242 {
00243 auto &p = _ports[i];
00244 if (p.poll_in_req && p.tx_ready() && p.tx.desc_avail())
00245 {
00246 p.poll_in_req = false;
00247 rx_data_available(i);
00248 }
00249
00250 if (p.poll_out_req && p.rx_ready() && p.rx.desc_avail())
00251 {
00252 p.poll_out_req = false;
00253 tx_space_available(i);
00254 }
00255 }

```

```

00256 }
00257
00272 unsigned port_read(char *buf, unsigned len, unsigned port = 0)
00273 {
00274 unsigned total = 0;
00275 Device_port &p = _ports[port];
00276 Virtqueue *q = &p.tx;
00277
00278 Data_buffer dst;
00279 dst.pos = buf;
00280 dst.left = len;
00281
00282 while (dst.left)
00283 {
00284 try
00285 {
00286 // Make sure we have a valid request where we can read data from
00287 if (!p.request.valid())
00288 {
00289 p.request = p.tx_ready() ? q->next_avail()
00290 : Virtqueue::Request();
00291 if (!p.request.valid())
00292 break;
00293
00294 p.rp.start(mem_info(), p.request, &p.src);
00295 }
00296
00297 total += p.src.copy_to(&dst);
00298
00299 // We might have eaten up the current descriptor. Move to the next
00300 // if this is the case. At the end of the descriptor chain we have
00301 // to retire the current request altogether.
00302 if (!p.src.left)
00303 {
00304 if (!p.rp.next(mem_info(), &p.src))
00305 {
00306 q->finish(p.request, this);
00307 p.request = Virtqueue::Request();
00308 }
00309 }
00310 }
00311 catch (Bad_descriptor const &)
00312 {
00313 q->finish(p.request, this);
00314 p.request = Virtqueue::Request();
00315 device_error();
00316 break;
00317 }
00318 }
00319
00320 if (total < len)
00321 p.poll_in_req = true;
00322
00323 return total;
00324 }
00325
00341 unsigned port_write(char const *buf, unsigned len, unsigned port = 0)
00342 {
00343 unsigned total = 0;
00344 Device_port &p = _ports[port];
00345 Virtqueue *q = &p.rx;
00346
00347 Data_buffer src;
00348 src.pos = const_cast<char*>(buf);
00349 src.left = len;
00350
00351 Request_processor rp;
00352 while (src.left)
00353 {
00354 auto r = p.rx_ready() ? q->next_avail() : Virtqueue::Request();
00355 if (!r.valid())
00356 break;
00357
00358 l4_uint32_t chunk = 0;
00359 try
00360 {
00361 Device_port::Buffer dst;
00362 rp.start(mem_info(), r, &dst);
00363
00364 for (;;)
00365 {
00366 chunk += src.copy_to(&dst);
00367 if (!src.left)
00368 break;
00369 if (!rp.next(mem_info(), &dst))
00370 break;
00371 }

```

```

00372 }
00373 catch (Bad_descriptor const &)
00374 {
00375 device_error();
00376 }
00377
00378 q->finish(r, this, chunk);
00379 total += chunk;
00380 }
00381
00382 if (total < len)
00383 p.poll_out_req = true;
00384
00385 return total;
00386 }
00387
00399 void process_device_ready(l4_uint16_t value) override
00400 {
00401 if (!value)
00402 return;
00403
00404 for (unsigned i = 0; i < max_ports(); ++i)
00405 port_add(i);
00406 }
00407
00422 void process_port_ready(l4_uint32_t id, l4_uint16_t value) override
00423 {
00424 Virtio_con::process_port_ready(id, value);
00425
00426 Port *p = port(id);
00427 if (p->status == Port::Port_failed)
00428 port_remove(id);
00429 else if (p->status == Port::Port_ready)
00430 port_open(id, true);
00431 }
00432
00443 virtual void process_port_open(l4_uint32_t id, l4_uint16_t value)
00444 {
00445 static_cast<void>(id);
00446 static_cast<void>(value);
00447 }
00448
00449 protected:
00450 Port* port(unsigned idx) override
00451 {
00452 return &_amp;ports[idx];
00453 }
00454
00455 Port const *port(unsigned idx) const override
00456 {
00457 return &_amp;ports[idx];
00458 }
00459
00460 private:
00461 Irq_object _irq_handler;
00462 cxx::unique_ptr<Device_port[]> _ports;
00463 L4Re::Util::Unique_cap<L4::Irq> _kick_driver_irq;
00464 };
00465
00466 }}} // name space

```

## 17.258 virtio-gpio-device

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * Copyright (C) 2025 Kernkonzept GmbH.
00004 * Author(s): Christian Pötzsch <christian.poetzsch@kernkonzept.com>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00009 #pragma once
00010
00011 #include <l4/re/error_helper>
00012 #include <l4/sys/cxx/ipc_epiface>
00013
00014 #include <l4/l4virtio/server/virtio>
00015 #include <l4/l4virtio/server/l4virtio>
00016 #include <l4/l4virtio/l4virtio>
00017
00018 #include <l4/re/error_helper>
00019 #include <l4/re/util/object_registry>
00020 #include <l4/re/util/br_manager>

```

```

00021 #include <l4/sys/cxx/ipc_epiface>
00022 #include <l4/cxx/pair>
00023
00024 #include <vector>
00025 #include <memory>
00026
00027 namespace L4virtio {
00028 namespace Svr {
00029
00030 /* GPIO message status types */
00031 enum : l4_uint8_t
00032 {
00033 Gpio_status_ok = 0x0,
00034 Gpio_status_err = 0x1
00035 };
00036
00037 /* GPIO message types */
00038 enum : l4_uint8_t
00039 {
00040 Gpio_msg_get_line_names = 0x1,
00041 Gpio_msg_get_direction = 0x2,
00042 Gpio_msg_set_direction = 0x3,
00043 Gpio_msg_get_value = 0x4,
00044 Gpio_msg_set_value = 0x5,
00045 Gpio_msg_set_irq_type = 0x6
00046 };
00047
00048 /* GPIO value types */
00049 enum : l4_uint8_t
00050 {
00051 Gpio_low = 0x0,
00052 Gpio_high = 0x1
00053 };
00054
00055 /* GPIO direction types */
00056 enum : l4_uint8_t
00057 {
00058 Gpio_direction_none = 0x0,
00059 Gpio_direction_out = 0x1,
00060 Gpio_direction_in = 0x2
00061 };
00062
00063 /* GPIO interrupt types */
00064 enum : l4_uint8_t
00065 {
00066 Gpio_irq_type_none = 0x0,
00067 Gpio_irq_type_edge_rising = 0x1,
00068 Gpio_irq_type_edge_falling = 0x2,
00069 Gpio_irq_type_edge_both = 0x3,
00070 Gpio_irq_type_level_high = 0x4,
00071 Gpio_irq_type_level_low = 0x8
00072 };
00073
00074 /* GPIO interrupt status types */
00075 enum : l4_uint8_t
00076 {
00077 Gpio_irq_status_invalid = 0x0,
00078 Gpio_irq_status_valid = 0x1
00079 };
00080
00081 struct Gpio_request
00082 {
00083 l4_uint16_t type;
00084 l4_uint16_t gpio;
00085 l4_uint32_t value;
00086 };
00087 static_assert(sizeof(Gpio_request) == 8,
00088 "Gpio_request contains padding bytes.");
00089
00090 struct Gpio_response
00091 {
00092 l4_uint8_t status;
00093 l4_uint8_t value;
00094 };
00095 static_assert(sizeof(Gpio_response) == 2,
00096 "Gpio_response contains padding bytes.");
00097
00098 struct Gpio_irq_request
00099 {
00100 l4_uint16_t gpio;
00101 };
00102 static_assert(sizeof(Gpio_irq_request) == 2,
00103 "Gpio_irq_request contains padding bytes.");
00104
00105 struct Gpio_irq_response
00106 {
00107 l4_uint8_t status;

```

```

00108 };
00109 static_assert(sizeof(Gpio_irq_response) == 1,
00110 "Gpio_irq_response contains padding bytes.");
00111
00112 struct Gpio_request_msg
00113 {
00114 struct Gpio_request *in_hdr = nullptr;
00115 struct Gpio_response *out_hdr = nullptr;
00116 };
00117
00118 struct Gpio_irq_request_msg
00119 {
00120 struct Gpio_irq_request *in_hdr = nullptr;
00121 struct Gpio_irq_response *out_hdr = nullptr;
00122 };
00123
00140 template <typename Request_handler,
00141 typename Epiface = L4virtio::Device>
00142 class Virtio_gpio : public L4virtio::Svr::Device,
00143 public L4::Epiface_t<Virtio_gpio<Request_handler,
00144 Epiface>,
00145 Epiface>
00146 {
00147 private:
00148 enum
00149 {
00150 queue_size = 128,
00151 };
00152
00153 public:
00154 using Gpio_request_handler = Request_handler;
00155
00166 struct Irq_handler
00167 {
00168 Irq_handler(Virtio_gpio *gpio, L4virtio::Svr::Virtqueue *q,
00169 L4virtio::Svr::Virtqueue::Head_desc const &head,
00170 l4_uint8_t *status)
00171 : _gpio(gpio), _q(q), _head(head), _status(status)
00172 {}
00173
00174 void handle_irq()
00175 {
00176 *_status = Gpio_irq_status_valid;
00177 _q->finish(_head, _gpio, sizeof(Gpio_irq_response));
00178 }
00179
00180 void cancel()
00181 {
00182 *_status = Gpio_irq_status_invalid;
00183 _q->finish(_head, _gpio, sizeof(Gpio_irq_response));
00184 }
00185
00186 private:
00187 Virtio_gpio *_gpio;
00188 L4virtio::Svr::Virtqueue *_q;
00189 L4virtio::Svr::Virtqueue::Head_desc _head;
00190 l4_uint8_t *_status;
00191 };
00192
00198 struct Host_irq : L4::Irqep_t<Host_irq>
00199 {
00200 explicit Host_irq(Virtio_gpio *gpio)
00201 : L4::Irqep_t<Host_irq>(), _gpio(gpio) {}
00202
00203 void handle_irq()
00204 { _gpio->handle_queue(); }
00205
00206 private:
00207 Virtio_gpio *_gpio;
00208 };
00209
00213 struct Request_processor : L4virtio::Svr::Request_processor
00214 {
00215 Request_processor(L4virtio::Svr::Virtqueue *q, Gpio_request_handler *hndlr,
00216 Virtio_gpio *gpio)
00217 : _q(q), _req_handler(hndlr), _gpio(gpio), _head(), _req()
00218 {}
00219
00220 protected:
00221 bool init_queue()
00222 {
00223 auto r = _q->next_avail();
00224
00225 if (L4_UNLIKELY(!r))
00226 return false;
00227
00228 _head = start(_gpio->mem_info(), r, &_req);

```

```

00229 return true;
00230 }
00231 }
00232
00240 template <typename T>
00241 T get_request()
00242 {
00243 T req;
00244 req.in_hdr = reinterpret_cast<decltype(T::in_hdr)>(_req.pos);
00245
00246 // Need the next output buffer.
00247 if (!next(_gpio->mem_info(), &_req) || !_current_flags().write())
00248 return req;
00249
00250 req.out_hdr = reinterpret_cast<decltype(T::out_hdr)>(_req.pos);
00251
00252 return req;
00253 }
00254
00255 struct Data_buffer : public L4virtio::Svr::Data_buffer
00256 {
00257 Data_buffer()
00258 {
00259 pos = nullptr;
00260 left = 0;
00261 }
00262 // This constructor is called from within start, so make it available.
00263 Data_buffer(L4virtio::Svr::Driver_mem_region const *r,
00264 L4virtio::Svr::Virtqueue::Desc const &d,
00265 L4virtio::Svr::Request_processor const *)
00266 {
00267 pos = static_cast<char *>(r->local(d.addr));
00268 left = d.len;
00269 }
00270 };
00271
00272 L4virtio::Svr::Virtqueue *_q;
00273 Gpio_request_handler *_req_handler;
00274 Virtio_gpio *_gpio;
00275 L4virtio::Svr::Virtqueue::Head_desc _head;
00276 Data_buffer _req;
00277 };
00278
00279 // Handler for the gpio request queue
00280 struct Req_processor : Request_processor
00281 {
00282 using Request_processor::Request_processor;
00283
00284 void handle_request()
00285 {
00286 if (!this->_head)
00287 if (!this->init_queue())
00288 return;
00289
00290 using Consumed_entry =
00291 cxx::Pair<L4virtio::Svr::Virtqueue::Head_desc, l4_uint32_t>;
00292 std::vector<Consumed_entry> consumed;
00293
00294 for (;;)
00295 {
00296 Gpio_request_msg req = this->template get_request<Gpio_request_msg>();
00297 if (!req.in_hdr || !req.out_hdr)
00298 {
00299 this->_gpio->device_error();
00300 break;
00301 }
00302
00303 // default response is error
00304 req.out_hdr->status = Gpio_status_err;
00305 switch (req.in_hdr->type)
00306 {
00307 case Gpio_msg_get_line_names:
00308 // we don't support this
00309 break;
00310 case Gpio_msg_get_direction:
00311 {
00312 if (this->_req_handler->get_direction(req.in_hdr->gpio,
00313 &req.out_hdr->value))
00314 req.out_hdr->status = Gpio_status_ok;
00315 break;
00316 }
00317 case Gpio_msg_set_direction:
00318 {
00319 if (req.in_hdr->value == Gpio_direction_none ||
00320 req.in_hdr->value == Gpio_direction_out ||
00321 req.in_hdr->value == Gpio_direction_in)

```

```

00323 {
00324 if (this->_req_handler->set_direction(req.in_hdr->gpio,
00325 req.in_hdr->value))
00326 req.out_hdr->status = Gpio_status_ok;
00327 }
00328 break;
00329 }
00330 case Gpio_msg_get_value:
00331 {
00332 if (this->_req_handler->get_value(req.in_hdr->gpio,
00333 &req.out_hdr->value))
00334 req.out_hdr->status = Gpio_status_ok;
00335 break;
00336 }
00337 case Gpio_msg_set_value:
00338 {
00339 if (req.in_hdr->value == Gpio_low ||
00340 req.in_hdr->value == Gpio_high)
00341 {
00342 if (this->_req_handler->set_value(req.in_hdr->gpio,
00343 req.in_hdr->value))
00344 req.out_hdr->status = Gpio_status_ok;
00345 break;
00346 }
00347 }
00348 case Gpio_msg_set_irq_type:
00349 {
00350 if (req.in_hdr->value == Gpio_irq_type_none ||
00351 req.in_hdr->value == Gpio_irq_type_edge_rising ||
00352 req.in_hdr->value == Gpio_irq_type_edge_falling ||
00353 req.in_hdr->value == Gpio_irq_type_edge_both ||
00354 req.in_hdr->value == Gpio_irq_type_level_high ||
00355 req.in_hdr->value == Gpio_irq_type_level_low)
00356 {
00357 if (this->_req_handler->set_irq_type(req.in_hdr->gpio,
00358 req.in_hdr->value))
00359 req.out_hdr->status = Gpio_status_ok;
00360 break;
00361 }
00362 }
00363 }
00364
00365 // Save the descriptors which are done
00366 consumed.emplace_back(this->_head, sizeof(Gpio_response));
00367
00368 if (!this->init_queue())
00369 break;
00370 }
00371
00372 // Put all finished descriptors back into the used list and notify the
00373 // driver.
00374 this->_q->finish(consumed.begin(), consumed.end(), this->_gpio);
00375
00376 this->_head = Virtqueue::Head_desc();
00377 }
00378 };
00379
00380 // Handler for the gpio event queue
00381 struct Irq_req_processor : Request_processor
00382 {
00383 using Request_processor::Request_processor;
00384
00385 void handle_request()
00386 {
00387 if (!this->_head)
00388 if (!this->init_queue())
00389 return;
00390
00391 for (;;)
00392 {
00393 // There is only one type of message in the event queue. This
00394 // basically arms (unmask) the irq.
00395 Gpio_irq_request_msg req = this->template get_request<Gpio_irq_request_msg>();
00396 if (!req.in_hdr || !req.out_hdr)
00397 {
00398 this->_gpio->device_error();
00399 break;
00400 }
00401
00402 // Save the virtio descriptor for this event in an extra Irq_handler
00403 // object. The descriptor will be returned to the client when the irq
00404 // is triggered or canceled.
00405 this->_req_handler->enable_irq(req.in_hdr->gpio,
00406 std::make_shared<Irq_handler>(this->_gpio,
00407 this->_q,
00408 this->_head,
00409 &req.out_hdr->status));

```

```

00410
00411 if (!this->init_queue())
00412 break;
00413 }
00414
00415 this->_head = Virtqueue::Head_desc();
00416 }
00417 };
00418
00419 struct Features : public L4virtio::Svr::Dev_config::Features
00420 {
00421 Features() = default;
00422 Features(l4_uint32_t raw) : L4virtio::Svr::Dev_config::Features(raw) {}
00423
00424 CXX_BITFIELD_MEMBER(0, 0, gpio_f_irq, raw);
00425 };
00426
00427 struct Gpio_config_space
00428 {
00429 l4_uint16_t ngpio;
00430 l4_uint8_t padding[2];
00431 l4_uint32_t gpio_names_size;
00432 };
00433
00434 Virtio_gpio(Gpio_request_handler *hndlr,
00435 L4Re::Util::Object_registry *registry,
00436 l4_uint16_t ngpio)
00437 : L4virtio::Svr::Device(&_dev_config),
00438 _registry(registry),
00439 _dev_config(L4VIRTIO_VENDOR_KK, L4VIRTIO_ID_GPIO, 2),
00440 _host_irq(this),
00441 _req_processor(&_q[0], hndlr, this),
00442 _irq_req_processor(&_q[1], hndlr, this)
00443 {
00444 init_mem_info(2);
00445
00446 for (size_t i = 0; i < 2; i++)
00447 {
00448 reset_queue_config(i, queue_size);
00449 setup_queue(&_q[i], i, queue_size);
00450 }
00451
00452 registry->register_irq_obj(&_host_irq);
00453
00454 Features hf(0);
00455 hf.ring_indirect_desc() = true;
00456 hf.gpio_f_irq() = true;
00457 _dev_config.host_features(0) = hf.raw;
00458 _dev_config.set_host_feature(L4VIRTIO_FEATURE_VERSION_1);
00459
00460 // fill gpio config space
00461 _dev_config.priv_config()->ngpio = ngpio;
00462 _dev_config.priv_config()->gpio_names_size = 0; // not supported
00463
00464 _dev_config.reset_hdr();
00465 }
00466
00467 ~Virtio_gpio()
00468 { _registry->unregister_obj(&_host_irq); }
00469
00470 void notify_queue(L4virtio::Svr::Virtqueue *queue)
00471 {
00472 if (queue->no_notify_guest())
00473 return;
00474
00475 _dev_config.add_irq_status(L4VIRTIO_IRQ_STATUS_VRING);
00476 L4Re::chkipc(_notify_guest_irq->trigger(), "trigger guest irq");
00477 }
00478
00479 void handle_queue()
00480 {
00481 _req_processor.handle_request();
00482 _irq_req_processor.handle_request();
00483 }
00484
00485 void reset() override
00486 {}
00487
00488 bool check_queues() override
00489 { return true; }
00490
00491 int reconfig_queue(unsigned idx) override
00492 {
00493 if (idx >= sizeof(_q) / sizeof(_q[0]))
00494 return -L4_ERANGE;
00495
00496 return setup_queue(_q + idx, idx, queue_size);

```



```

00497 }
00498
00499 void trigger_driver_config_irq() override
00500 {
00501 _dev_config.add_irq_status(L4VIRTIO_IRQ_STATUS_CONFIG);
00502 _notify_guest_irq->trigger();
00503 }
00504
00505 L4::Ipc_svr::Server_iface *server_iface() const override
00506 { return L4::Epiface::server_iface(); }
00507
00508 long op_set_status(L4virtio::Device::Rights r, unsigned status)
00509 { return L4virtio::Svr::Device::op_set_status(r, status); }
00510
00511 long op_config_queue(L4virtio::Device::Rights r, unsigned queue)
00512 { return L4virtio::Svr::Device::op_config_queue(r, queue); }
00513
00514 long op_device_config(L4virtio::Device::Rights r,
00515 L4::Ipc::Cap<L4Re::Dataspace> &config_ds,
00516 l4_addr_t &ds_offset)
00517 { return L4virtio::Svr::Device::op_device_config(r, config_ds, ds_offset); }
00518
00519 L4::Cap<L4::Irc> device_notify_irq() const override
00520 { return L4::cap_cast<L4::Irc>(_host_irq.obj_cap()); }
00521
00522 void register_single_driver_irq() override
00523 {
00524 _notify_guest_irq = L4Re::chkcap
00525 (server_iface()->template rcv_cap<L4::Irc>(0));
00526
00527 L4Re::chksys(server_iface()->realloc_rcv_cap(0));
00528 }
00529
00530 private:
00531 L4Re::Util::Object_registry *_registry;
00532 L4virtio::Svr::Dev_config_t<Gpio_config_space> _dev_config;
00533 Host_irq _host_irq;
00534 L4::Cap<L4::Irc> _notify_guest_irq;
00535 L4virtio::Svr::Virtqueue _q[2];
00536 Req_processor _req_processor;
00537 Irc_req_processor _irc_req_processor;
00538 };
00539
00540 } // namespace Svr
00541 } // namespace L4virtio

```

## 17.259 virtio-i2c-device

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * Copyright (C) 2024 Kernkonzept GmbH.
00004 * Author(s): Martin Kuetzler <martin.kuetzler@kernkonzept.com>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00009 #pragma once
00010
00011 #include <l4/re/error_helper>
00012 #include <l4/sys/cxx/ipc_epiface>
00013
00014 #include <l4/l4virtio/server/virtio>
00015 #include <l4/l4virtio/server/l4virtio>
00016 #include <l4/l4virtio/l4virtio>
00017
00018 #include <l4/re/error_helper>
00019 #include <l4/re/util/object_registry>
00020 #include <l4/re/util/br_manager>
00021 #include <l4/sys/cxx/ipc_epiface>
00022
00023 #include <vector>
00024 #include <l4/cxx/pair>
00025
00026 namespace L4virtio {
00027 namespace Svr {
00028
00029 enum : l4_uint8_t
00030 {
00031 I2c_msg_ok = 0,
00032 I2c_msg_err = 1,
00033 };
00034
00035 struct I2c_request_flags

```

```

00036 {
00037 l4_uint32_t raw;
00038
00039 CXX_BITFIELD_MEMBER(0, 0, fail_next, raw);
00040 CXX_BITFIELD_MEMBER(1, 1, m_rd, raw);
00041 };
00042 static_assert(sizeof(I2c_request_flags) == 4,
00043 "I2c_request_flags contains padding bytes.");
00044
00045 struct I2c_out_hdr
00046 {
00047 l4_uint16_t addr;
00048 l4_uint16_t padding;
00049 I2c_request_flags flags;
00050 };
00051 static_assert(sizeof(I2c_out_hdr) == 8, "I2c_out_hdr contains padding bytes.");
00052
00053 struct I2c_in_hdr
00054 {
00055 l4_uint8_t status;
00056 };
00057 static_assert(sizeof(I2c_in_hdr) == 1, "I2c_in_hdr contains padding bytes.");
00058
00059 struct I2c_req
00060 {
00061 struct I2c_out_hdr out_hdr;
00062 unsigned buf_len;
00063 l4_uint8_t* buf;
00064 struct I2c_in_hdr *in_hdr;
00065
00066 unsigned write_size;
00067
00068 void set_status(l4_uint8_t status)
00069 {
00070 in_hdr->status = status;
00071 }
00072 };
00073
00074 template <typename Request_handler,
00075 typename Epiface = L4virtio::Device>
00076 class Virtio_i2c : public L4virtio::Svr::Device,
00077 public L4::Epiface_t<Virtio_i2c<Request_handler,
00078 Epiface>,
00079 Epiface>
00080 {
00081 private:
00082 enum
00083 {
00084 Num_request_queues = 1,
00085 queue_size = 128,
00086 };
00087 public:
00088 using I2c_request_handler = Request_handler;
00089
00090 class Host_irq : public L4::Irqep_t<Host_irq>
00091 {
00092 public:
00093 explicit Host_irq(Virtio_i2c *i2c) : L4::Irqep_t<Host_irq>(), _i2c(i2c) {}
00094
00095 void handle_irq()
00096 {
00097 _i2c->handle_queue();
00098 }
00099
00100 private:
00101 Virtio_i2c *_i2c;
00102 };
00103
00104 class Request_processor : public L4virtio::Svr::Request_processor
00105 {
00106 public:
00107 struct Data_buffer : public L4virtio::Svr::Data_buffer
00108 {
00109 Data_buffer()
00110 {
00111 pos = nullptr;
00112 left = 0;
00113 }
00114
00115 // This constructor is called from within start, so make it available.
00116 Data_buffer(L4virtio::Svr::Driver_mem_region const *r,
00117 L4virtio::Svr::Virtqueue::Desc const &d,
00118 L4virtio::Svr::Request_processor const *)
00119 {
00120 pos = static_cast<char *>(r->local(d.addr));
00121 left = d.len;
00122 }
00123 };

```

```

00141 }
00142
00143 };
00144
00145 Request_processor(L4virtio::Svr::Virtqueue *q, I2c_request_handler *hndlr,
00146 Virtio_i2c *i2c)
00147 : _q(q), _req_handler(hndlr), _i2c(i2c), _head(), _req(),
00148 _fail_next(false)
00149 {}
00150
00151 bool init_queue()
00152 {
00153 auto r = _q->next_avail();
00154
00155 if (L4_UNLIKELY(!r))
00156 return false;
00157
00158 _head = start(_i2c->mem_info(), r, &_req);
00159
00160 return true;
00161 }
00162
00172 I2c_req get_request()
00173 {
00174 I2c_req request;
00175 memcpy(&request.out_hdr, _req.pos, sizeof(I2c_out_hdr));
00176
00177 // number of bytes to be written in the answer.
00178 request.write_size = sizeof(I2c_in_hdr);
00179 request.buf_len = 0;
00180
00181 Data_buffer req;
00182 // 2nd part: either the optional buffer or the in_hdr
00183 if (next(_i2c->mem_info(), &req))
00184 {
00185
00186 request.buf_len += req.left;
00187 request.buf = reinterpret_cast<l4_uint8_t *>(&req.pos);
00188 }
00189
00190 // 3rd part: in_hdr
00191 if (next(_i2c->mem_info(), &req))
00192 {
00193 // 2nd part was indeed a buffer
00194 if (request.out_hdr.flags.m_rd())
00195 request.write_size += request.buf_len;
00196
00197 // actual 3rd part
00198 request.in_hdr = reinterpret_cast<I2c_in_hdr *>(&req.pos);
00199 }
00200 else
00201 {
00202 // no 3rd part, 2nd part is in_hdr;
00203 request.in_hdr = reinterpret_cast<I2c_in_hdr *>(request.buf);
00204 request.buf = nullptr;
00205 request.buf_len = 0;
00206 }
00207
00208 return request;
00209 }
00210
00211 void handle_request()
00212 {
00213 if (!_head)
00214 if (!init_queue())
00215 return;
00216
00217 using Consumed_entry =
00218 cxx::Pair<L4virtio::Svr::Virtqueue::Head_desc, l4_uint32_t>;
00219 std::vector<Consumed_entry> consumed;
00220
00221 for (;;)
00222 {
00223 auto r = get_request();
00224 if (_fail_next)
00225 {
00226 r.set_status(I2c_msg_err);
00227 _fail_next = r.out_hdr.flags.fail_next();
00228 }
00229 else
00230 {
00231 bool ok;
00232 l4_uint16_t i2c_addr = r.out_hdr.addr >> 1;
00233 if (r.out_hdr.flags.m_rd())
00234 ok = _req_handler->handle_read(i2c_addr, r.buf, r.buf_len);
00235 else
00236 ok = _req_handler->handle_write(i2c_addr, r.buf, r.buf_len);

```

```

00237 if (ok)
00238 {
00239 r.set_status(I2c_msg_ok);
00240 _fail_next = false;
00241 }
00242 else
00243 {
00244 r.set_status(I2c_msg_err);
00245 _fail_next = r.out_hdr.flags.fail_next();
00246 }
00247 }
00248 consumed.emplace_back(_head, r.write_size);
00249 if (!init_queue())
00250 break;
00251 }
00252
00253 _q->finish(consumed.begin(), consumed.end(), _i2c);
00254
00255 _head = Virtqueue::Head_desc();
00256 }
00257
00258 private:
00259 L4virtio::Svr::Virtqueue *_q;
00260 I2c_request_handler *_req_handler;
00261 Virtio_i2c *_i2c;
00262 L4virtio::Svr::Virtqueue::Head_desc _head;
00263 Data_buffer _req;
00264 bool _fail_next;
00265 };
00266
00267 struct Features : public L4virtio::Svr::Dev_config::Features
00268 {
00269 Features() = default;
00270 Features(L4_uint32_t raw) : L4virtio::Svr::Dev_config::Features(raw) {}
00271
00272 // This feature is mandatory. The driver is requested to abort communication
00273 // if this is not offered.
00274 CXX_BITFIELD_MEMBER(0, 0, zero_length_request, raw);
00275 };
00276
00277 Virtio_i2c(I2c_request_handler *hndlr, L4Re::Util::Object_registry *registry)
00278 : L4virtio::Svr::Device(&_dev_config),
00279 _dev_config(L4VIRTIO_VENDOR_KK, L4VIRTIO_ID_I2C, Num_request_queues),
00280 _req_handler(hndlr),
00281 _host_irq(this),
00282 _request_processor(&_q, hndlr, this)
00283 {
00284 init_mem_info(2);
00285 reset_queue_config(0, queue_size);
00286 setup_queue(&_q, 0, queue_size);
00287 registry->register_irq_obj(&_host_irq);
00288
00289 Features hf(0);
00290 hf.ring_indirect_desc() = true;
00291 hf.zero_length_request() = true;
00292 _dev_config.host_features(0) = hf.raw;
00293 _dev_config.set_host_feature(L4VIRTIO_FEATURE_VERSION_1);
00294 _dev_config.reset_hdr();
00295 }
00296
00297 void notify_queue(L4virtio::Svr::Virtqueue *)
00298 {
00299 if (_q.no_notify_guest())
00300 return;
00301
00302 _dev_config.add_irq_status(L4VIRTIO_IRQ_STATUS_VRING);
00303 L4Re::chkipc(_notify_guest_irq->trigger(), "trigger guest irq");
00304 }
00305
00306 void handle_queue()
00307 {
00308 _request_processor.handle_request();
00309 }
00310
00311 void reset() override
00312 {
00313 }
00314
00315 bool check_queues() override
00316 {
00317 return true;
00318 }
00319
00320 int reconfig_queue(unsigned idx) override
00321 {
00322 if (idx != 0)
00323 return -L4_ERANGE;

```

```

00324
00325 setup_queue(&q, 0, queue_size);
00326
00327 return L4_EOK;
00328 }
00329
00330 void trigger_driver_config_irq() override
00331 {
00332 _dev_config.add_irq_status(L4VIRTIO_IRQ_STATUS_CONFIG);
00333 _notify_guest_irq->trigger();
00334 }
00335
00336 L4::Ipc_svr::Server_iface *server_iface() const override
00337 {
00338 return L4::Epiface::server_iface();
00339 }
00340
00341 long op_set_status(L4virtio::Device::Rights r, unsigned status)
00342 {
00343 return L4virtio::Svr::Device::op_set_status(r, status);
00344 }
00345
00346 long op_config_queue(L4virtio::Device::Rights r, unsigned queue)
00347 {
00348 return L4virtio::Svr::Device::op_config_queue(r, queue);
00349 }
00350
00351 long op_device_config(L4virtio::Device::Rights r,
00352 L4::Ipc::Cap<L4Re::Dataspace> &config_ds,
00353 l4_addr_t &ds_offset)
00354 {
00355 return L4virtio::Svr::Device::op_device_config(r, config_ds, ds_offset);
00356 }
00357
00358 L4::Cap<L4::Irq> device_notify_irq() const override
00359 {
00360 return L4::cap_cast<L4::Irq>(_host_irq.obj_cap());
00361 }
00362
00363 void register_single_driver_irq() override
00364 {
00365 _notify_guest_irq = L4Re::chkcapi
00366 (server_iface()->template rcv_cap<L4::Irq>(0));
00367 L4Re::chksys(server_iface()->realloc_rcv_cap(0));
00368 }
00369
00370
00371 private:
00372 L4virtio::Svr::Dev_config_t<L4virtio::Svr::No_custom_data> _dev_config;
00373 I2c_request_handler *_req_handler;
00374 L4virtio::Svr::Virtqueue _q;
00375 Host_irq _host_irq;
00376 L4::Cap<L4::Irq> _notify_guest_irq;
00377 Request_processor _request_processor;
00378 };
00379
00380
00381 } // namespace Svr
00382 } // namespace L4virtio

```

## 17.260 virtio-rng-device

```

00001 // vi:ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * Copyright (C) 2024 Kernkonzept GmbH.
00004 * Author(s): Martin Kuettler <martin.kuettler@kernkonzept.com>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00009 #pragma once
00010
00011 #include <l4/re/error_helper>
00012 #include <l4/sys/cxx/ipc_epiface>
00013
00014 #include <l4/l4virtio/server/virtio>
00015 #include <l4/l4virtio/server/l4virtio>
00016 #include <l4/l4virtio/l4virtio>
00017
00018 namespace L4virtio {
00019 namespace Svr {
00020
00021 template <typename Rnd_state, typename Epiface = L4virtio::Device>

```

```

00033 class Virtio_rng : public L4virtio::Svr::Device,
00034 public L4::Epiface_t<Virtio_rng<Rnd_state>, Epiface>
00035 {
00036 private:
00037 enum
00038 {
00039 Num_request_queues = 1,
00040 queue_size = 128,
00041 };
00042
00043 public:
00044 using Random_state = Rnd_state;
00045
00051 class Host_irq : public L4::Irqep_t<Host_irq>
00052 {
00053 public:
00054 explicit Host_irq(Virtio_rng *rng) : L4::Irqep_t<Host_irq>(), _rng(rng) {}
00055
00056 void handle_irq()
00057 {
00058 _rng->handle_queue();
00059 }
00060
00061 private:
00062 Virtio_rng *_rng;
00063 };
00064
00068 class Request_processor : public L4virtio::Svr::Request_processor
00069 {
00070 public:
00071 struct Data_buffer : public L4virtio::Svr::Data_buffer
00072 {
00073 Data_buffer() = default;
00074 // This constructor is called from within start, so make it available.
00075 Data_buffer(L4virtio::Svr::Driver_mem_region const *r,
00076 L4virtio::Svr::Virtqueue::Desc const &d,
00077 L4virtio::Svr::Request_processor const *)
00078 {
00079 pos = static_cast<char *>(r->local(d.addr));
00080 left = d.len;
00081 }
00082 };
00083
00084 Request_processor(L4virtio::Svr::Virtqueue *q, Random_state *rnd,
00085 Virtio_rng *rng)
00086 : _q(q), _rnd(rnd), _rng(rng), _head() {}
00087
00088 bool init_queue()
00089 {
00090 auto r = _q->next_avail();
00091
00092 if (L4_UNLIKELY(!r))
00093 return false;
00094
00095 _head = start(_rng->mem_info(), r, &_req);
00096
00097 return true;
00098 }
00099
00100 void handle_request()
00101 {
00102 if (!_head)
00103 if (!init_queue())
00104 return;
00105
00106 for (;;)
00107 {
00108 auto const pos = reinterpret_cast<unsigned char *>(_req.pos);
00109 _rnd->get_random(_req.left, pos);
00110 _q->finish(_head, _rng, _req.left);
00111 if (!init_queue())
00112 break;
00113 }
00114 return;
00115 }
00116
00117 private:
00118 L4virtio::Svr::Virtqueue *_q;
00119 Random_state *_rnd;
00120 Virtio_rng *_rng;
00121 L4virtio::Svr::Virtqueue::Head_desc _head;
00122 Data_buffer _req;
00123 };
00124
00125 Virtio_rng(Random_state *rnd, L4::Registry_iface *registry)
00126 : L4virtio::Svr::Device(&_dev_config),
00127 _dev_config(L4VIRTIO_VENDOR_KK, L4VIRTIO_ID_RNG, Num_request_queues),

```

```

00128 _rnd(rnd),
00129 _host_irq(this),
00130 _request_processor(&_q, rnd, this)
00131 {
00132 init_mem_info(2);
00133 reset_queue_config(0, queue_size);
00134 setup_queue(&_q, 0, queue_size);
00135 registry->register_irq_obj(&_host_irq);
00136
00137 L4virtio::Svr::Dev_config::Features hf;
00138 hf.ring_indirect_desc() = true;
00139 _dev_config.host_features(0) = hf.raw;
00140 _dev_config.set_host_feature(L4VIRTIO_FEATURE_VERSION_1);
00141 _dev_config.reset_hdr();
00142 }
00143
00144 void notify_queue(L4virtio::Svr::Virtqueue *)
00145 {
00146 if (_q.no_notify_guest())
00147 return;
00148
00149 _dev_config.add_irq_status(L4VIRTIO_IRQ_STATUS_VRING);
00150 L4Re::chkipc(_notify_guest_irq->trigger(), "trigger guest irq");
00151 }
00152
00153 void handle_queue()
00154 {
00155 _request_processor.handle_request();
00156 }
00157
00158 void reset() override
00159 {
00160 }
00161
00162 bool check_queues() override
00163 {
00164 return true;
00165 }
00166
00167 int reconfig_queue(unsigned idx) override
00168 {
00169 if (idx != 0)
00170 return -L4_ERANGE;
00171
00172 setup_queue(&_q, 0, queue_size);
00173
00174 return L4_EOK;
00175 }
00176
00177 void trigger_driver_config_irq() override
00178 {
00179 _dev_config.add_irq_status(L4VIRTIO_IRQ_STATUS_CONFIG);
00180 _notify_guest_irq->trigger();
00181 }
00182
00183 L4::Ipc_svr::Server_iface *server_iface() const override
00184 {
00185 return L4::Epiface::server_iface();
00186 }
00187
00188 long op_set_status(L4virtio::Device::Rights r, unsigned status)
00189 {
00190 return L4virtio::Svr::Device::op_set_status(r, status);
00191 }
00192
00193 long op_config_queue(L4virtio::Device::Rights r, unsigned queue)
00194 {
00195 return L4virtio::Svr::Device::op_config_queue(r, queue);
00196 }
00197
00198 long op_device_config(L4virtio::Device::Rights r,
00199 L4::Ipc::Cap<L4Re::Dataspace> &config_ds,
00200 l4_addr_t &ds_offset)
00201 {
00202 return L4virtio::Svr::Device::op_device_config(r, config_ds, ds_offset);
00203 }
00204
00205 L4::Cap<L4::Irq> device_notify_irq() const override
00206 {
00207 return L4::cap_cast<L4::Irq>(_host_irq.obj_cap());
00208 }
00209
00210 void register_single_driver_irq() override
00211 {
00212 _notify_guest_irq = L4Re::chkcap
00213 (server_iface()->template rcv_cap<L4::Irq>(0));
00214 }

```

```

00215 L4Re::chksys(server_iface()->realloc_rcv_cap(0));
00216 }
00217
00218
00219 private:
00220 L4virtio::Svr::Dev_config_t<L4virtio::Svr::No_custom_data>_dev_config;
00221 Random_state *_rnd;
00222 L4virtio::Svr::Virtqueue _q;
00223 Host_irq _host_irq;
00224 L4::Cap<L4::Irq> _notify_guest_irq;
00225 Request_processor _request_processor;
00226 };
00227
00228 } // namespace Svr
00229 } // namespace L4virtio

```

## 17.261 virtio-scmi-device

```

00001 // vi:ft=c++
00002 /* SPDX-License-Identifier: MIT */
00003 /*
00004 * Copyright (C) 2024 Kernkonzept GmbH.
00005 * Author(s): Christian Pötzsch <christian.poetzsch@kernkonzept.com>
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 #pragma once
00010
00011 #include <l4/cxx/bitmap>
00012 #include <l4/l4virtio/l4virtio>
00013 #include <l4/l4virtio/server/l4virtio>
00014 #include <l4/l4virtio/server/virtio>
00015 #include <l4/re/util/object_registry>
00016
00017 #include <map>
00018 #include <vector>
00019
00020 namespace L4virtio { namespace Svr { namespace Scmi {
00021
00022 enum
00023 {
00024 Version = 0x20000
00025 };
00026
00027 enum
00028 {
00029 Success = 0,
00030 Not_supported = -1,
00031 Invalid_parameters = -2,
00032 Denied = -3,
00033 Not_found = -4,
00034 Out_of_range = -5,
00035 Busy = -6,
00036 Comms_error = -7,
00037 Generic_error = -8,
00038 Hardware_error = -9,
00039 Protocol_error = -10
00040 };
00041
00042 struct Scmi_hdr_t
00043 {
00044 l4_uint32_t hdr_raw = 0;
00045 CXX_BITFIELD_MEMBER(18, 27, token, hdr_raw);
00046 CXX_BITFIELD_MEMBER(10, 17, protocol_id, hdr_raw);
00047 CXX_BITFIELD_MEMBER(8, 9, message_type, hdr_raw);
00048 CXX_BITFIELD_MEMBER(0, 7, message_id, hdr_raw);
00049 };
00050
00051 enum
00052 {
00053 Base_protocol = 0x10,
00054 Power_domain_management_protocol = 0x11,
00055 System_power_management_protocol = 0x12,
00056 Performance_domain_management_protocol = 0x13,
00057 Clock_management_protocol = 0x14,
00058 Sensor_management_protocol = 0x15,
00059 Reset_domain_management_protocol = 0x16,
00060 Voltage_domain_management_protocol = 0x17
00061 };
00062
00063 enum
00064 {
00065 Protocol_version = 0x0,

```



```

00071 Protocol_attributes = 0x1,
00072 Protocol_message_attributes = 0x2,
00073 };
00074
00076 enum
00077 {
00078 Base_discover_vendor = 0x3,
00079 Base_discover_sub_vendor = 0x4,
00080 Base_discover_implementation_version = 0x5,
00081 Base_discover_list_protocols = 0x6,
00082 Base_discover_agent = 0x7,
00083 Base_notify_errors = 0x8,
00084 Base_set_device_permissions = 0x9,
00085 Base_set_protocol_permissions = 0xa,
00086 Base_reset_agent_configuration = 0xb
00087 };
00088
00090 struct Base_attr_t
00091 {
00092 14_uint32_t attr_raw = 0;
00093 CXX_BITFIELD_MEMBER(8, 15, nagents, attr_raw);
00094 CXX_BITFIELD_MEMBER(0, 7, nprots, attr_raw);
00095 };
00096
00098 enum
00099 {
00100 Performance_domain_attributes = 0x3,
00101 Performance_describe_levels = 0x4,
00102 Performance_limits_set = 0x5,
00103 Performance_limits_get = 0x6,
00104 Performance_level_set = 0x7,
00105 Performance_level_get = 0x8,
00106 Performance_notify_limits = 0x9,
00107 Performance_notify_level = 0xa,
00108 Performance_describe_fastchannel = 0xb,
00109 };
00110
00112 struct Performance_attr_t
00113 {
00114 14_uint32_t attr_raw = 0;
00115 CXX_BITFIELD_MEMBER(16, 16, power, attr_raw);
00116 CXX_BITFIELD_MEMBER(0, 15, domains, attr_raw);
00117
00118 14_uint32_t stat_addr_low = 0;
00119 14_uint32_t stat_addr_high = 0;
00120 14_uint32_t stat_len = 0;
00121 };
00122
00124 struct Performance_domain_attr_t
00125 {
00126 14_uint32_t attr_raw = 0;
00127 CXX_BITFIELD_MEMBER(31, 31, set_limits, attr_raw);
00128 CXX_BITFIELD_MEMBER(30, 30, set_perf_level, attr_raw);
00129 CXX_BITFIELD_MEMBER(29, 29, perf_limits_change_notify, attr_raw);
00130 CXX_BITFIELD_MEMBER(28, 28, perf_level_change_notify, attr_raw);
00131 CXX_BITFIELD_MEMBER(27, 27, fast_channel, attr_raw);
00132
00133 14_uint32_t rate_limit_raw = 0;
00134 CXX_BITFIELD_MEMBER(0, 19, rate_limit, rate_limit_raw);
00135
00136 14_uint32_t sustained_freq = 0;
00137 14_uint32_t sustained_perf_level = 0;
00138 char name[16] = { 0 };
00139 };
00140
00142 struct Performance_describe_levels_n_t
00143 {
00144 14_uint32_t num_levels_raw = 0;
00145 CXX_BITFIELD_MEMBER(16, 31, nremain_perf_levels, num_levels_raw);
00146 CXX_BITFIELD_MEMBER(0, 11, nperf_levels, num_levels_raw);
00147 };
00148
00150 struct Performance_describe_level_t
00151 {
00152 14_uint32_t perf_level = 0;
00153 14_uint32_t power_cost = 0;
00154 14_uint16_t trans_latency = 0;
00155 14_uint16_t res0 = 0;
00156 };
00157
00158 template<typename OBSERV>
00159 struct Queue_worker : Request_processor
00160 {
00161 Queue_worker(OBSERV *o, Virtqueue *queue)
00162 : o(o), q(queue)
00163 {}
00164

```

```

00165 bool init_queue()
00166 {
00167 auto r = q->next_avail();
00168
00169 if (L4_UNLIKELY(!r))
00170 return false;
00171
00172 head = start(o->mem_info(), r, &req);
00173
00174 return true;
00175 }
00176
00177 bool next()
00178 { return Request_processor::next(o->mem_info(), &req); }
00179
00180 void finish(l4_uint32_t total)
00181 { q->finish(head, o, total); }
00182
00183 template<typename T>
00184 l4_ssize_t read(Data_buffer *buf, T *data, l4_size_t s = sizeof(T))
00185 {
00186 buf->pos = reinterpret_cast<char *>(data);
00187 buf->left = s;
00188 l4_size_t chunk = 0;
00189 for (;;)
00190 {
00191 chunk += req.copy_to(buf);
00192 if (req.done())
00193 next();
00194 if (!buf->left)
00195 break;
00196 }
00197 if (chunk != s)
00198 return -1;
00199 return chunk;
00200 }
00201
00202 template<typename T>
00203 l4_ssize_t write(Data_buffer *buf, T *data, l4_size_t s = sizeof(T))
00204 {
00205 buf->pos = reinterpret_cast<char *>(data);
00206 buf->left = s;
00207 l4_size_t chunk = 0;
00208 for (;;)
00209 {
00210 chunk += buf->copy_to(&req);
00211 if (req.done())
00212 next();
00213 if (!buf->left)
00214 break;
00215 }
00216 if (chunk != s)
00217 return -1;
00218 return chunk;
00219 }
00220
00221 l4_ssize_t handle_request()
00222 {
00223 try
00224 {
00225 if (!head && L4_UNLIKELY(!init_queue()))
00226 return 0;
00227
00228 for (;;)
00229 {
00230 l4_ssize_t total = 0;
00231 l4_ssize_t res = 0;
00232 Scmi_hdr_t hdr;
00233 Data_buffer buf = Data_buffer(&hdr);
00234 if ((res = read(&buf, &hdr)) < 0)
00235 return res;
00236
00237 // Search/execute handler for given protocol
00238 auto proto = o->proto(hdr.protocol_id());
00239 if (proto)
00240 {
00241 if ((res = proto->handle_request(hdr, buf, this)) < 0)
00242 return res;
00243 total += res;
00244 }
00245 else
00246 {
00247 if ((res = write(&buf, &hdr)) < 0)
00248 return res;
00249 total += res;
00250
00251 l4_int32_t status = Not_supported;

```

```

00252 if ((res = write(&buf, &status)) < 0)
00253 return res;
00254 total += res;
00255 }
00256
00257 finish(total);
00258
00259 head = Virtqueue::Head_desc();
00260 if (L4_UNLIKELY(!init_queue()))
00261 return 0;
00262 }
00263 }
00264 catch (L4virtio::Svr::Bad_descriptor const &e)
00265 {
00266 return e.error;
00267 }
00268 return 0;
00269 }
00270
00271 private:
00272 struct Buffer : Data_buffer
00273 {
00274 Buffer() = default;
00275 Buffer(L4virtio::Svr::Driver_mem_region const *r,
00276 Virtqueue::Desc const &d, Request_processor const *)
00277 {
00278 pos = static_cast<char *>(r->local(d.addr));
00279 left = d.len;
00280 }
00281 };
00282
00283 Virtqueue::Head_desc head;
00284 Buffer req;
00285
00286 OBSERV *o;
00287 Virtqueue *q;
00288 };
00289
00290 template<typename OBSERV>
00291 struct Proto
00292 {
00293 virtual l4_ssize_t handle_request(Scmi_hdr_t &hdr,
00294 Data_buffer &buf,
00295 Queue_worker<OBSERV> *qw) = 0;
00296 };
00297
00298 class Scmi_dev : public L4virtio::Svr::Device
00299 {
00300 struct Features : L4virtio::Svr::Dev_config::Features
00301 {
00302 Features() = default;
00303 Features(l4_uint32_t raw) : L4virtio::Svr::Dev_config::Features(raw) {}
00304 };
00305
00306 struct Host_irq : public L4::Irqep_t<Host_irq>
00307 {
00308 Scmi_dev *c;
00309 explicit Host_irq(Scmi_dev *c) : c(c) {}
00310 void handle_irq() { c->kick(); }
00311 };
00312
00313 enum
00314 {
00315 Queue_size = 0x10
00316 };
00317
00318 public:
00319 Scmi_dev(L4Re::Util::Object_registry *registry)
00320 : L4virtio::Svr::Device(&_dev_config),
00321 _dev_config(L4VIRTIO_VENDOR_KK, L4VIRTIO_ID_SCMI, 1),
00322 _host_irq(this),
00323 _request_worker(this, &q[0])
00324 {
00325 init_mem_info(2);
00326
00327 L4Re::chkcap(registry->register_irq_obj(&_host_irq),
00328 "Register irq object");
00329
00330 Features hf(0);
00331 hf.ring_indirect_desc() = true;
00332
00333 _dev_config.host_features(0) = hf.raw;
00334
00335 _dev_config.set_host_feature(L4VIRTIO_FEATURE_VERSION_1);
00336 _dev_config.reset_hdr();
00337
00338 reset();
00339 }

```

```

00377 }
00378
00380 void add_proto(l4_uint32_t id, Proto<Scmi_dev> *proto)
00381 { _protos.insert({id, proto}); }
00382
00383 Proto<Scmi_dev> *proto(l4_uint32_t id) const
00384 {
00385 if (_protos.find(id) != _protos.end())
00386 return _protos.at(id);
00387 return nullptr;
00388 }
00389
00390 void notify_queue(L4virtio::Virtqueue *queue)
00391 {
00392 if (queue->no_notify_guest())
00393 return;
00394
00395 _dev_config.add_irq_status(L4VIRTIO_IRQ_STATUS_VRING);
00396 _kick_guest_irq->trigger();
00397 }
00398
00399 private:
00400 L4::Cap<L4::Irq> device_notify_irq() const override
00401 { return L4::cap_cast<L4::Irq>(_host_irq.obj_cap()); }
00402
00403 void register_single_driver_irq() override
00404 {
00405 _kick_guest_irq = L4Re::Util::Unique_cap<L4::Irq>(
00406 L4Re::chkcap(server_iface()->template rcv_cap<L4::Irq>(0)));
00407
00408 L4Re::chksys(server_iface()->realloc_rcv_cap(0));
00409 }
00410
00411 void kick()
00412 {
00413 if (_request_worker.handle_request() < 0)
00414 device_error();
00415 }
00416
00417 void reset() override
00418 {
00419 for (Virtqueue &q : _q)
00420 q.disable();
00421
00422 for (l4_uint32_t i = 0; i < _dev_config.num_queues(); i++)
00423 reset_queue_config(i, Queue_size);
00424 }
00425
00426 bool check_queues() override
00427 {
00428 return true;
00429 }
00430
00431 int reconfig_queue(unsigned idx) override
00432 {
00433 if (idx >= sizeof(_q) / sizeof(_q[0]))
00434 return -L4_ERANGE;
00435
00436 return setup_queue(_q + idx, idx, Queue_size);
00437 }
00438
00439 void trigger_driver_config_irq() override
00440 {
00441 _dev_config.add_irq_status(L4VIRTIO_IRQ_STATUS_CONFIG);
00442 _kick_guest_irq->trigger();
00443 }
00444
00445 L4virtio::Svr::Dev_config_t<L4virtio::Svr::No_custom_data> _dev_config;
00446 Host_irq _host_irq;
00447 L4Re::Util::Unique_cap<L4::Irq> _kick_guest_irq;
00448 Virtqueue _q[1];
00449 Queue_worker<Scmi_dev> _request_worker;
00450 std::map<l4_uint32_t, Proto<Scmi_dev> *> _protos;
00451 };
00452
00453 class Base_proto : public Proto<Scmi_dev>
00454 {
00455 virtual l4_int32_t fill_attr(Base_attr_t *attr) const = 0;
00456
00457 virtual std::vector<l4_uint32_t> prots() const = 0;
00458
00459 l4_ssize_t handle_request(Scmi_hdr_t &hdr, Data_buffer &buf,
00460 Queue_worker<Scmi_dev> *qw) override
00461 {
00462 l4_ssize_t total = 0;
00463 l4_ssize_t res = 0;
00464 switch (hdr.message_id())

```

```
00473 {
00474 case Protocol_version:
00475 {
00476 if ((res = qw->write(&buf, &hdr)) < 0)
00477 return res;
00478 total += res;
00479
00480 struct
00481 {
00482 14_int32_t status = Success;
00483 14_uint32_t version = Version;
00484 } version;
00485 if ((res = qw->write(&buf, &version)) < 0)
00486 return res;
00487 total += res;
00488 break;
00489 }
00490 case Protocol_attributes:
00491 {
00492 if ((res = qw->write(&buf, &hdr)) < 0)
00493 return res;
00494 total += res;
00495
00496 Base_attr_t ba;
00497 14_int32_t status = fill_attr(&ba);
00498 if ((res = qw->write(&buf, &status)) < 0)
00499 return res;
00500 total += res;
00501 if (status == Success)
00502 {
00503 if ((res = qw->write(&buf, &ba)) < 0)
00504 return res;
00505 total += res;
00506 }
00507 break;
00508 }
00509 case Protocol_message_attributes:
00510 {
00511 14_uint32_t msg_id = 0;
00512 if ((res = qw->read(&buf, &msg_id)) < 0)
00513 return res;
00514
00515 if ((res = qw->write(&buf, &hdr)) < 0)
00516 return res;
00517 total += res;
00518
00519 struct
00520 {
00521 14_int32_t status = Not_found;
00522 14_uint32_t attr = 0;
00523 } attr;
00524 if (msg_id >= Protocol_version &&
00525 msg_id <= Base_discover_list_protocols)
00526 attr.status = Success;
00527
00528 if ((res = qw->write(&buf, &attr)) < 0)
00529 return res;
00530 total += res;
00531 break;
00532 }
00533 case Base_discover_vendor:
00534 {
00535 if ((res = qw->write(&buf, &hdr)) < 0)
00536 return res;
00537 total += res;
00538
00539 struct
00540 {
00541 14_int32_t status = Success;
00542 14_uint8_t vendor_identifier[16] = { "L4Re" };
00543 } vendor;
00544 if ((res = qw->write(&buf, &vendor)) < 0)
00545 return res;
00546 total += res;
00547 break;
00548 }
00549 case Base_discover_sub_vendor:
00550 {
00551 if ((res = qw->write(&buf, &hdr)) < 0)
00552 return res;
00553 total += res;
00554
00555 struct
00556 {
00557 14_int32_t status = Success;
00558 14_uint8_t vendor_identifier[16] = { "Scmi" };
00559 } vendor;
```

```

00560 if ((res = qw->write(&buf, &vendor)) < 0)
00561 return res;
00562 total += res;
00563 break;
00564 }
00565 case Base_discover_implementation_version:
00566 {
00567 if ((res = qw->write(&buf, &hdr)) < 0)
00568 return res;
00569 total += res;
00570
00571 struct
00572 {
00573 l4_int32_t status = Success;
00574 l4_uint32_t version = 1;
00575 } version;
00576 if ((res = qw->write(&buf, &version)) < 0)
00577 return res;
00578 total += res;
00579 break;
00580 }
00581 case Base_discover_list_protocols:
00582 {
00583 l4_uint32_t skip = 0;
00584 if ((res = qw->read(&buf, &skip)) < 0)
00585 return res;
00586
00587 if ((res = qw->write(&buf, &hdr)) < 0)
00588 return res;
00589 total += res;
00590
00591 auto p = prots();
00592 struct
00593 {
00594 l4_int32_t status = Success;
00595 l4_uint32_t num;
00596 } proto;
00597 proto.num = p.size();
00598 if ((res = qw->write(&buf, &proto)) < 0)
00599 return res;
00600 total += res;
00601
00602 // Array of uint32 where 4 protos are packed into one uint32. So
00603 // round up to 4 bytes and fill the array byte by byte.
00604 l4_uint8_t parr[(p.size() + 3) / 4 * 4] = { 0 };
00605 for (l4_size_t i = 0; i < p.size(); i++)
00606 parr[i] = p.at(i);
00607
00608 if ((res = qw->write(&buf, parr, sizeof(parr))) < 0)
00609 return res;
00610 total += res;
00611 break;
00612 }
00613 default:
00614 {
00615 if ((res = qw->write(&buf, &hdr)) < 0)
00616 return res;
00617 total += res;
00618
00619 l4_int32_t status = Not_supported;
00620 if ((res = qw->write(&buf, &status)) < 0)
00621 return res;
00622 total += res;
00623 break;
00624 }
00625 }
00626
00627 return total;
00628 }
00629 };
00630
00631 class Perf_proto : public Proto<Scmi_dev>
00632 {
00633 virtual l4_int32_t fill_attr(Performance_attr_t *attr) const = 0;
00634 virtual l4_int32_t fill_domain_attr(l4_uint32_t domain_id,
00635 Performance_domain_attr_t *attr) const = 0;
00636 virtual l4_int32_t fill_describe_levels_n(l4_uint32_t domain_id,
00637 l4_uint32_t level_idx,
00638 Performance_describe_levels_n_t *attr) const = 0;
00639 virtual l4_int32_t fill_describe_levels(l4_uint32_t domain_id,
00640 l4_uint32_t level_idx,
00641 l4_uint32_t num,
00642 Performance_describe_level_t *attr) const = 0;

```

```

00686 virtual l4_int32_t level_set(l4_uint32_t domain_id,
00687 l4_uint32_t perf_level) = 0;
00688
00690 virtual l4_int32_t level_get(l4_uint32_t domain_id,
00691 l4_uint32_t *perf_level) const = 0;
00692
00693 l4_ssize_t handle_request(Scmi_hdr_t &hdr, Data_buffer &buf,
00694 Queue_worker<Scmi_dev> *qw) override
00695 {
00696 l4_ssize_t total = 0;
00697 l4_ssize_t res = 0;
00698 switch (hdr.message_id())
00699 {
00700 case Protocol_version:
00701 {
00702 if ((res = qw->write(&buf, &hdr)) < 0)
00703 return res;
00704 total += res;
00705
00706 struct
00707 {
00708 l4_int32_t status = Success;
00709 l4_uint32_t version = Version;
00710 } version;
00711 if ((res = qw->write(&buf, &version)) < 0)
00712 return res;
00713 total += res;
00714 break;
00715 }
00716 case Protocol_attributes:
00717 {
00718 if ((res = qw->write(&buf, &hdr)) < 0)
00719 return res;
00720 total += res;
00721
00722 Performance_attr_t pa;
00723 l4_int32_t status = fill_attr(&pa);
00724 if ((res = qw->write(&buf, &status)) < 0)
00725 return res;
00726 total += res;
00727 if (status == Success)
00728 {
00729 if ((res = qw->write(&buf, &pa)) < 0)
00730 return res;
00731 total += res;
00732 }
00733 break;
00734 }
00735 case Protocol_message_attributes:
00736 {
00737 l4_uint32_t msg_id = 0;
00738 if ((res = qw->read(&buf, &msg_id)) < 0)
00739 return res;
00740
00741 if ((res = qw->write(&buf, &hdr)) < 0)
00742 return res;
00743 total += res;
00744
00745 struct
00746 {
00747 l4_int32_t status = Not_found;
00748
00749 l4_uint32_t attr_raw = 0;
00750 CXX_BITFIELD_MEMBER(0, 0, fast_channel, attr_raw); // ignored
00751 } attr;
00752 if ((msg_id >= Protocol_version &&
00753 msg_id <= Performance_describe_levels) ||
00754 (msg_id >= Performance_level_set &&
00755 msg_id <= Performance_level_get))
00756 attr.status = Success;
00757
00758 if ((res = qw->write(&buf, &attr)) < 0)
00759 return res;
00760 total += res;
00761 break;
00762 }
00763 case Performance_domain_attributes:
00764 {
00765 l4_uint32_t domain_id = 0;
00766 if ((res = qw->read(&buf, &domain_id)) < 0)
00767 return res;
00768
00769 if ((res = qw->write(&buf, &hdr)) < 0)
00770 return res;
00771 total += res;
00772
00773 Performance_domain_attr_t attr;

```

```

00774 l4_int32_t status = fill_domain_attr(domain_id, &attr);
00775 if ((res = qw->write(&buf, &status)) < 0)
00776 return res;
00777 total += res;
00778 if (status == Success)
00779 {
00780 if ((res = qw->write(&buf, &attr)) < 0)
00781 return res;
00782 total += res;
00783 }
00784 break;
00785 }
00786 case Performance_describe_levels:
00787 {
00788 struct
00789 {
00790 l4_uint32_t domain_id = 0;
00791 l4_uint32_t level_idx = 0;
00792 } param;
00793 if ((res = qw->read(&buf, ¶m)) < 0)
00794 return res;
00795
00796 if ((res = qw->write(&buf, &hdr)) < 0)
00797 return res;
00798 total += res;
00799
00800 // First figure out how many levels we support
00801 Performance_describe_levels_n_t attr;
00802 l4_int32_t status = fill_describe_levels_n(param.domain_id, param.level_idx,
00803 &attr);
00804 if (status != Success)
00805 {
00806 // On error bail out early
00807 if ((res = qw->write(&buf, &status)) < 0)
00808 return res;
00809 total += res;
00810 break;
00811 }
00812
00813 // Now fetch the actual levels
00814 Performance_describe_level_t attr1[attr.nperf_levels().get()];
00815 status = fill_describe_levels(param.domain_id, param.level_idx,
00816 attr.nperf_levels(), attr1);
00817 if ((res = qw->write(&buf, &status)) < 0)
00818 return res;
00819 total += res;
00820 if (status == Success)
00821 {
00822 // Write both answers to the client
00823 if ((res = qw->write(&buf, &attr)) < 0)
00824 return res;
00825 total += res;
00826 if ((res = qw->write(&buf, attr1, sizeof(attr1))) < 0)
00827 return res;
00828 total += res;
00829 }
00830 break;
00831 }
00832 case Performance_level_set:
00833 {
00834 struct
00835 {
00836 l4_uint32_t domain_id;
00837 l4_uint32_t perf_level;
00838 } param;
00839 if ((res = qw->read(&buf, ¶m)) < 0)
00840 return res;
00841
00842 if ((res = qw->write(&buf, &hdr)) < 0)
00843 return res;
00844 total += res;
00845
00846 l4_int32_t status = level_set(param.domain_id, param.perf_level);
00847 if ((res = qw->write(&buf, &status)) < 0)
00848 return res;
00849 total += res;
00850 break;
00851 }
00852 case Performance_level_get:
00853 {
00854 l4_uint32_t domain_id = 0;
00855 if ((res = qw->read(&buf, &domain_id)) < 0)
00856 return res;
00857
00858 if ((res = qw->write(&buf, &hdr)) < 0)
00859 return res;
00860 total += res;

```



```

00861
00862 l4_uint32_t perf_level;
00863 l4_int32_t status = level_get(domain_id, &perf_level);
00864 if ((res = qw->write(&buf, &status)) < 0)
00865 return res;
00866 total += res;
00867 if (status == Success)
00868 {
00869 if ((res = qw->write(&buf, &perf_level)) < 0)
00870 return res;
00871 total += res;
00872 }
00873 break;
00874 }
00875 default:
00876 {
00877 if ((res = qw->write(&buf, &hdr)) < 0)
00878 return res;
00879 total += res;
00880
00881 l4_int32_t status = Not_supported;
00882 if ((res = qw->write(&buf, &status)) < 0)
00883 return res;
00884 total += res;
00885 break;
00886 }
00887 }
00888 return total;
00889 }
00890 };
00891
00892 } /* Scmi */ } /* Svr */ } /* L4virtio */

```

## 17.262 virtio.h

```

00001 /* SPDX-License-Identifier: MIT */
00002 /*
00003 * Copyright (C) 2013-2022, 2024 Kernkonzept GmbH.
00004 * Author(s): Alexander Warg <alexander.warg@kernkonzept.com>
00005 * Matthias Lange <matthias.lange@kernkonzept.com>
00006 *
00007 */
00008 #pragma once
00009
00010
00011
00012
00013
00014
00015
00016
00017
00018
00019
00020
00021
00022
00023
00024
00025
00026
00027
00028 #include <l4/sys/compiler.h>
00029 #include <l4/sys/utcb.h>
00030 #include <l4/sys/ipc.h>
00031 #include <l4/sys/types.h>
00032
00033 enum L4_virtio_protocol
00034 {
00035 L4VIRTIO_PROTOCOL = 0,
00036 };
00037
00038 enum L4virtio_magic
00039 {
00040 L4VIRTIO_MAGIC = 0x74726976
00041 };
00042
00043 enum L4virtio_vendor
00044 {
00045 L4VIRTIO_VENDOR_KK = 0x44
00046 };
00047
00048 enum L4_virtio_opcodes
00049 {
00050 L4VIRTIO_OP_SET_STATUS = 0,
00051 L4VIRTIO_OP_CONFIG_QUEUE = 1,
00052 L4VIRTIO_OP_REGISTER_DS = 3,
00053 L4VIRTIO_OP_DEVICE_CONFIG = 4,
00054 L4VIRTIO_OP_GET_DEVICE_IRQ = 5,
00055 };
00056
00057 enum L4virtio_device_ids
00058 {
00059 L4VIRTIO_ID_NET = 1,
00060 L4VIRTIO_ID_BLOCK = 2,
00061 L4VIRTIO_ID_CONSOLE = 3,
00062 L4VIRTIO_ID_RNG = 4,
00063 L4VIRTIO_ID_BALLOON = 5,
00064 L4VIRTIO_ID_RPMMSG = 7,
00065 };

```

```

00070 L4VIRTIO_ID_SCSI = 8,
00071 L4VIRTIO_ID_9P = 9,
00072 L4VIRTIO_ID_RPROC_SERIAL = 11,
00073 L4VIRTIO_ID_CAIF = 12,
00074 L4VIRTIO_ID_GPU = 16,
00075 L4VIRTIO_ID_INPUT = 18,
00076 L4VIRTIO_ID_VSOCK = 19,
00077 L4VIRTIO_ID_CRYPT = 20,
00078 L4VIRTIO_ID_FS = 26,
00079 L4VIRTIO_ID_SCSI = 32,
00080 L4VIRTIO_ID_I2C = 34,
00081 L4VIRTIO_ID_GPIO = 41,
00082
00083 L4VIRTIO_ID_SOCK = 0x9999,
00084 };
00085
00087 enum L4virtio_device_status
00088 {
00089 L4VIRTIO_STATUS_ACKNOWLEDGE = 1,
00090 L4VIRTIO_STATUS_DRIVER = 2,
00091 L4VIRTIO_STATUS_DRIVER_OK = 4,
00092 L4VIRTIO_STATUS_FEATURES_OK = 8,
00093 L4VIRTIO_STATUS_DEVICE_NEEDS_RESET = 0x40,
00094 L4VIRTIO_STATUS_FAILED = 0x80
00095 };
00096
00098 enum L4virtio_feature_bits
00099 {
00101 L4VIRTIO_FEATURE_VERSION_1 = 32,
00103 L4VIRTIO_FEATURE_CMD_CONFIG = 160
00104 };
00105
00110 enum L4_virtio_irq_status
00111 {
00112 L4VIRTIO_IRQ_STATUS_VRING = 1,
00113 L4VIRTIO_IRQ_STATUS_CONFIG = 2,
00114 };
00115
00119 enum L4_virtio_cmd
00120 {
00121 L4VIRTIO_CMD_NONE = 0x00000000,
00122 L4VIRTIO_CMD_SET_STATUS = 0x01000000,
00123 L4VIRTIO_CMD_CFG_QUEUE = 0x02000000,
00124 L4VIRTIO_CMD_CFG_CHANGED = 0x04000000,
00125 L4VIRTIO_CMD_NOTIFY_QUEUE = 0x08000000,
00126 L4VIRTIO_CMD_MASK = 0xff000000,
00127 };
00128
00132 typedef struct l4virtio_config_hdr_t
00133 {
00134 /* Virtio(0x00): device config */
00135 l4_uint32_t magic;
00136 l4_uint32_t version;
00137 l4_uint32_t device;
00138 l4_uint32_t vendor;
00139
00140 /* Virtio(0x10): device features */
00141 l4_uint32_t dev_features;
00142 l4_uint32_t dev_features_sel;
00143 l4_uint32_t _res1[2];
00144
00145 /* Virtio(0x20): driver features */
00146 l4_uint32_t driver_features;
00147 l4_uint32_t driver_features_sel;
00148
00149 /* L4Virtio(0x28): L4 queue */
00150 l4_uint32_t num_queues;
00151 l4_uint32_t queues_offset;
00152
00153 /* Virtio(0x30): queue status */
00154 l4_uint32_t queue_sel;
00155 l4_uint32_t queue_num_max;
00156 l4_uint32_t queue_num;
00157 l4_uint32_t _res3[2];
00158 l4_uint32_t queue_ready;
00159 l4_uint32_t _res4[2];
00160
00161 /* Virtio(0x50): queue notify */
00162 l4_uint32_t queue_notify;
00163 l4_uint32_t _res5[3];
00164
00165 /* Virtio(0x60): interrupt handling */
00166 l4_uint32_t irq_status;
00167 l4_uint32_t irq_ack;
00168 l4_uint32_t _res6[2];
00169
00170 /* Virtio(0x70): Device status register (read-only). The register must be

```

```

00171 * written using l4virtio_set_status(). */
00172 l4_uint32_t status;
00173
00174 /* L4Virtio(0x74): W: Event index to be used for config notifications (device to driver) */
00175 l4_uint32_t cfg_driver_notify_index;
00176 /* L4Virtio(0x78): R: Event index to be used for config notifications (driver to device) */
00177 l4_uint32_t cfg_device_notify_index;
00178
00179 /* L4Virtio(0x7c) L4 specific command register polled by the driver iff supported */
00180 l4_uint32_t cmd;
00181
00182 /* Virtio(0x80): queue descriptors */
00183 l4_uint64_t queue_desc;
00184 l4_uint32_t _res8[2];
00185 l4_uint64_t queue_avail;
00186 l4_uint32_t _res9[2];
00187 l4_uint64_t queue_used;
00188
00189 l4_uint32_t _res10[1];
00190
00191 /* Virtio(0xac): shared memory region */
00192 l4_uint32_t shm_sel;
00193 l4_uint64_t shm_len;
00194 l4_uint64_t shm_base;
00195
00196 /* L4Virtio(0xc0): use the unused space here for device and driver feature bitmaps */
00197 l4_uint32_t dev_features_map[6];
00198 l4_uint32_t _res11[2];
00199 l4_uint32_t driver_features_map[6];
00200 l4_uint32_t _res12[1];
00201
00202 /* Virtio(0xfc): config generation */
00203 l4_uint32_t generation;
00204 } l4virtio_config_hdr_t;
00205
00223 typedef struct l4virtio_config_queue_t
00224 {
00226 l4_uint16_t num_max;
00228 l4_uint16_t num;
00229
00231 l4_uint16_t ready;
00232
00234 l4_uint16_t driver_notify_index;
00235
00236 l4_uint64_t desc_addr;
00237 l4_uint64_t avail_addr;
00238 l4_uint64_t used_addr;
00239
00241 l4_uint16_t device_notify_index;
00242 } l4virtio_config_queue_t;
00243
00244 L4_BEGIN_DECLS
00245
00251 L4_INLINE l4virtio_config_queue_t *
00252 l4virtio_config_queues(l4virtio_config_hdr_t const *cfg)
00253 {
00254 return (l4virtio_config_queue_t *)(((l4_addr_t)cfg) + cfg->queues_offset);
00255 }
00256
00262 L4_INLINE void *
00263 l4virtio_device_config(l4virtio_config_hdr_t const *cfg)
00264 {
00265 return (void *)(((l4_addr_t)cfg) + 0x100);
00266 }
00267
00271 L4_INLINE void
00272 l4virtio_set_feature(l4_uint32_t *feature_map, unsigned feat)
00273 {
00274 unsigned idx = feat / 32;
00275
00276 if (idx < 8)
00277 feature_map[idx] |= 1UL << (feat % 32);
00278 }
00279
00283 L4_INLINE void
00284 l4virtio_clear_feature(l4_uint32_t *feature_map, unsigned feat)
00285 {
00286 unsigned idx = feat / 32;
00287
00288 if (idx < 8)
00289 feature_map[idx] &= ~(1UL << (feat % 32));
00290 }
00291
00295 L4_INLINE unsigned
00296 l4virtio_get_feature(l4_uint32_t *feature_map, unsigned feat)
00297 {
00298 unsigned idx = feat / 32;

```

```

00299
00300 if (idx >= 8)
00301 return 0;
00302
00303 return feature_map[idx] & (1UL << (feat % 32));
00304 }
00305
00311 L4_CV int
00312 l4virtio_set_status(l4_cap_idx_t cap, unsigned status) L4_NOTHROW;
00313
00319 L4_CV int
00320 l4virtio_config_queue(l4_cap_idx_t cap, unsigned queue) L4_NOTHROW;
00321
00327 L4_CV int
00328 l4virtio_register_ds(l4_cap_idx_t cap, l4_cap_idx_t ds_cap,
00329 l4_uint64_t base, l4_umword_t offset,
00330 l4_umword_t size) L4_NOTHROW;
00331
00337 L4_CV int
00338 l4virtio_device_config_ds(l4_cap_idx_t cap, l4_cap_idx_t config_ds,
00339 l4_addr_t *ds_offset) L4_NOTHROW;
00340
00346 L4_CV int
00347 l4virtio_device_notification_irq(l4_cap_idx_t cap, unsigned index,
00348 l4_cap_idx_t irq) L4_NOTHROW;
00349
00350 L4_END_DECLS
00351

```

## 17.263 virtio\_block.h

```

00001 /* SPDX-License-Identifier: MIT */
00002 /*
00003 * (c) 2014 Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00004 */
00005
00006 #pragma once
00007
00013
00014 #include <l4/sys/types.h>
00015
00019 enum L4virtio_block_operations
00020 {
00021 L4VIRTIO_BLOCK_T_IN = 0,
00022 L4VIRTIO_BLOCK_T_OUT = 1,
00023 L4VIRTIO_BLOCK_T_FLUSH = 4,
00024 L4VIRTIO_BLOCK_T_GET_ID = 8,
00025 L4VIRTIO_BLOCK_T_DISCARD = 11,
00026 L4VIRTIO_BLOCK_T_WRITE_ZEROES = 13,
00027 };
00028
00032 enum L4virtio_block_status
00033 {
00034 L4VIRTIO_BLOCK_S_OK = 0,
00035 L4VIRTIO_BLOCK_S_IOERR = 1,
00036 L4VIRTIO_BLOCK_S_UNSUPP = 2,
00037 };
00038
00042 typedef struct l4virtio_block_header_t
00043 {
00044 l4_uint32_t type;
00045 l4_uint32_t ioprio;
00046 l4_uint64_t sector;
00047 } l4virtio_block_header_t;
00048
00049 enum L4virtio_block_discard_flags_t
00050 {
00051 L4VIRTIO_BLOCK_DISCARD_F_UNMAP = 0x00000001UL,
00052 L4VIRTIO_BLOCK_DISCARD_F_RESERVED = 0xFFFFFFFFUL,
00053 };
00054
00058 typedef struct l4virtio_block_discard_t
00059 {
00060 l4_uint64_t sector;
00061 l4_uint32_t num_sectors;
00062 l4_uint32_t flags;
00063 } l4virtio_block_discard_t;
00064
00068 typedef struct l4virtio_block_config_t
00069 {
00070 l4_uint64_t capacity;
00071 l4_uint32_t size_max;
00072 l4_uint32_t seg_max;

```

```

00073 struct l4virtio_block_config_geometry_t
00074 {
00075 l4_uint16_t cylinders;
00076 l4_uint8_t heads;
00077 l4_uint8_t sectors;
00078 } geometry;
00079 l4_uint32_t blk_size;
00080 struct l4virtio_block_config_topology_t
00081 {
00082 l4_uint8_t physical_block_exp;
00083 l4_uint8_t alignment_offset;
00084 l4_uint16_t min_io_size;
00085 l4_uint32_t opt_io_size;
00086 } topology;
00087 l4_uint8_t writeback;
00088 l4_uint8_t unused0[1];
00089 l4_uint16_t num_queues;
00090 l4_uint32_t max_discard_sectors;
00091 l4_uint32_t max_discard_seg;
00092 l4_uint32_t discard_sector_alignment;
00093 l4_uint32_t max_write_zeroes_sectors;
00094 l4_uint32_t max_write_zeroes_seg;
00095 l4_uint8_t write_zeroes_may_unmap;
00096 l4_uint8_t unused1[3];
00097 } l4virtio_block_config_t;
00098
00099

```

## 17.264 virtio\_input.h

```

00001 /* SPDX-License-Identifier: MIT */
00002 /*
00003 * Copyright (C) 2019, 2022, 2024 Kernkonzept GmbH.
00004 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00005 */
00006 #pragma once
00007
00013
00014 #include <l4/sys/types.h>
00015
00019 enum l4virtio_input_config_select
00020 {
00021 L4VIRTIO_INPUT_CFG_UNSET = 0,
00022 L4VIRTIO_INPUT_CFG_ID_NAME = 1,
00023 L4VIRTIO_INPUT_CFG_ID_SERIAL = 2,
00024 L4VIRTIO_INPUT_CFG_ID_DEVIDS = 3,
00025 L4VIRTIO_INPUT_CFG_PROP_BITS = 0x10,
00026 L4VIRTIO_INPUT_CFG_EV_BITS = 0x11,
00027 L4VIRTIO_INPUT_CFG_ABS_INFO = 0x12
00028 };
00029
00033 typedef struct l4virtio_input_absinfo_t
00034 {
00035 l4_uint32_t min;
00036 l4_uint32_t max;
00037 l4_uint32_t fuzz;
00038 l4_uint32_t flat;
00039 l4_uint32_t res;
00040 } l4virtio_absinfo_t;
00041
00045 typedef struct l4virtio_input_devids_t
00046 {
00047 l4_uint16_t bustype;
00048 l4_uint16_t vendor;
00049 l4_uint16_t product;
00050 l4_uint16_t version;
00051 } l4virtio_input_devids_t;
00052
00056 typedef struct l4virtio_input_config_t
00057 {
00058 l4_uint8_t select;
00059 l4_uint8_t subselect;
00060 l4_uint8_t size;
00061 l4_uint8_t reserved[5];
00062 union
00063 {
00064 char string[128];
00065 l4_uint8_t bitmap[128];
00066 struct l4virtio_input_absinfo_t abs;
00067 struct l4virtio_input_devids_t ids;
00068 } u;
00069 } l4virtio_input_config_t;
00070
00074 typedef struct l4virtio_input_event_t

```

```

00075 {
00076 l4_uint16_t type;
00077 l4_uint16_t code;
00078 l4_uint32_t value;
00079 } l4virtio_input_event_t;
00080

```

## 17.265 virtqueue

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /* SPDX-License-Identifier: MIT */
00003 /*
00004 * (c) 2014 Alexander Warg <warg@os.inf.tu-dresden.de>
00005 */
00006
00007 #include <l4/re/util/debug>
00008 #include <l4/sys/types.h>
00009 #include <l4/sys/err.h>
00010 #include <l4/cxx/bitfield>
00011 #include <l4/cxx/exceptions>
00012 #include <stdint>
00013
00014 #pragma once
00015
00016 namespace L4virtio {
00017
00018 #if defined(__ARM_ARCH) && __ARM_ARCH == 7
00019 static inline void wmb() { asm volatile ("dmb ishst" : : : "memory"); }
00020 static inline void rmb() { asm volatile ("dmb ish" : : : "memory"); }
00021 #elif defined(__ARM_ARCH) && __ARM_ARCH >= 8
00022 static inline void wmb() { asm volatile ("dmb ishst" : : : "memory"); }
00023 static inline void rmb() { asm volatile ("dmb ishld" : : : "memory"); }
00024 #elif defined(__mips__)
00025 static inline void wmb() { asm volatile ("sync" : : : "memory"); }
00026 static inline void rmb() { asm volatile ("sync" : : : "memory"); }
00027 #elif defined(__amd64__) || defined(__i386__) || defined(__i686__)
00028 static inline void wmb() { asm volatile ("sfence" : : : "memory"); }
00029 static inline void rmb() { asm volatile ("lfence" : : : "memory"); }
00030 #elif defined(__riscv)
00031 static inline void wmb() { asm volatile ("fence ow, ow" : : : "memory"); }
00032 static inline void rmb() { asm volatile ("fence ir, ir" : : : "memory"); }
00033 #else
00034 #warning Missing proper memory write barrier
00035 static inline void wmb() { asm volatile ("": : : : "memory"); }
00036 static inline void rmb() { asm volatile ("": : : : "memory"); }
00037 #endif
00038
00039
00040 template< typename T >
00041 class Ptr
00042 {
00043 public:
00044 enum Invalid_type { Invalid };
00045
00046 Ptr() = default;
00047
00048 Ptr(Invalid_type) : _p(~0ULL) {}
00049
00050 explicit Ptr(l4_uint64_t vm_addr) : _p(vm_addr) {}
00051
00052 l4_uint64_t get() const { return _p; }
00053
00054 bool is_valid() const { return _p != ~0ULL; }
00055
00056 private:
00057 l4_uint64_t _p;
00058 };
00059
00060 class Virtqueue
00061 {
00062 public:
00063 class Desc
00064 {
00065 public:
00066 struct Flags
00067 {
00068 l4_uint16_t raw;
00069 Flags() = default;
00070
00071 explicit Flags(l4_uint16_t v) : raw(v) {}
00072
00073 CXX_BITFIELD_MEMBER(0, 0, next, raw);
00074

```

```

00103 CXX_BITFIELD_MEMBER(1, 1, write, raw);
00105 CXX_BITFIELD_MEMBER(2, 2, indirect, raw);
00106 };
00107
00108 Ptr<void> addr;
00109 l4_uint32_t len;
00110 Flags flags;
00111 l4_uint16_t next;
00112
00116 void dump(unsigned idx) const
00117 {
00118 L4Re::Util::Dbg().printf("D[%04x]: %08llx (%x) f=%04x n=%04x\n",
00119 idx, addr.get(),
00120 len, static_cast<unsigned>(flags.raw),
00121 static_cast<unsigned>(next));
00122 }
00123 };
00124
00128 class Avail
00129 {
00130 public:
00134 struct Flags
00135 {
00136 l4_uint16_t raw;
00137 Flags() = default;
00138
00140 explicit Flags(l4_uint16_t v) : raw(v) {}
00141
00143 CXX_BITFIELD_MEMBER(0, 0, no_irq, raw);
00144 };
00145
00146 Flags flags;
00147 l4_uint16_t idx;
00148 l4_uint16_t ring[];
00149 };
00150
00154 struct Used_elem
00155 {
00156 Used_elem() = default;
00157
00165 Used_elem(l4_uint16_t id, l4_uint32_t len) : id(id), len(len) {}
00166 l4_uint32_t id;
00167 l4_uint32_t len;
00168 };
00169
00173 class Used
00174 {
00175 public:
00179 struct Flags
00180 {
00181 l4_uint16_t raw;
00182 Flags() = default;
00183
00185 explicit Flags(l4_uint16_t v) : raw(v) {}
00186
00188 CXX_BITFIELD_MEMBER(0, 0, no_notify, raw);
00189 };
00190
00191 Flags flags;
00192 l4_uint16_t idx;
00193 Used_elem ring[];
00194 };
00195
00196 protected:
00197 Desc *_desc = nullptr;
00198 Avail *_avail = nullptr;
00199 Used *_used = nullptr;
00200
00202 l4_uint16_t _current_avail = 0;
00203
00208 l4_uint16_t _idx_mask = 0;
00209
00213 Virtqueue() = default;
00214
00215 Virtqueue(Virtqueue const &) = delete;
00216 ~Virtqueue() = default;
00217
00218 public:
00224 void disable()
00225 { _desc = 0; }
00226
00230 enum
00231 {
00232 Desc_align = 4, //< Alignment of the descriptor table.
00233 Avail_align = 1, //< Alignment of the available ring.
00234 Used_align = 2, //< Alignment of the used ring.
00235 };

```

```

00236
00245 static unsigned long total_size(unsigned num)
00246 {
00247 static_assert(Desc_align >= Avail_align,
00248 "virtqueue alignment assumptions broken");
00249 return l4_round_size(desc_size(num) + avail_size(num), Used_align)
00250 + used_size(num);
00251 }
00252
00261 static unsigned long desc_size(unsigned num)
00262 { return num * 16; }
00263
00269 static unsigned long desc_align()
00270 { return Desc_align; }
00271
00279 static unsigned long avail_size(unsigned num)
00280 { return 2 * num + 6; }
00281
00287 static unsigned long avail_align()
00288 { return Avail_align; }
00289
00298 static unsigned long used_size(unsigned num)
00299 { return 8 * num + 6; }
00300
00306 static unsigned long used_align()
00307 { return Used_align; }
00308
00314 unsigned long total_size() const
00315 {
00316 return (reinterpret_cast<char *>(_used) - reinterpret_cast<char *>(_desc))
00317 + used_size(num());
00318 }
00319
00323 unsigned long avail_offset() const
00324 { return reinterpret_cast<char *>(_avail) - reinterpret_cast<char *>(_desc); }
00325
00329 unsigned long used_offset() const
00330 { return reinterpret_cast<char *>(_used) - reinterpret_cast<char *>(_desc); }
00331
00349 void setup(unsigned num, void *desc, void *avail, void *used)
00350 {
00351 if (num > 0x10000)
00352 throw L4::Runtime_error(-L4_EINVAL, "Queue too large.");
00353
00354 _idx_mask = num - 1;
00355 _desc = static_cast<Desc*>(desc);
00356 _avail = static_cast<Avail*>(avail);
00357 _used = static_cast<Used*>(used);
00358
00359 _current_avail = 0;
00360
00361 L4Re::Util::Dbg().printf("VQ[%p]: num=%d d:%p a:%p u:%p\n",
00362 this, num, _desc, _avail, _used);
00363 }
00364
00378 void setup_simple(unsigned num, void *ring)
00379 {
00380 l4_addr_t desc = reinterpret_cast<l4_addr_t>(ring);
00381 l4_addr_t avail = l4_round_size(desc + desc_size(num), Avail_align);
00382 void *used = reinterpret_cast<void *>(
00383 l4_round_size(avail + avail_size(num), Used_align));
00384 setup(num, ring, reinterpret_cast<void *>(avail), used);
00385 }
00386
00392 void dump(Desc const *d) const
00393 { d->dump(d - _desc); }
00394
00400 bool ready() const
00401 { return L4_LIKELY(_desc != 0); }
00402
00404 unsigned num() const
00405 { return _idx_mask + 1; }
00406
00414 bool no_notify_guest() const
00415 {
00416 return _avail->flags.no_irq();
00417 }
00418
00426 bool no_notify_host() const
00427 {
00428 return _used->flags.no_notify();
00429 }
00430
00436 void no_notify_host(bool value)
00437 {
00438 _used->flags.no_notify() = value;
00439 }

```



```

00440
00449 l4_uint16_t get_avail_idx() const { return _avail->idx; }
00450
00456 l4_uint16_t get_tail_avail_idx() const { return _current_avail; }
00457
00458 };
00459
00460 namespace Driver {
00461
00470 class Virtqueue : public L4virtio::Virtqueue
00471 {
00472 private:
00474 l4_uint16_t _next_free;
00475
00476 public:
00477 enum End_of_queue
00478 {
00479 // Indicates the end of the queue.
00480 Eoq = 0xFFFF
00481 };
00482
00483 Virtqueue() : _next_free(Eoq) {}
00484
00494 void initialize_rings(unsigned num)
00495 {
00496 _used->idx = 0;
00497 _avail->idx = 0;
00498
00499 // setup the freelist
00500 for (l4_uint16_t d = 0; d < num - 1; ++d)
00501 _desc[d].next = d + 1;
00502 _desc[num - 1].next = Eoq;
00503 _next_free = 0;
00504 }
00505
00522 void init_queue(unsigned num, void *desc, void *avail, void *used)
00523 {
00524 setup(num, desc, avail, used);
00525 initialize_rings(num);
00526 }
00527
00537 void init_queue(unsigned num, void *base)
00538 {
00539 setup_simple(num, base);
00540 initialize_rings(num);
00541 }
00542
00543
00558 l4_uint16_t alloc_descriptor()
00559 {
00560 l4_uint16_t idx = _next_free;
00561 if (idx == Eoq)
00562 return Eoq;
00563
00564 _next_free = _desc[idx].next;
00565
00566 return idx;
00567 }
00568
00574 void enqueue_descriptor(l4_uint16_t descno)
00575 {
00576 if (descno > _idx_mask)
00577 throw L4::Bounds_error();
00578
00579 _avail->ring[_avail->idx & _idx_mask] = descno; // _avail->idx expected to wrap
00580 wmb();
00581 ++_avail->idx;
00582 }
00583
00590 Desc &desc(l4_uint16_t descno)
00591 {
00592 if (descno > _idx_mask)
00593 throw L4::Bounds_error();
00594
00595 return _desc[descno];
00596 }
00597
00609 l4_uint16_t find_next_used(l4_uint32_t *len = nullptr)
00610 {
00611 if (_current_avail == _used->idx)
00612 return Eoq;
00613
00614 auto elem = _used->ring[_current_avail++ & _idx_mask];
00615
00616 if (len)
00617 *len = elem.len;
00618

```

```

00619 return elem.id;
00620 }
00621
00631 void free_descriptor(l4_uint16_t head, l4_uint16_t tail)
00632 {
00633 if (head > _idx_mask || tail > _idx_mask)
00634 throw L4::Bounds_error();
00635
00636 _desc[tail].next = _next_free;
00637 _next_free = head;
00638 }
00639 };
00640
00641 }
00642 } // namespace L4virtio

```

## 17.266 block\_device\_mgr.h

```

00001 /*
00002 * Copyright (C) 2018-2020, 2022-2024 Kernkonzept GmbH.
00003 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00004 * Manuel von Oltersdorff-Kalettkka <manuel.kalettkka@kernkonzept.com>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include <cassert>
00011 #include <cstring>
00012 #include <memory>
00013 #include <string>
00014 #include <vector>
00015
00016 #include <l4/cxx/ref_ptr>
00017 #include <l4/cxx/ref_ptr_list>
00018 #include <l4/cxx/unique_ptr>
00019 #include <l4/re/error_helper>
00020 #include <l4/sys/factory>
00021 #include <l4/sys/cxx/ipc_epiface>
00022
00023 #include <l4/libblock-device/debug.h>
00024 #include <l4/libblock-device/errand.h>
00025 #include <l4/libblock-device/partition.h>
00026 #include <l4/libblock-device/part_device.h>
00027 #include <l4/libblock-device/virtio_client.h>
00028 #include <l4/libblock-device/scheduler.h>
00029
00030 namespace Block_device {
00031
00032 template <typename DEV>
00033 struct Simple_factory
00034 {
00035 using Device_type = DEV;
00036 using Client_type = Virtio_client<Device_type>;
00037
00038 static cxx::unique_ptr<Client_type>
00039 create_client(cxx::Ref_ptr<Device_type> const &dev,
00040 unsigned numds, bool readonly)
00041 { return cxx::make_unique<Client_type>(dev, numds, readonly); }
00042 };
00043
00044 template <typename BASE_DEV>
00045 struct Partitionable_factory
00046 {
00047 using Device_type = BASE_DEV;
00048 using Client_type = Virtio_client<Device_type>;
00049
00050 static cxx::unique_ptr<Client_type>
00051 create_client(cxx::Ref_ptr<Device_type> const &dev,
00052 unsigned numds, bool readonly)
00053 {
00054 return cxx::make_unique<Client_type>(dev, numds, readonly);
00055 }
00056
00057 static cxx::Ref_ptr<Device_type>
00058 create_partition(cxx::Ref_ptr<Device_type> const &dev, unsigned partition_id,
00059 Partition_info const &pi)
00060 {
00061 return cxx::Ref_ptr<Device_type>{
00062 new Partitioned_device<Device_type>(dev, partition_id, pi);
00063 };
00064 };
00065 };

```

```

00066
00077 template <typename DEV, typename FACTORY = Simple_factory<DEV>,
00078 typename SCHEDULER = Rr_scheduler<typename FACTORY::Device_type>
00079 class Device_mgr
00080 {
00081 using Device_factory_type = FACTORY;
00082 using Client_type = typename Device_factory_type::Client_type;
00083 using Device_type = typename Device_factory_type::Device_type;
00084 using Scheduler_type = SCHEDULER;
00085
00086 using Ds_vector = std::vector<L4::Cap<L4Re::Dataspace>;
00087
00088 using Pairing_callback = std::function<void(Device_type *)>;
00089
00093 struct Pending_client
00094 {
00096 std::string device_id;
00098 L4::Cap<L4::Rcv_endpoint> gate;
00100 int num_ds;
00102 bool readonly;
00103
00104 bool enable_trusted_ds_validation;
00105
00106 std::shared_ptr<Ds_vector const> trusted_dataspaces;
00107
00109 Pairing_callback pairing_cb;
00110
00111 Pending_client() = default;
00112
00113 Pending_client(L4::Cap<L4::Rcv_endpoint> g, std::string const &dev, int ds,
00114 bool ro, bool enable_trusted_ds_validation,
00115 std::shared_ptr<Ds_vector const> trusted_dataspaces,
00116 Pairing_callback cb)
00117 : device_id(dev), gate(g), num_ds(ds), readonly(ro),
00118 enable_trusted_ds_validation(enable_trusted_ds_validation),
00119 trusted_dataspaces(trusted_dataspaces), pairing_cb(cb)
00120 {}
00121 };
00122
00123 class Connection : public cxx::Ref_obj_list_item<Connection>
00124 {
00125 public:
00126 explicit Connection(Device_mgr *mgr, cxx::Ref_ptr<Device_type> &dev)
00127 : _shutdown_state(Shutdown_type::Running),
00128 _device(cxx::move(dev)),
00129 _mgr(mgr)
00130 {}
00131
00132 L4::Cap<void> cap() const
00133 { return _interface ? _interface->obj_cap() : L4::Cap<void>(); }
00134
00135 void start_disk_scan(Errand::Callback const &callback)
00136 {
00137 _device->start_device_scan(
00138 [=]()
00139 {
00140 scan_disk_partitions(callback, 0);
00141 });
00142 }
00143
00144 void unregister_interfaces(L4::Registry_iface *registry) const
00145 {
00146 if (_interface)
00147 registry->unregister_obj(_interface.get());
00148
00149 for (auto *sub : _subs)
00150 sub->unregister_interfaces(registry);
00151 }
00152
00153 int create_interface_for(Pending_client *c, L4::Registry_iface *registry)
00154 {
00155 if (_shutdown_state != Shutdown_type::Running)
00156 return -L4_EIO;
00157
00158 if (_interface)
00159 return contains_device(c->device_id) ? -L4_EBUSY : -L4_ENODEV;
00160
00161 // check for match in partitions
00162
00163 bool busy = false;
00164 for (auto *sub : _subs)
00165 {
00166 if (sub->_interface)
00167 busy = true;
00168
00169 int ret = sub->create_interface_for(c, registry);
00170

```

```

00171 if (ret != -L4_ENODEV) // includes L4_EOK
00172 return ret;
00173 }
00174
00175 if (!match_hid(c->device_id))
00176 return -L4_ENODEV;
00177
00178 if (busy)
00179 return -L4_EBUSY;
00180
00181 auto clt = Device_factory_type::create_client(_device, c->num_ds,
00182 c->readonly);
00183
00184 clt->add_trusted_dataspaces(c->trusted_dataspaces);
00185 if (c->enable_trusted_ds_validation)
00186 clt->enable_trusted_ds_validation();
00187
00188 if (c->gate.is_valid())
00189 {
00190 if (!clt->register_obj(registry, c->gate).is_valid())
00191 return -L4_ENOMEM;
00192 }
00193 else
00194 {
00195 c->gate = L4::cap_reinterpret_cast<L4::Rcv_endpoint>(
00196 clt->register_obj(registry));
00197 if (!c->gate.is_valid())
00198 return -L4_ENOMEM;
00199 }
00200
00201 _mgr->_scheduler->add_client(clt.get());
00202 _interface.reset(clt.release());
00203
00204 // Let it be known that the client and the device paired
00205 if (c->pairing_cb)
00206 c->pairing_cb(_device.get());
00207 return L4_EOK;
00208 }
00209
00210 void check_clients(L4::Registry_iface *registry)
00211 {
00212 if (_interface)
00213 {
00214 if (_interface->obj_cap() && !_interface->obj_cap().validate().label())
00215 remove_client(registry);
00216
00217 return;
00218 }
00219
00220 // Sub-devices only need to be checked when the parent device was free.
00221 for (auto *sub : _subs)
00222 sub->check_clients(registry);
00223 }
00224
00226 void shutdown_event(Shutdown_type type)
00227 {
00228 // Set new shutdown state
00229 _shutdown_state = type;
00230 for (auto const &sub: _subs)
00231 sub->shutdown_event(type);
00232 if (_interface)
00233 _interface->shutdown_event(type);
00234 }
00235
00236 private:
00248 template <typename T = Device_factory_type>
00249 auto scan_disk_partitions(Errand::Callback const &callback, int)
00250 -> decltype((T::create_partition)(cxx::Ref_ptr<Device_type>(), 0, Partition_info(), void()))
00251 {
00252 auto reader = cxx::make_ref_obj<Partition_reader<Device_type>>(_device.get());
00253 // The reference to reader will be captured in the lambda passed to
00254 // reader's own read() method. At the same time, reader will store
00255 // the reference to the lambda.
00256 reader->read(
00257 [=]()
00258 {
00259 l4_size_t sz = reader->table_size();
00260
00261 for (l4_size_t i = 1; i <= sz; ++i)
00262 {
00263 Partition_info info;
00264 if (reader->get_partition(i, &info) < 0)
00265 continue;
00266
00267 auto conn = cxx::make_ref_obj<Connection>(
00268 _mgr,
00269 Device_factory_type::create_partition(_device, i, info));

```

```

00270 _subs.push_front(std::move(conn));
00271 }
00272
00273 callback();
00274
00275 // Prolong the life-span of reader until we are sure the reader is
00276 // not currently invoked (i.e. capture the last reference to it in
00277 // an independent timeout callback).
00278 Errand::schedule([reader]() {}, 0);
00279 });
00280 }
00281
00282 template <typename T = Device_factory_type>
00283 void scan_disk_partitions(Errand::Callback const &callback, long)
00284 { callback(); }
00285
00286 void remove_client(L4::Registry_iface *registry)
00287 {
00288 assert(_interface);
00289
00290 // This operation is idempotent.
00291 _interface->shutdown_event(Shutdown_type::Client_gone);
00292
00293 if (_interface->busy())
00294 {
00295 Dbg::trace().printf("Deferring dead client removal.\n");
00296
00297 // Cannot remove the client while it still has active I/O requests.
00298 // This means that the device did not abort its inflight requests in
00299 // its reset() callback. It is still desirable though to wait for
00300 // those requests to finish and defer the dead client removal until
00301 // later.
00302 Errand::schedule([this, registry]() { remove_client(registry); },
00303 10000);
00304 return;
00305 }
00306
00307 _interface->unregister_obj(registry);
00308 _mgr->_scheduler->remove_client(_interface.get());
00309 _interface.reset();
00310 }
00311
00312 bool contains_device(std::string const &name) const
00313 {
00314 if (match_hid(name))
00315 return true;
00316
00317 for (auto *sub : _subs)
00318 if (sub->contains_device(name))
00319 return true;
00320
00321 return false;
00322 }
00323
00324 bool match_hid(std::string const &name) const
00325 { return _device->match_hid(cxx::String(name.c_str(), name.length())); }
00326
00327 Shutdown_type _shutdown_state;
00328 cxx::Ref_ptr<Device_type> _device;
00329 cxx::unique_ptr<Client_type> _interface;
00330 cxx::Ref_ptr_list<Connection> _subs;
00331
00332 Device_mgr *_mgr;
00333 };
00334
00335 public:
00336 Device_mgr(L4::Registry_iface *registry)
00337 : _registry(registry)
00338 {
00339 _scheduler = cxx::make_unique<Scheduler_type>(registry);
00340 }
00341
00342 virtual ~Device_mgr()
00343 {
00344 for (auto *c : _connpts)
00345 c->unregister_interfaces(_registry);
00346 }
00347
00348 static int parse_device_name(std::string const ¶m, std::string &device)
00349 {
00350 std::string const partlabel("partlabel:");
00351 std::string const partuuid("partuuid:");
00352
00353 if (param.size() > partlabel.size()
00354 && param.compare(0, partlabel.size(), partlabel) == 0)
00355 {

```

```

00391 device = param.substr(partlabel.size());
00392 return L4_EOK;
00393 }
00394 else if (param.size() > partuuid.size()
00395 && param.compare(0, partuuid.size(), partuuid) == 0)
00396 {
00397 auto device_partuuid = param.substr(partuuid.size());
00398 if (!is_uuid(device_partuuid.c_str()))
00399 {
00400 Dbg::trace().printf("The 'partuuid:' parameter expects a UUID.\n");
00401 return -L4_EINVAL;
00402 }
00403
00404 device = device_partuuid;
00405 std::transform(device.begin(), device.end(), device.begin(),
00406 [](unsigned char c){ return std::toupper(c); });
00407 return L4_EOK;
00408 }
00409 else
00410 {
00411 device = param;
00412 if (is_uuid(param.c_str()))
00413 std::transform(device.begin(), device.end(), device.begin(),
00414 [](unsigned char c) { return std::toupper(c); });
00415 return L4_EOK;
00416 }
00417 }
00418
00419 int add_static_client(L4::Cap<L4::Rcv_endpoint> client, const char *device,
00420 int partno, int num_ds, bool readonly = false,
00421 Pairing_callback cb = nullptr,
00422 bool enable_trusted_ds_validation = false,
00423 std::shared_ptr<Ds_vector const> trusted_dataspaces
00424 = nullptr)
00425 {
00426 char _buf[30];
00427 const char *buf;
00428
00429 if (partno == 0)
00430 {
00431 Err().printf("Invalid partition number 0.\n");
00432 return -L4_ENODEV;
00433 }
00434
00435 if (partno != -1)
00436 {
00437 /* Could we avoid to make a string here and parsing this again
00438 * deeper in the stack? */
00439 snprintf(_buf, sizeof(_buf), "%s:%d", device, partno);
00440 buf = _buf;
00441 }
00442 else
00443 buf = device;
00444
00445 _pending_clients.emplace_back(client, buf, num_ds, readonly,
00446 enable_trusted_ds_validation,
00447 trusted_dataspaces, cb);
00448
00449 return L4_EOK;
00450 }
00451
00452 int create_dynamic_client(std::string const &device, int partno, int num_ds,
00453 L4::Cap<void> *cap, bool readonly = false,
00454 Pairing_callback cb = nullptr,
00455 bool enable_trusted_ds_validation = false,
00456 std::shared_ptr<Ds_vector const> trusted_dataspaces
00457 = nullptr)
00458 {
00459 Pending_client clt;
00460
00461 // Maximum number of dataspaces that can be registered.
00462 clt.num_ds = num_ds;
00463
00464 clt.readonly = readonly;
00465
00466 clt.device_id = device;
00467
00468 clt.pairing_cb = cb;
00469
00470 clt.trusted_dataspaces = trusted_dataspaces;
00471
00472 clt.enable_trusted_ds_validation = enable_trusted_ds_validation;
00473
00474 if (partno > 0)
00475 {
00476 clt.device_id += ':';
00477 clt.device_id += std::to_string(partno);

```

```

00478 }
00479
00480 for (auto *c : _connpts)
00481 {
00482 int ret = c->create_interface_for(&clt, _registry);
00483
00484 if (ret == -L4_ENODEV)
00485 continue;
00486
00487 if (ret < 0)
00488 return ret;
00489
00490 // found the requested device
00491 *cap = clt.gate;
00492 return L4_EOK;
00493 }
00494
00495 return -L4_ENODEV;
00496 }
00497
00501 void check_clients()
00502 {
00503 for (auto *c : _connpts)
00504 c->check_clients(_registry);
00505 }
00506
00507 void add_disk(cxx::Ref_ptr<Device_type> &&device, Errand::Callback const &callback)
00508 {
00509 auto conn = cxx::make_ref_obj<Connection>(this, std::move(device));
00510
00511 conn->start_disk_scan(
00512 [=] ()
00513 {
00514 _connpts.push_front(conn);
00515 connect_static_clients(conn.get());
00516 callback();
00517 });
00518 }
00519
00521 void shutdown_event(Shutdown_type type)
00522 {
00523 l4_assert(type != Client_gone);
00524 l4_assert(type != Client_shutdown);
00525
00526 for (auto const &con : _connpts)
00527 con->shutdown_event(type);
00528 }
00529
00530 private:
00531 void connect_static_clients(Connection *con)
00532 {
00533 for (auto &c : _pending_clients)
00534 {
00535 Dbg::trace().printf("Checking existing client %s\n", c.device_id.c_str());
00536 if (!c.gate.is_valid())
00537 continue;
00538
00539 int ret = con->create_interface_for(&c, _registry);
00540
00541 if (ret == L4_EOK)
00542 {
00543 c.gate = L4::Cap<L4::Rcv_endpoint>();
00544 // There might be other clients waiting for other partitions.
00545 // Continue search.
00546 continue;
00547 }
00548
00549 if (ret != -L4_ENODEV)
00550 break;
00551 }
00552 }
00553
00554 static constexpr bool is_uuid(char const *s)
00555 {
00556 for (unsigned i = 0; i < 36; ++i)
00557 if (i == 8 || i == 13 || i == 18 || i == 23)
00558 {
00559 if (s[i] != '-')
00560 return false;
00561 }
00562 else
00563 {
00564 if (!isxdigit(s[i]))
00565 return false;
00566 }
00567 return s[36] == '\0';
00568 }

```

```

00569
00571 L4::Registry_iface *_registry;
00573 cxx::Ref_ptr_list<Connection> _connpts;
00575 std::vector<Pending_client> _pending_clients;
00577 cxx::unique_ptr<Scheduler_type> _scheduler;
00578 };
00579
00580 } // name space

```

## 17.267 device.h

```

00001 /*
00002 * Copyright (C) 2018-2020, 2024 Kernkonzept GmbH.
00003 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/cxx/ref_ptr>
00010 #include <l4/cxx/string>
00011 #include <l4/re/dataspace>
00012 #include <l4/re/dma_space>
00013
00014 #include <l4/libblock-device/errand.h>
00015 #include <l4/libblock-device/types.h>
00016
00017 namespace Block_device {
00018
00032 struct Notification_domain
00033 {
00034 };
00035
00036 struct Device : public cxx::Ref_obj
00037 {
00038 virtual ~Device() = 0;
00039
00041 virtual Notification_domain const *notification_domain() const = 0;
00042
00044 virtual bool is_read_only() const = 0;
00046 virtual bool match_hid(cxx::String const &hid) const = 0;
00048 virtual l4_uint64_t capacity() const = 0;
00050 virtual l4_size_t sector_size() const = 0;
00052 virtual l4_size_t max_size() const = 0;
00054 virtual unsigned max_segments() const = 0;
00055
00057 virtual void reset() = 0;
00058
00060 virtual int dma_map(Block_device::Mem_region *region, l4_addr_t offset,
00061 l4_size_t num_sectors, L4Re::Dma_space::Direction dir,
00062 L4Re::Dma_space::Dma_addr *phys) = 0;
00063
00065 virtual int dma_unmap(L4Re::Dma_space::Dma_addr phys, l4_size_t num_sectors,
00066 L4Re::Dma_space::Direction dir) = 0;
00067
00082 virtual int inout_data(l4_uint64_t sector,
00083 Block_device::Inout_block const &blocks,
00084 Block_device::Inout_callback const &cb,
00085 L4Re::Dma_space::Direction dir) = 0;
00086
00097 virtual int flush(Block_device::Inout_callback const &cb) = 0;
00098
00100 virtual void start_device_scan(Block_device::Errand::Callback const &callback) = 0;
00101 };
00102
00103 inline Device::~~Device() = default;
00104
00108 template <typename DEV>
00109 struct Device_with_notification_domain : DEV
00110 {
00111 Notification_domain dom;
00112 Notification_domain const *notification_domain() const override
00113 { return &dom; }
00114 };
00115
00119 struct Device_discard_feature
00120 {
00121 struct Discard_info
00122 {
00123 unsigned max_discard_sectors = 0;
00124 unsigned max_discard_seg = 0;
00125 unsigned discard_sector_alignment = 1;
00126 unsigned max_write_zeroes_sectors = 0;

```



```

00127 unsigned max_write_zeroes_seg = 0;
00128 bool write_zeroes_may_unmap = false;
00129 };
00130
00131 virtual Discard_info discard_info() const = 0;
00132
00133 virtual int discard(l4_uint64_t offset,
00134 Block_device::Inout_block const &blocks,
00135 Block_device::Inout_callback const &cb, bool discard) = 0;
00136
00137 protected:
00138 ~Device_discard_feature() = default;
00139 };
00140
00141
00142 } // name space

```

## 17.268 errand.h

```

00001 /*
00002 * Copyright (C) 2014, 2020, 2024 Kernkonzept GmbH.
00003 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/re/env.h>
00010 #include <l4/re/util/object_registry>
00011 #include <l4/re/util/br_manager>
00012 #include <l4/cxx/ipc_timeout_queue>
00013 #include <l4/cxx/ref_ptr>
00014 #include <l4/cxx/exceptions>
00015 #include <l4/libblock-device/debug.h>
00016
00017 #include <functional>
00018
00019 namespace Block_device { namespace Errand {
00020
00021 extern L4::Ipc_svr::Server_iface *_sif;
00022
00023 typedef std::function<void()> Callback;
00024
00025 class Poll_errand
00026 : public L4::Ipc_svr::Timeout_queue::Timeout,
00027 public cxx::Ref_obj
00028 {
00029 public:
00030 void expired() final
00031 {
00032 // Recapture the reference pointer from the timeout queue.
00033 cxx::Ref_ptr<Poll_errand> p(this, false);
00034
00035 try
00036 {
00037 if (_poll())
00038 _callback(true);
00039 else
00040 if (--_retries <= 0)
00041 _callback(false);
00042 else
00043 reschedule();
00044 }
00045 catch (L4::Runtime_error const &e)
00046 {
00047 Err().printf("Polling task failed: %s\n", e.str());
00048 }
00049 }
00050
00051 void reschedule()
00052 {
00053 // create a place holder reference pointer for the timeout queue
00054 cxx::Ref_ptr<Poll_errand> p(this);
00055
00056 _sif->add_timeout(p.release(), l4_kip_clock(l4re_kip()) + _interval);
00057 }
00058
00059 // Class can only be instantiated as a reference counting object.
00060 template< typename T, typename... Args >
00061 friend
00062 cxx::Ref_ptr<T> cxx::make_ref_obj(Args &&... args);
00063
00064 private:

```

```

00073 Poll_errand(int retries, int interval,
00074 std::function<bool()> const &poll_func,
00075 std::function<void(bool)> const &callback)
00076 : _retries(retries),
00077 _interval(interval),
00078 _poll(poll_func),
00079 _callback(callback)
00080 {}
00081
00082 int _retries;
00083 int _interval;
00084 std::function<bool()> _poll;
00085 std::function<void(bool)> _callback;
00086 };
00087
00098 class Errand
00099 : public L4::Ipc_svr::Timeout_queue::Timeout,
00100 public cxx::Ref_obj
00101 {
00102 public:
00103 void expired() final
00104 {
00105 // Recapture the reference pointer from the timeout queue.
00106 cxx::Ref_ptr<Errand> p(this, false);
00107
00108 if (_callback)
00109 {
00110 try
00111 {
00112 _callback();
00113 }
00114 catch (L4::Runtime_error const &e)
00115 {
00116 Err().printf("Asynchronous task failed: %s\n", e.str());
00117 }
00118 }
00119 }
00120
00121 void reschedule(unsigned interval = 0)
00122 {
00123 // create a placeholder reference pointer for the timeout queue
00124 cxx::Ref_ptr<Errand> p(this);
00125
00126 _sif->add_timeout(p.release(), l4_kip_clock(l4re_kip()) + interval);
00127 }
00128
00129 // Class can only be instantiated as a reference counting object.
00130 template< typename T, typename... Args >
00131 friend
00132 cxx::Ref_ptr<T> cxx::make_ref_obj(Args &&... args);
00133
00134 private:
00135 Errand(Callback const &callback) : _callback(callback) {}
00136
00137 Callback _callback;
00138 };
00139
00140 struct Loop_hooks
00141 : L4::Ipc_svr::Timeout_queue_hooks<Loop_hooks, L4Re::Util::Br_manager>,
00142 L4::Ipc_svr::Ignore_errors
00143 {
00144 l4_kernel_clock_t now() { return l4_kip_clock(l4re_kip()); }
00145 };
00146
00147 using Errand_server = L4Re::Util::Registry_server<Loop_hooks>;
00148
00154 inline void set_server_iface(L4::Ipc_svr::Server_iface *sif) { _sif = sif; }
00155
00166 inline void schedule(Callback const &callback, int interval)
00167 {
00168 cxx::make_ref_obj<Errand>(callback)->reschedule(interval);
00169 }
00170
00191 inline void poll(int retries, int interval,
00192 std::function<bool()> const &poll_func,
00193 std::function<void(bool)> const &callback)
00194 {
00195 if (poll_func())
00196 callback(true);
00197 else
00198 cxx::make_ref_obj<Poll_errand>(retries, interval, poll_func,
00199 callback)->reschedule();
00200 }
00201
00202
00203 } } // name space

```

## 17.269 gpt.h

```

00001 /*
00002 * Copyright (C) 2018, 2024 Kernkonzept GmbH.
00003 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/sys/types.h>
00010
00011 namespace Block_device {
00012 namespace Gpt {
00013
00014 struct Header
00015 {
00016 char signature[8];
00017 l4_uint32_t version;
00018 l4_uint32_t header_size;
00019 l4_uint32_t crc;
00020 l4_uint32_t _reserved;
00021 l4_uint64_t current_lba;
00022 l4_uint64_t backup_lba;
00023 l4_uint64_t first_lba;
00024 l4_uint64_t last_lba;
00025 char disk_guid[16];
00026 l4_uint64_t partition_array_lba;
00027 l4_uint32_t partition_array_size;
00028 l4_uint32_t entry_size;
00029 l4_uint32_t crc_array;
00030 } __attribute__((packed));
00031
00032 struct Entry
00033 {
00034 unsigned char type_guid[16];
00035 unsigned char partition_guid[16];
00036 l4_uint64_t first;
00037 l4_uint64_t last;
00038 l4_uint64_t flags;
00039 l4_uint16_t name[36];
00040 };
00041
00042 } // namespace
00043
00044 namespace Pc_partition_table {
00045
00046 struct Part_table {
00047 l4_uint8_t bootable;
00048 l4_uint8_t first_sector_chs[3];
00049 l4_uint8_t type;
00050 l4_uint8_t last_sector_chs[3];
00051 l4_uint32_t start_sector_lba;
00052 l4_uint32_t num_sector_lba;
00053 } __attribute__((packed));
00054
00055 } // namespace
00056 } // namespace

```

## 17.270 inout\_memory.h

```

00001 /*
00002 * Copyright (C) 2014, 2019-2020, 2024 Kernkonzept GmbH.
00003 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/re/error_helper>
00010 #include <l4/re/env>
00011 #include <l4/re/util/unique_cap>
00012 #include <l4/re/rm>
00013 #include <l4/re/dma_space>
00014 #include <l4/cxx/ref_ptr>
00015
00016 #include <l4/libblock-device/types.h>
00017
00018 namespace Block_device {
00019
00024 template <typename DEV>
00025 class Inout_memory : public cxx::Ref_obj

```

```

00026 {
00027 public:
00028 using Device_type = DEV;
00029
00030 Inout_memory() : _paddr(0) {}
00031 Inout_memory(l4_uint32_t num_sectors, Device_type *dev,
00032 L4Re::Dma_space::Direction dir)
00033 : _device(dev), _paddr(0), _dir(dir), _num_sectors(num_sectors)
00034 {
00035 auto lcap = L4Re::chkcap(L4Re::Util::make_unique_cap<L4Re::Dataspace>()),
00036 "Allocate dataspace capability for IO memory.");
00037
00038 auto *e = L4Re::Env::env();
00039 long sz = num_sectors * _device->sector_size();
00040 L4Re::chksys(e->mem_alloc()->alloc(sz, lcap.get(),
00041 L4Re::Mem_alloc::Continuous
00042 | L4Re::Mem_alloc::Pinned),
00043 "Allocate pinned memory.");
00044
00045 L4Re::chksys(e->rm()->attach(&_region, sz,
00046 L4Re::Rm::F::Search_addr | L4Re::Rm::F::RW,
00047 L4::Ipc::make_cap_rw(lcap.get(), 0,
00048 L4_PAGESHIFT),
00049 "Attach IO memory.");
00050
00051 _mem_region =
00052 cxx::make_unique<Block_device::Mem_region>(0, sz, 0, cxx::move(lcap));
00053 L4Re::chksys(_device->dma_map(_mem_region.get(), 0, _num_sectors, dir,
00054 &_paddr),
00055 "Lock memory region for DMA.");
00056 }
00057
00058 Inout_memory(Inout_memory const &) = delete;
00059 Inout_memory(Inout_memory &&) = delete;
00060
00061 Inout_memory &operator=(Inout_memory &&rhs)
00062 {
00063 if (this != &rhs)
00064 {
00065 _device = rhs._device;
00066 _mem_region = cxx::move(rhs._mem_region);
00067 _region = cxx::move(rhs._region);
00068 _paddr = rhs._paddr;
00069 _dir = rhs._dir;
00070 _num_sectors = rhs._num_sectors;
00071 rhs._paddr = 0;
00072 }
00073
00074 return *this;
00075 }
00076
00077 ~Inout_memory()
00078 {
00079 if (_paddr)
00080 unmap();
00081 }
00082
00083 void unmap()
00084 {
00085 L4Re::chksys(_device->dma_unmap(_paddr, _num_sectors, _dir));
00086 _paddr = 0;
00087 }
00088
00089 Inout_block inout_block() const
00090 {
00091 Inout_block blk;
00092
00093 blk.dma_addr = _paddr;
00094 blk.virt_addr = _region.get();
00095 blk.num_sectors = _num_sectors;
00096 blk.next.reset();
00097
00098 return blk;
00099 }
00100
00101 template <class T>
00102 T *get(unsigned offset) const
00103 { return reinterpret_cast<T *>(_region.get() + offset); }
00104
00105 private:
00106 Device_type *_device;
00107 cxx::unique_ptr<Block_device::Mem_region> _mem_region;
00108 L4Re::Rm::Unique_region<char *> _region;
00109 L4Re::Dma_space::Dma_addr _paddr;
00110 L4Re::Dma_space::Direction _dir;
00111 l4_uint32_t _num_sectors;

```

```

00113 };
00114
00115 } // name space

```

## 17.271 part\_device.h

```

00001 /*
00002 * Copyright (C) 2018-2022, 2024 Kernkonzept GmbH.
00003 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/cxx/ref_ptr>
00010
00011 #include <l4/libblock-device/device.h>
00012 #include <l4/libblock-device/partition.h>
00013
00014 #include <string>
00015 #include <locale>
00016 #include <codecvt>
00017
00018 namespace Block_device {
00019
00020 namespace Impl {
00021
00022 template <typename PART_DEV, typename BASE_DEV,
00023 bool = std::is_base_of<Device_discard_feature, BASE_DEV>::value>
00028 class Partitioned_device_discard_mixin : public BASE_DEV {};
00029
00030 template <typename PART_DEV, typename BASE_DEV>
00037 class Partitioned_device_discard_mixin<PART_DEV, BASE_DEV, true>
00038 : public BASE_DEV
00039 {
00040 public:
00041 using Base = BASE_DEV;
00042 using Part_device = PART_DEV;
00043
00044 typename Base::Discard_info discard_info() const override
00045 {
00046 return dev()->parent()->discard_info();
00047 }
00048
00049 int discard(l4_uint64_t offset, Inout_block const &blocks,
00050 Inout_callback const &cb, bool discard) override
00051 {
00052 auto sz = dev()->partition_size();
00053
00054 if (offset > sz)
00055 return -L4_EINVAL;
00056
00057 Inout_block const *cur = &blocks;
00058 while (cur)
00059 {
00060 if (cur->sector >= sz - offset)
00061 return -L4_EINVAL;
00062 if (cur->num_sectors > sz)
00063 return -L4_EINVAL;
00064 if (offset + cur->sector > sz - cur->num_sectors)
00065 return -L4_EINVAL;
00066
00067 cur = cur->next.get();
00068 }
00069
00070 auto start = offset + dev()->partition_start();
00071 Dbg::trace("partition")
00072 .printf("Starting sector on disk: 0x%llx\n", start);
00073 return dev()->parent()->discard(start, blocks, cb, discard);
00074 }
00075
00076 private:
00077 Part_device const *dev() const
00078 { return static_cast<Part_device const *>(this); }
00079 };
00080
00081 }
00082
00090 template <typename BASE_DEV = Device>
00091 class Partitioned_device
00092 : public Impl::Partitioned_device_discard_mixin<Partitioned_device<BASE_DEV>, BASE_DEV>
00093 {
00094 public:

```

```

00095 using Device_type = BASE_DEV;
00096
00097 Partitioned_device(cxx::Ref_ptr<Device_type> const &dev,
00098 unsigned partition_id, Partition_info const &pi)
00099 : _name(pi.name),
00100 _parent(dev),
00101 _start(pi.first),
00102 _size(pi.last - pi.first + 1)
00103 {
00104 if (pi.last < pi.first)
00105 L4Re::chksys(-L4_EINVAL,
00106 "Last sector of partition before first sector.");
00107
00108 if (partition_id > 999)
00109 L4Re::chksys(-L4_EINVAL,
00110 "Partition ID must be smaller than 1000.");
00111
00112 snprintf(_partition_id, sizeof(_partition_id), "%d", partition_id);
00113
00114 static_assert(sizeof(_guid) == sizeof(pi.guid), "String size mismatch");
00115 memcpy(_guid, pi.guid, sizeof(_guid));
00116 }
00117
00118 Notification_domain const *notification_domain() const override
00119 { return _parent->notification_domain(); }
00120
00121 bool is_read_only() const override
00122 { return _parent->is_read_only(); }
00123
00124 bool match_hid(cxx::String const &hid) const override
00125 {
00126 if (hid == cxx::String(_guid, 36))
00127 return true;
00128
00129 _Pragma("GCC diagnostic push");
00130 _Pragma("GCC diagnostic ignored \\"-Wdeprecated-declarations\\"");
00131 std::u16string whid =
00132 std::wstring_convert<std::codecvt_utf8_utf16<char16_t>, char16_t>{}
00133 .from_bytes(std::string(hid.start(), hid.len()));
00134 _Pragma("GCC diagnostic pop");
00135 if (whid == _name)
00136 return true;
00137
00138 // check for identifier of form: <device_name>:<partition id>
00139 char const *delim = ":";
00140 char const *pos = hid.rfind(delim);
00141
00142 if (pos == hid.end() || !_parent->match_hid(cxx::String(hid.start(), pos)))
00143 return false;
00144
00145 return cxx::String(pos + 1, hid.end()) == cxx::String(_partition_id);
00146 }
00147
00148 l4_uint64_t capacity() const override
00149 { return _size * _parent->sector_size(); }
00150
00151 l4_size_t sector_size() const override
00152 { return _parent->sector_size(); }
00153
00154 l4_size_t max_size() const override
00155 { return _parent->max_size(); }
00156
00157 unsigned max_segments() const override
00158 { return _parent->max_segments(); }
00159
00160 void reset() override
00161 {}
00162
00163 int dma_map(Block_device::Mem_region *region, l4_addr_t offset,
00164 l4_size_t num_sectors, L4Re::Dma_space::Direction dir,
00165 L4Re::Dma_space::Dma_addr *phys) override
00166 { return _parent->dma_map(region, offset, num_sectors, dir, phys); }
00167
00168 int dma_unmap(L4Re::Dma_space::Dma_addr phys, l4_size_t num_sectors,
00169 L4Re::Dma_space::Direction dir) override
00170 { return _parent->dma_unmap(phys, num_sectors, dir); }
00171
00172 int inout_data(l4_uint64_t sector, Inout_block const &blocks,
00173 Inout_callback const &cb,
00174 L4Re::Dma_space::Direction dir) override
00175 {
00176 if (sector >= _size)
00177 return -L4_EINVAL;
00178
00179 l4_uint64_t total = 0;
00180 Inout_block const *cur = &blocks;
00181 while (cur)

```

```

00182 {
00183 total += cur->num_sectors;
00184 cur = cur->next.get();
00185 }
00186
00187 if (total > _size - sector)
00188 return -L4_EINVAL;
00189
00190 Dbg::trace("partition").printf("Sector on disk: 0x%llx\n", sector + _start);
00191 return _parent->inout_data(sector + _start, blocks, cb, dir);
00192 }
00193
00194 int flush(Inout_callback const &cb) override
00195 {
00196 return _parent->flush(cb);
00197 }
00198
00199 void start_device_scan(Block_device::Errand::Callback const &callback) override
00200 { callback(); }
00201
00202 l4_uint64_t partition_size() const
00203 { return _size; }
00204
00205 l4_uint64_t partition_start() const
00206 { return _start; }
00207
00208 Device_type *parent() const
00209 { return _parent.get(); }
00210
00211
00212 private:
00213 char _guid[37];
00214 std::ul6string _name;
00215 char _partition_id[4];
00216 cxx::Ref_ptr<Device_type> _parent;
00217 l4_uint64_t _start;
00218 l4_uint64_t _size;
00219 };
00220
00221 } // name space

```

## 17.272 partition.h

```

00001 /*
00002 * Copyright (C) 2018, 2020-2024 Kernkonzept GmbH.
00003 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <cstring>
00010 #include <string>
00011 #include <cassert>
00012
00013 #include <l4/cxx/ref_ptr>
00014
00015 #include <l4/l4virtio/virtio_block.h>
00016
00017 #include <l4/libblock-device/debug.h>
00018 #include <l4/libblock-device/errand.h>
00019 #include <l4/libblock-device/inout_memory.h>
00020 #include <l4/libblock-device/gpt.h>
00021
00022 #include <l4/sys/cache.h>
00023
00024 namespace Block_device {
00025
00026 struct Partition_info
00027 {
00028 char guid[37];
00029 std::ul6string name;
00030 l4_uint64_t first;
00031 l4_uint64_t last;
00032 l4_uint64_t flags;
00033 };
00034
00035 template <typename DEV>
00036 class Partition_reader : public cxx::Ref_obj
00037 {
00038 enum
00039 {

```

```

00047 Max_partitions = 1024
00048 };
00049
00050 public:
00051 using Device_type = DEV;
00052
00053 Partition_reader(Device_type *dev)
00054 : _num_partitions(0),
00055 _dev(dev),
00056 _header(2, dev, L4Re::Dma_space::Direction::From_device)
00057 {}
00058
00059 void read(Errand::Callback const &callback)
00060 {
00061 _num_partitions = 0;
00062 _callback = callback;
00063
00064 // preparation: read the first two sectors
00065 _db = _header.inout_block();
00066 read_sectors(0, &Partition_reader::get_gpt);
00067 }
00068
00069 l4_size_t table_size() const
00070 { return _num_partitions; }
00071
00072 int get_partition(l4_size_t idx, Partition_info *inf) const
00073 {
00074 if (idx == 0 || idx > _num_partitions)
00075 return -L4_ERANGE;
00076
00077 unsigned secsz = _dev->sector_size();
00078 auto *header = _header.template get<Gpt::Header const>(secsz);
00079
00080 Gpt::Entry *e = _parray.template get<Gpt::Entry>((idx - 1) * header->entry_size);
00081
00082 if ((*((l4_uint64_t *) &e->partition_guid) == 0ULL)
00083 return -L4_ENODEV;
00084
00085 render_guid(e->partition_guid, inf->guid);
00086
00087 auto name =
00088 std::u16string((char16_t *)e->name, sizeof(e->name) / sizeof(e->name[0]));
00089 inf->name = name.substr(0, name.find((char16_t) 0));
00090
00091 inf->first = e->first;
00092 inf->last = e->last;
00093 inf->flags = e->flags;
00094
00095 auto info = Dbg::info();
00096 if (info.is_active())
00097 {
00098 info.printf("%3zu: %10lld %10lld %5gMiB [%3.7s]\n",
00099 idx, e->first, e->last,
00100 (e->last - e->first + 1.0) * secsz / (1 « 20),
00101 inf->guid);
00102
00103 char buf[37];
00104 info.printf(" : Type: %s\n", render_guid(e->type_guid, buf));
00105 }
00106
00107 auto warn = Dbg::warn();
00108 if (inf->last < inf->first)
00109 {
00110 warn.printf(
00111 "Invalid settings of %3zu. Last lba before first lba. Will ignore.\n",
00112 idx);
00113 // Errors in the GPT shall not crash any service -- just ignore the
00114 // corresponding partition.
00115 return -L4_ENODEV;
00116 }
00117
00118 return L4_EOK;
00119 }
00120
00121 private:
00122 void invoke_callback()
00123 {
00124 assert(_callback);
00125 _callback();
00126 // Reset the callback to drop potential transitive self-references
00127 _callback = nullptr;
00128 }
00129
00130 void get_gpt(int error, l4_size_t)
00131 {
00132 _header.unmap();
00133 }

```



```

00134 if (error < 0)
00135 {
00136 // can't read from device, we are done
00137 invoke_callback();
00138 return;
00139 }
00140
00141 // prepare reading of the table from disk
00142 unsigned secsz = _dev->sector_size();
00143 auto *header = _header.template get<Gpt::Header const>(secsz);
00144
00145 auto info = Dbg::info();
00146 auto trace = Dbg::trace();
00147
00148 if (strncmp(header->signature, "EFI PART", 8) != 0)
00149 {
00150 info.printf("No GUID partition header found.\n");
00151 invoke_callback();
00152 return;
00153 }
00154
00155 // XXX check CRC32 of header
00156
00157 info.printf("GUID partition header found with up to %d partitions.\n",
00158 header->partition_array_size);
00159 char buf[37];
00160 info.printf("GUID: %s\n", render_guid(header->disk_guid, buf));
00161 trace.printf("Header positions: %llx (Backup: %llx)\n",
00162 header->current_lba, header->backup_lba);
00163 trace.printf("First + last: %llx and %llx\n",
00164 header->first_lba, header->last_lba);
00165 trace.printf("Partition table at %llx\n",
00166 header->partition_array_lba);
00167 trace.printf("Size of a partition entry: %d\n",
00168 header->entry_size);
00169
00170 info.printf("GUID partition header found with %d partitions.\n",
00171 header->partition_array_size);
00172
00173 _num_partitions = cxx::min<l4_uint32_t>(header->partition_array_size,
00174 Max_partitions);
00175
00176 l4_size_t arraysz = _num_partitions * header->entry_size;
00177 l4_size_t numsec = (arraysz - 1 + secsz) / secsz;
00178
00179 _parray = Inout_memory<Device_type>(numsec, _dev, L4Re::Dma_space::Direction::From_device);
00180 trace.printf("Reading GPT table @ 0x%p\n", _parray.template get<void>(0));
00181
00182 _db = _parray.inout_block();
00183 read_sectors(header->partition_array_lba, &Partition_reader::done_gpt);
00184 }
00185
00186 void done_gpt(int error, l4_size_t)
00187 {
00188 _parray.unmap();
00189
00190 // XXX check CRC32 of table
00191
00192 if (error < 0)
00193 _num_partitions = 0;
00194
00195 invoke_callback();
00196 }
00197
00198 void read_sectors(l4_uint64_t sector,
00199 void (Partition_reader::*func)(int, l4_size_t))
00200 {
00201 using namespace std::placeholders;
00202 auto next = std::bind(func, this, _1, _2);
00203
00204 l4_addr_t vstart = (l4_addr_t)_db.virt_addr;
00205 l4_addr_t vend = vstart + _db.num_sectors * _dev->sector_size();
00206 l4_cache_inv_data(vstart, vend);
00207
00208 Errand::poll(10, 10000,
00209 [=]()
00210 {
00211 int ret = _dev->inout_data(
00212 sector, _db,
00213 [next, vstart, vend](int error, l4_size_t size)
00214 {
00215 l4_cache_inv_data(vstart, vend);
00216 next(error, size);
00217 },
00218 L4Re::Dma_space::Direction::From_device);
00219 if (ret < 0 && ret != -L4_EBUSY)
00220

```

```

00221 invoke_callback();
00222 return ret != -L4_EBUSY;
00223 },
00224 [=](bool ret) { if (!ret) invoke_callback(); }
00225);
00226 }
00227
00228 static char const *render_guid(void const *guid_p, char buf[])
00229 {
00230 auto *p = static_cast<unsigned char const *>(guid_p);
00231 snprintf(buf, 37,
00232 "%02X%02X%02X%02X-%02X%02X-%02X%02X-%02X%02X%02X%02X%02X",
00233 p[3], p[2], p[1], p[0], p[5], p[4], p[7], p[6],
00234 p[8], p[9], p[10], p[11], p[12], p[13], p[14], p[15]);
00235
00236 return buf;
00237 }
00238
00239 l4_size_t _num_partitions;
00240 Inout_block _db;
00241 Device_type *_dev;
00242 Inout_memory<Device_type> _header;
00243 Inout_memory<Device_type> _parray;
00244 Errand::Callback _callback;
00245 };
00246
00247
00248
00249 }

```

## 17.273 scheduler.h

```

00001 /*
00002 * Copyright (C) 2024 Kernkonzept GmbH.
00003 * Author(s): Jakub Jermar <jakub.jermar@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #pragma once
00009
00010 #include <vector>
00011
00012 #include <l4/cxx/unique_ptr>
00013 #include <l4/re/error_helper>
00014 #include <l4/sys/cxx/ipc_epiface>
00015
00016 #include <l4/libblock-device/debug.h>
00017 #include <l4/libblock-device/virtio_client.h>
00018
00019 namespace Block_device {
00020
00034 template <typename DEV>
00035 class Scheduler_base
00036 {
00037 protected:
00038 using Device_type = DEV;
00039 using Client_type = Virtio_client<Device_type>;
00040
00041 private:
00042 class Irq_object : public L4::Irqep_t<Irq_object>
00043 {
00044 public:
00045 Irq_object(Scheduler_base *parent) : _parent(parent) {}
00046
00047 void handle_irq() { _parent->schedule(); }
00048
00049 private:
00050 Scheduler_base *_parent;
00051 };
00052 Irq_object _irq_handler;
00053
00054 struct Context
00055 {
00056 cxx::unique_ptr<Pending_request> pending;
00057 Client_type *client;
00058
00059 bool device_busy;
00060 l4_size_t cost;
00061
00062 Context(Client_type *client) : client(client), device_busy(false), cost(0)
00063 {}
00064

```

```

00065 bool same_notification_domain(Client_type const *c) const
00066 { return c->notification_domain() == client->notification_domain(); }
00067 };
00068
00069 using Queue_type = std::vector<cxx::unique_ptr<Context>>;
00070 using Iterator_type = typename Queue_type::const_iterator;
00071
00072 public:
00073 Scheduler_base(L4::Registry_iface *registry)
00074 : _irq_handler(this), _registry(registry), _next(_clients.cend())
00075 {
00076 L4Re::chkcap(registry->register_irq_obj(&_irq_handler),
00077 "Registering device notify IRQ object.");
00078 }
00079
00080 virtual ~Scheduler_base()
00081 {
00082 // We need to explicitly delete the IRQ object created in register_irq_obj()
00083 // ourselves. Even though unregister_obj() will unmap the cap, it might stay
00084 // alive because it was given out to the client. Hence it might be
00085 // dispatched even after unregister_obj() returned!
00086 L4::Cap<L4::Task>(L4Re::This_task)
00087 ->unmap(_irq_handler.obj_cap().fpage(),
00088 L4_FP_ALL_SPACES | L4_FP_DELETE_OBJ);
00089 _registry->unregister_obj(&_irq_handler);
00090 }
00091
00095 virtual l4_size_t get_weight(Client_type const *) = 0;
00096
00100 virtual l4_size_t get_cost(Pending_request const &) = 0;
00101
00102 void add_client(Client_type *client)
00103 {
00104 Dbg::trace().printf("Adding client %p to request scheduler.\n", client);
00105
00106 // make sure the client uses the request scheduler's device_notify_irq
00107 client->set_device_notify_irq(
00108 L4::cap_cast<L4::Irq>(_irq_handler.obj_cap()));
00109
00110 client->set_client_invalidate_cb([this, client](bool fail_pending) {
00111 client_invalidate(client, fail_pending);
00112 });
00113
00114 client->set_client_idle_cb([this, client]() { client_idle(client); });
00115
00116 _clients.push_back(cxx::make_unique<Context>(client));
00117 _next = _clients.cend();
00118 }
00119
00120 void remove_client(Client_type *client)
00121 {
00122 Dbg::trace().printf("Removing client %p from request scheduler.\n", client);
00123 _clients.erase(std::remove_if(_clients.begin(), _clients.end(),
00124 [client](cxx::unique_ptr<Context> &c) {
00125 return c->client == client;
00126 }));
00127 _next = _clients.cend();
00128 }
00129
00130 Queue_type const &clients()
00131 { return _clients; }
00132
00133 private:
00134 void client_invalidate(Client_type *client, bool fail_pending)
00135 {
00136 for (auto &c : _clients)
00137 if (c->client == client)
00138 {
00139 c->device_busy = false;
00140 c->cost = 0;
00141 if (c->pending)
00142 {
00143 if (fail_pending)
00144 c->pending->fail_request();
00145 c->pending.reset();
00146 }
00147 }
00148 }
00149
00150 void client_idle(Client_type *client)
00151 {
00152 bool resched = false;
00153 for (auto &c : _clients)
00154 if (c->device_busy && c->same_notification_domain(client))
00155 {
00156 c->device_busy = false;
00157 resched = true;
00158 }
00159 }

```

```

00160 }
00161
00162 if (resched)
00163 {
00164 // By triggering the scheduler asynchronously we make synchronous
00165 // request processing in the device implementation possible. In
00166 // any case we need to be careful not to start scheduling the
00167 // pending request which is being currently handled.
00168 L4::cap_cast<L4::Irq>(this->_irq_handler.obj_cap())->trigger();
00169 }
00170 }
00171
00181 bool handle_pending(Context *c)
00182 {
00183 auto cost = get_cost(*(c->pending));
00184
00185 if (c->cost + cost > get_weight(c->client))
00186 {
00187 Dbg::trace().printf("Preempting client %p (cost=%zu+zu, weight=%zu)\n",
00188 c->client, c->cost, cost, get_weight(c->client));
00189
00190 // Charge client's entire weight to force schedule() to give another
00191 // client a chance.
00192 c->cost = get_weight(c->client);
00193 return true;
00194 }
00195
00196 // Keep the pending request in its place while handling the request.
00197 // This helps to make sure that the scheduler will not try to schedule
00198 // new requests while handling the pending one.
00199 int ret = c->pending->handle_request();
00200 if (ret == -L4_EBUSY)
00201 {
00202 c->device_busy = true;
00203 return false;
00204 }
00205
00206 c->cost += cost;
00207
00208 if (ret < 0)
00209 c->pending.reset();
00210 else
00211 c->pending.release();
00212 return true;
00213 }
00214
00231 bool schedule_client(Context *c)
00232 {
00233 if (c->pending)
00234 {
00235 if (c->device_busy)
00236 {
00237 Dbg::trace().printf(
00238 "Skipping pending request of client %p (busy).\n", c->client);
00239 return false;
00240 }
00241
00242 Dbg::trace().printf("Handling pending request of client %p.\n",
00243 c->client);
00244 // The client has a pending request, we need to handle it first
00245 // before new requests can be processed. If we manage to handle it,
00246 // we need to check again in the next round for new requests.
00247 return handle_pending(c);
00248 }
00249
00250 if (c->client->check_for_new_requests())
00251 {
00252 auto req = c->client->get_request();
00253 if (req)
00254 {
00255 Dbg::trace().printf("Scheduling request from client %p.\n",
00256 c->client);
00257 c->pending = c->client->start_request(cxx::move(req));
00258 if (c->pending)
00259 {
00260 // We processed one new request by turning it into a pending
00261 // one and possibly sending it to the device (or not). We
00262 // need to recheck only if the request was successfully sent
00263 // to the device.
00264 return handle_pending(c);
00265 }
00266 // We processed one new request immediately (e.g. failed
00267 // sanity check, runtime error or client state).
00268 return true;
00269 }
00270 }
00271 }

```

```

00272 return false;
00273 }
00274
00287 void schedule()
00288 {
00289 if (_clients.empty())
00290 return;
00291
00292 if (_next == _clients.cend())
00293 _next = _clients.cbegin();
00294
00295 (*_next)->cost = 0;
00296
00297 Iterator_type start(_next);
00298 bool recheck = false;
00299 for (;;)
00300 {
00301 bool progress = schedule_client(_next->get());
00302 // Move onto the next client only after the current client has depleted
00303 // its chances to process its queue or if it didn't make any forward
00304 // progress
00305 if (!progress || ((*_next)->cost >= get_weight((*_next)->client)))
00306 {
00307 ++_next;
00308 if (_next == _clients.cend())
00309 _next = _clients.cbegin();
00310 (*_next)->cost = 0;
00311 }
00312 recheck |= progress;
00313 if (_next == start)
00314 {
00315 if (!recheck)
00316 {
00317 // already processed all clients and requests, start with
00318 // the next client next time
00319 ++_next;
00320 break;
00321 }
00322 else
00323 recheck = false;
00324 }
00325 }
00326 }
00327
00328 L4::Registry_iface *_registry;
00329 Queue_type _clients;
00330 Iterator_type _next;
00331 };
00332
00339 template <typename DEV>
00340 struct Rr_scheduler : Scheduler_base<DEV>
00341 {
00342 using Scheduler_base<DEV>::Scheduler_base;
00343
00344 l4_size_t
00345 get_weight(typename Scheduler_base<DEV>::Client_type const *) override
00346 { return 1; }
00347
00348 l4_size_t get_cost(Pending_request const &) override
00349 { return 1; }
00350 };
00351
00352
00353 } // name space

```

## 17.274 l4/sys/scheduler.h File Reference

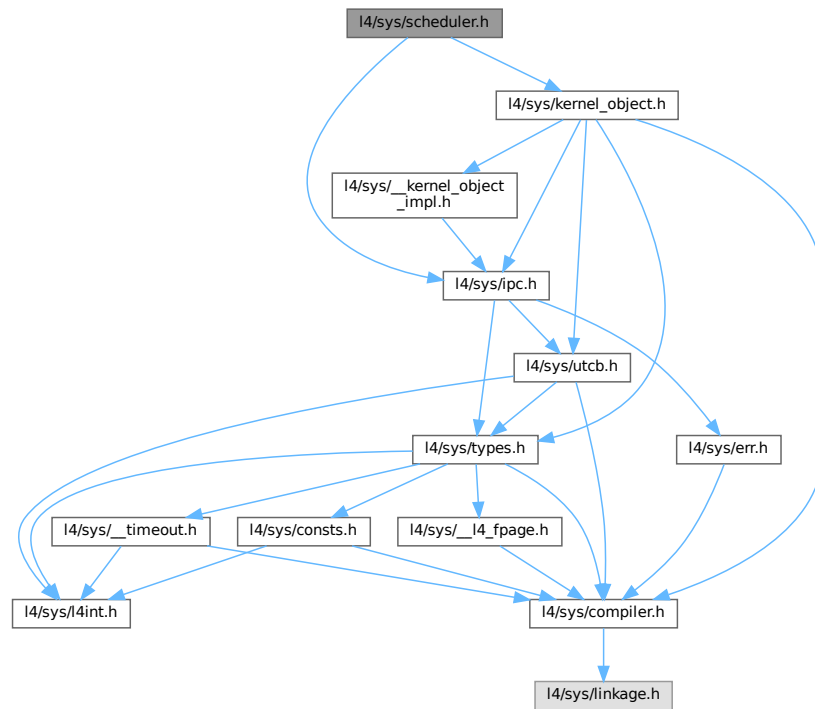
Scheduler object functions.

```

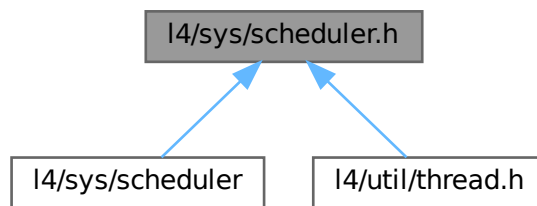
#include <l4/sys/kernel_object.h>
#include <l4/sys/ipc.h>

```

Include dependency graph for scheduler.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [l4\\_sched\\_cpu\\_set\\_t](#)  
*CPU sets.*
- struct [l4\\_sched\\_param\\_t](#)  
*Scheduler parameter set.*

## Typedefs

- typedef struct l4\_sched\_cpu\_set\_t **l4\_sched\_cpu\_set\_t**  
*CPU sets.*
- typedef struct l4\_sched\_param\_t **l4\_sched\_param\_t**  
*Scheduler parameter set.*

## Enumerations

- enum **L4\_scheduler\_classes** { **L4\_SCHEDULER\_CLASS\_FIXED\_PRIO** = 1UL << 1, **L4\_SCHEDULER\_CLASS\_WFQ** = 1UL << 2 }  
*Supported scheduler classes.*
- enum **L4\_scheduler\_ops** { **L4\_SCHEDULER\_INFO\_OP** = 0UL, **L4\_SCHEDULER\_RUN\_THREAD\_OP** = 1UL, **L4\_SCHEDULER\_IDLE\_TIME\_OP** = 2UL }  
*Operations on the Scheduler object.*

## Functions

- **l4\_sched\_cpu\_set\_t l4\_sched\_cpu\_set** (l4\_umword\_t offset, unsigned char granularity, l4\_umword\_t map=1) **L4\_NOTHROW**
- **l4\_msgtag\_t l4\_scheduler\_info** (l4\_cap\_idx\_t scheduler, l4\_umword\_t \*cpu\_max, l4\_sched\_cpu\_set\_t \*cpus) **L4\_NOTHROW**)  
*Get scheduler information.*
- **l4\_msgtag\_t l4\_scheduler\_info\_with\_classes** (l4\_cap\_idx\_t scheduler, l4\_umword\_t \*cpu\_max, l4\_sched\_cpu\_set\_t \*cpus, l4\_umword\_t \*sched\_classes) **L4\_NOTHROW**)  
*Get scheduler information.*
- **l4\_sched\_param\_t l4\_sched\_param** (unsigned prio, l4\_umword\_t quantum=0) **L4\_NOTHROW**  
*Construct scheduler parameter.*
- **l4\_msgtag\_t l4\_scheduler\_run\_thread** (l4\_cap\_idx\_t scheduler, l4\_cap\_idx\_t thread, l4\_sched\_param\_t const \*sp) **L4\_NOTHROW**)  
*Run a thread on a Scheduler.*
- **l4\_msgtag\_t l4\_scheduler\_idle\_time** (l4\_cap\_idx\_t scheduler, l4\_sched\_cpu\_set\_t const \*cpus, l4\_kernel\_clock\_t \*us) **L4\_NOTHROW**)  
*Query the idle time (in  $\mu$ s) of a CPU.*
- int **l4\_scheduler\_is\_online** (l4\_cap\_idx\_t scheduler, l4\_umword\_t cpu) **L4\_NOTHROW**  
*Query if a CPU is online.*

### 17.274.1 Detailed Description

Scheduler object functions.

Definition in file [scheduler.h](#).

## 17.275 scheduler.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00014 #include <l4/sys/kernel_object.h>
00015 #include <l4/sys/ipc.h>
00016
00040
00046 enum L4_scheduler_classes
00047 {
00049 L4_SCHEDULER_CLASS_FIXED_PRIO = 1UL < 1,
00051 L4_SCHEDULER_CLASS_WFQ = 1UL < 2,
00052 };
00053
00058 typedef struct l4_sched_cpu_set_t
00059 {
00072 l4_umword_t gran_offset;
00073
00077 l4_umword_t map;
00078
00079 #ifdef __cplusplus
00081 unsigned char granularity() const { return gran_offset >> 24; }
00083 unsigned offset() const { return gran_offset & 0x00ffffff; }
00090 void set(unsigned char granularity, unsigned offset)
00091 {
00092 gran_offset = (static_cast<l4_umword_t>(granularity) << 24)
00093 | (offset & 0x00ffffff);
00094 }
00095 #endif
00096 } l4_sched_cpu_set_t;
00097
00108 L4_INLINE l4_sched_cpu_set_t
00109 l4_sched_cpu_set(l4_umword_t offset, unsigned char granularity,
00110 l4_umword_t map L4_DEFAULT_PARAM(1)) L4_NOTHROW;
00111
00128 L4_INLINE l4_msgtag_t
00129 l4_scheduler_info(l4_cap_idx_t scheduler, l4_umword_t *cpu_max,
00130 l4_sched_cpu_set_t *cpus)
00131 L4_NOTHROW __attribute__((nonnull (3)));
00132
00154 L4_INLINE l4_msgtag_t
00155 l4_scheduler_info_with_classes(l4_cap_idx_t scheduler, l4_umword_t *cpu_max,
00156 l4_sched_cpu_set_t *cpus,
00157 l4_umword_t *sched_classes)
00158 L4_NOTHROW __attribute__((nonnull (3)));
00159
00163 L4_INLINE l4_msgtag_t
00164 l4_scheduler_info_u(l4_cap_idx_t scheduler, l4_umword_t *cpu_max,
00165 l4_sched_cpu_set_t *cpus, l4_umword_t *sched_classes,
00166 l4_utcb_t *utcb) L4_NOTHROW __attribute__((nonnull (3, 5)));
00167
00168
00173 typedef struct l4_sched_param_t
00174 {
00176 l4_sched_cpu_set_t affinity;
00182 l4_umword_t prio;
00184 l4_umword_t quantum;
00185 } l4_sched_param_t;
00186
00195 L4_INLINE l4_sched_param_t
00196 l4_sched_param(unsigned prio,
00197 l4_umword_t quantum L4_DEFAULT_PARAM(0)) L4_NOTHROW;
00198
00206 L4_INLINE l4_msgtag_t
00207 l4_scheduler_run_thread(l4_cap_idx_t scheduler,
00208 l4_cap_idx_t thread, l4_sched_param_t const *sp)
00209 L4_NOTHROW __attribute__((nonnull));
00210
00214 L4_INLINE l4_msgtag_t
00215 l4_scheduler_run_thread_u(l4_cap_idx_t scheduler, l4_cap_idx_t thread,
00216 l4_sched_param_t const *sp, l4_utcb_t *utcb)
00217 L4_NOTHROW __attribute__((nonnull));
00218
00226 L4_INLINE l4_msgtag_t
00227 l4_scheduler_idle_time(l4_cap_idx_t scheduler, l4_sched_cpu_set_t const *cpus,
00228 l4_kernel_clock_t *us)

```



```

00229 L4_NOTHROW __attribute__((nonnull));
00230
00234 L4_INLINE l4_msgtag_t
00235 l4_scheduler_idle_time_u(l4_cap_idx_t scheduler, l4_sched_cpu_set_t const *cpus,
00236 l4_kernel_clock_t *us, l4_utcb_t *utcb)
00237 L4_NOTHROW __attribute__((nonnull));
00238
00239
00240
00251 L4_INLINE int
00252 l4_scheduler_is_online(l4_cap_idx_t scheduler, l4_umword_t cpu) L4_NOTHROW;
00253
00257 L4_INLINE int
00258 l4_scheduler_is_online_u(l4_cap_idx_t scheduler, l4_umword_t cpu,
00259 l4_utcb_t *utcb) L4_NOTHROW __attribute__((nonnull));
00260
00261
00262
00269 enum l4_scheduler_ops
00270 {
00271 L4_SCHEDULER_INFO_OP = 0UL,
00272 L4_SCHEDULER_RUN_THREAD_OP = 1UL,
00273 L4_SCHEDULER_IDLE_TIME_OP = 2UL,
00274 };
00275
00276 /***** Implementations *****/
00277
00278 L4_INLINE l4_sched_cpu_set_t
00279 l4_sched_cpu_set(l4_umword_t offset, unsigned char granularity,
00280 l4_umword_t map) L4_NOTHROW
00281 {
00282 l4_sched_cpu_set_t cs;
00283 cs.gran_offset = ((l4_umword_t)granularity < 24) | (offset & 0x00ffffff);
00284 cs.map = map;
00285 return cs;
00286 }
00287
00288 L4_INLINE l4_sched_param_t
00289 l4_sched_param(unsigned prio, l4_umword_t quantum) L4_NOTHROW
00290 {
00291 l4_sched_param_t sp;
00292 sp.prio = prio;
00293 sp.quantum = quantum;
00294 sp.affinity = l4_sched_cpu_set(0, ~0, 1);
00295 return sp;
00296 }
00297
00298
00299 L4_INLINE l4_msgtag_t
00300 l4_scheduler_info_u(l4_cap_idx_t scheduler, l4_umword_t *cpu_max,
00301 l4_sched_cpu_set_t *cpus, l4_umword_t *sched_classes,
00302 l4_utcb_t *utcb) L4_NOTHROW
00303 {
00304 l4_msg_regs_t *m = l4_utcb_mr_u(utcb);
00305 l4_msgtag_t res;
00306
00307 m->mr[0] = L4_SCHEDULER_INFO_OP;
00308 m->mr[1] = cpus->gran_offset;
00309
00310 res = l4_ipc_call(scheduler, utcb, l4_msgtag(L4_PROTO_SCHEDULER, 2, 0, 0), L4_IPC_NEVER);
00311
00312 if (l4_msgtag_has_error(res))
00313 return res;
00314
00315 cpus->map = m->mr[0];
00316
00317 if (cpu_max)
00318 *cpu_max = m->mr[1];
00319
00320 if (sched_classes)
00321 *sched_classes = m->mr[2];
00322
00323 return res;
00324 }
00325
00326 L4_INLINE l4_msgtag_t
00327 l4_scheduler_run_thread_u(l4_cap_idx_t scheduler, l4_cap_idx_t thread,
00328 l4_sched_param_t const *sp, l4_utcb_t *utcb) L4_NOTHROW
00329 {
00330 l4_msg_regs_t *m = l4_utcb_mr_u(utcb);
00331 m->mr[0] = L4_SCHEDULER_RUN_THREAD_OP;
00332 m->mr[1] = sp->affinity.gran_offset;
00333 m->mr[2] = sp->affinity.map;
00334 m->mr[3] = sp->prio;
00335 m->mr[4] = sp->quantum;
00336 m->mr[5] = l4_map_obj_control(0, 0);
00337 m->mr[6] = l4_obj_fpage(thread, 0, L4_CAP_FPAGE_RWS).raw;

```

```

00338
00339 return l4_ipc_call(scheduler, utcb, l4_msgtag(L4_PROTO_SCHEDULER, 5, 1, 0), L4_IPC_NEVER);
00340 }
00341
00342 L4_INLINE l4_msgtag_t
00343 l4_scheduler_idle_time_u(l4_cap_idx_t scheduler, l4_sched_cpu_set_t const *cpus,
00344 l4_kernel_clock_t *us, l4_utcb_t *utcb) L4_NOTHROW
00345 {
00346 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00347 l4_msgtag_t res;
00348
00349 v->mr[0] = L4_SCHEDULER_IDLE_TIME_OP;
00350 v->mr[1] = cpus->gran_offset;
00351 v->mr[2] = cpus->map;
00352
00353 res = l4_ipc_call(scheduler, utcb,
00354 l4_msgtag(L4_PROTO_SCHEDULER, 3, 0, 0), L4_IPC_NEVER);
00355
00356 if (l4_msgtag_has_error(res))
00357 return res;
00358
00359 *us = v->mr64[l4_utcb_mr64_idx(0)];
00360
00361 return res;
00362 }
00363
00364
00365 L4_INLINE int
00366 l4_scheduler_is_online_u(l4_cap_idx_t scheduler, l4_umword_t cpu,
00367 l4_utcb_t *utcb) L4_NOTHROW
00368 {
00369 l4_sched_cpu_set_t s;
00370 l4_msgtag_t r;
00371 s.gran_offset = cpu;
00372 r = l4_scheduler_info_u(scheduler, NULL, &s, NULL, utcb);
00373 if (l4_msgtag_has_error(r) || l4_msgtag_label(r) < 0)
00374 return 0;
00375
00376 return s.map & 1;
00377 }
00378
00379
00380 L4_INLINE l4_msgtag_t
00381 l4_scheduler_info(l4_cap_idx_t scheduler, l4_umword_t *cpu_max,
00382 l4_sched_cpu_set_t *cpus) L4_NOTHROW
00383 {
00384 return l4_scheduler_info_u(scheduler, cpu_max, cpus, NULL, l4_utcb());
00385 }
00386
00387 L4_INLINE l4_msgtag_t
00388 l4_scheduler_info_with_classes(l4_cap_idx_t scheduler, l4_umword_t *cpu_max,
00389 l4_sched_cpu_set_t *cpus,
00390 l4_umword_t *sched_classes) L4_NOTHROW
00391 {
00392 return l4_scheduler_info_u(scheduler, cpu_max, cpus, sched_classes, l4_utcb());
00393 }
00394
00395 L4_INLINE l4_msgtag_t
00396 l4_scheduler_run_thread(l4_cap_idx_t scheduler,
00397 l4_cap_idx_t thread, l4_sched_param_t const *sp) L4_NOTHROW
00398 {
00399 return l4_scheduler_run_thread_u(scheduler, thread, sp, l4_utcb());
00400 }
00401
00402 L4_INLINE l4_msgtag_t
00403 l4_scheduler_idle_time(l4_cap_idx_t scheduler, l4_sched_cpu_set_t const *cpus,
00404 l4_kernel_clock_t *us) L4_NOTHROW
00405 {
00406 return l4_scheduler_idle_time_u(scheduler, cpus, us, l4_utcb());
00407 }
00408
00409 L4_INLINE int
00410 l4_scheduler_is_online(l4_cap_idx_t scheduler, l4_umword_t cpu) L4_NOTHROW
00411 {
00412 return l4_scheduler_is_online_u(scheduler, cpu, l4_utcb());
00413 }

```

## 17.276 types.h

```

00001 /*
00002 * (c) 2008-2010 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00003 * economic rights: Technische Universität Dresden (Germany)
00004 * License: see LICENSE.spdx (in this directory or the directories above)

```

```

00005 */
00006 #pragma once
00007
00008 #include <l4/vbus/vbus_types.h>
00009
00014 enum l4io_iomem_flags_t
00015 {
00016 L4IO_MEM_NONCACHED = 0,
00017 L4IO_MEM_CACHED = 1,
00018 L4IO_MEM_USE_MTRR = 2,
00019 L4IO_MEM_ATTR_MASK = 0xf,
00020
00021 // combinations
00022 L4IO_MEM_WRITE_COMBINED = L4IO_MEM_USE_MTRR | L4IO_MEM_CACHED,
00023
00024
00027 L4IO_MEM_USE_RESERVED_AREA = 0x40 « 8,
00029 L4IO_MEM_EAGER_MAP = 0x80 « 8,
00030 };
00031
00036 enum l4io_device_types_t {
00037 L4IO_DEVICE_INVALID = 0,
00038 L4IO_DEVICE_PCI,
00039 L4IO_DEVICE_USB,
00040 L4IO_DEVICE_OTHER,
00041 L4IO_DEVICE_ANY = ~0
00042 };
00043
00048 enum l4io_resource_types_t {
00049 L4IO_RESOURCE_INVALID = L4VBUS_RESOURCE_INVALID,
00050 L4IO_RESOURCE_IRQ = L4VBUS_RESOURCE_IRQ,
00051 L4IO_RESOURCE_MEM = L4VBUS_RESOURCE_MEM,
00052 L4IO_RESOURCE_PORT = L4VBUS_RESOURCE_PORT,
00053 L4IO_RESOURCE_ANY = ~0
00054 };
00055
00056
00057 typedef l4vbus_device_handle_t l4io_device_handle_t;
00058 typedef unsigned l4io_resource_handle_t;
00059
00067 typedef l4vbus_resource_t l4io_resource_t;
00068
00072 typedef l4vbus_device_t l4io_device_t;

```

## 17.277 types.h

```

00001 /*
00002 * Copyright (C) 2018-2019, 2023-2024 Kernkonzept GmbH.
00003 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <functional>
00010
00011 #include <l4/cxx/unique_ptr>
00012 #include <l4/re/dma_space>
00013 #include <l4/l4virtio/server/l4virtio>
00014
00015 namespace Block_device {
00016
00018 enum Inout_flags
00019 {
00020 Inout_f_wb = 1,
00021 Inout_f_unmap = 2,
00022 };
00023
00024 enum Shutdown_type
00025 {
00027 Running = 0,
00029 // client had crashed.
00030 Client_gone,
00032 Client_shutdown,
00034 System_shutdown,
00036 System_suspend
00037 };
00038
00043 struct Dma_region_info
00044 {
00045 virtual ~Dma_region_info() = default;
00046 };
00047

```

```

00052 struct Mem_region_info
00053 {
00054 cxx::unique_ptr<Dma_region_info> dma_info;
00055 };
00056
00057 using Mem_region =
00058 L4virtio::Svr::Driver_mem_region_t<Mem_region_info>;
00059
00066 struct Inout_block
00067 {
00068 L4Re::Dma_space::Dma_addr dma_addr = 0;
00069 void *virt_addr = nullptr;
00071 l4_uint64_t sector = 0;
00072 l4_uint32_t num_sectors = 0;
00074 l4_uint32_t flags = 0;
00075 cxx::unique_ptr<Inout_block> next;
00076 };
00077
00078 typedef std::function<void(int, l4_size_t)> Inout_callback;
00079
00080 } // name space

```

## 17.278 l4/sys/types.h File Reference

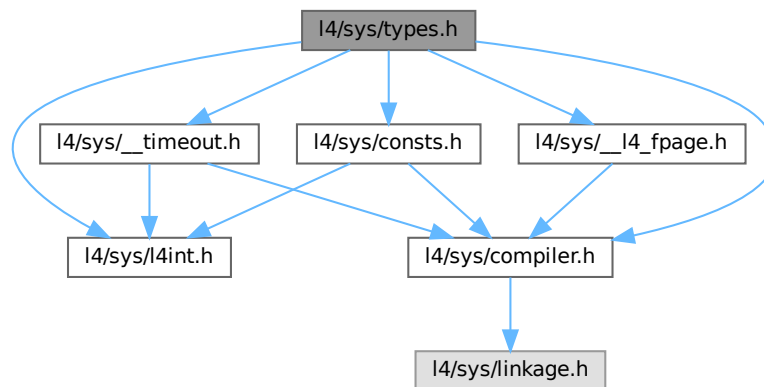
Common L4 ABI Data Types.

```

#include <l4/sys/l4int.h>
#include <l4/sys/compiler.h>
#include <l4/sys/consts.h>
#include <l4/sys/__l4_fpage.h>
#include <l4/sys/__timeout.h>

```

Include dependency graph for types.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [l4\\_msgtag\\_t](#)  
*Message tag data structure.*

## Typedefs

- typedef struct l4\_msgtag\_t [l4\\_msgtag\\_t](#)  
*Message tag data structure.*
- typedef unsigned long [l4\\_cap\\_idx\\_t](#)  
*Capability selector type.*

## Enumerations

- enum [L4\\_msgtag\\_protocol](#) {  
[L4\\_PROTO\\_NONE](#) = 0 , [L4\\_PROTO\\_ALLOW\\_SYSCALL](#) = 1 , [L4\\_PROTO\\_PF\\_EXCEPTION](#) = 1 ,  
[L4\\_PROTO\\_IRQ](#) = -1L ,  
[L4\\_PROTO\\_PAGE\\_FAULT](#) = -2L , [L4\\_PROTO\\_EXCEPTION](#) = -5L , [L4\\_PROTO\\_SIGMA0](#) = -6L ,  
[L4\\_PROTO\\_IO\\_PAGE\\_FAULT](#) = -8L ,  
[L4\\_PROTO\\_THREAD\\_GROUP](#) = -9L , [L4\\_PROTO\\_KOBJECT](#) = -10L , [L4\\_PROTO\\_TASK](#) = -11L ,  
[L4\\_PROTO\\_THREAD](#) = -12L ,  
[L4\\_PROTO\\_LOG](#) = -13L , [L4\\_PROTO\\_SCHEDULER](#) = -14L , [L4\\_PROTO\\_FACTORY](#) = -15L ,  
[L4\\_PROTO\\_VM](#) = -16L ,  
[L4\\_PROTO\\_DMA\\_SPACE](#) = -17L , [L4\\_PROTO\\_IRQ\\_SENDER](#) = -18L , [L4\\_PROTO\\_SEMAPHORE](#) = -20L ,  
[L4\\_PROTO\\_META](#) = -21L ,  
[L4\\_PROTO\\_IOMMU](#) = -22L , [L4\\_PROTO\\_DEBUGGER](#) = -23L , [L4\\_PROTO\\_SMCCC](#) = -24L ,  
[L4\\_PROTO\\_VCPU\\_CONTEXT](#) = -25L }  
*Message tag for IPC operations.*
- enum [L4\\_msgtag\\_flags](#) { [L4\\_MSGTAG\\_ERROR](#) , [L4\\_MSGTAG\\_TRANSFER\\_FPU](#) , [L4\\_MSGTAG\\_SCHEDULE](#)  
, [L4\\_MSGTAG\\_PROPAGATE](#) = 0x4000 , [L4\\_MSGTAG\\_FLAGS](#) }  
*Flags for message tags.*

## Functions

- [l4\\_msgtag\\_t l4\\_msgtag](#) (long label, unsigned words, unsigned items, unsigned flags) [L4\\_NOTHROW](#)  
*Create a message tag from the specified values.*
- long [l4\\_msgtag\\_label](#) ([l4\\_msgtag\\_t](#)) [L4\\_NOTHROW](#)  
*Get the protocol of tag.*
- unsigned [l4\\_msgtag\\_words](#) ([l4\\_msgtag\\_t](#)) [L4\\_NOTHROW](#)  
*Get the number of untyped words.*
- unsigned [l4\\_msgtag\\_items](#) ([l4\\_msgtag\\_t](#)) [L4\\_NOTHROW](#)  
*Get the number of typed items.*
- unsigned [l4\\_msgtag\\_flags](#) ([l4\\_msgtag\\_t](#)) [L4\\_NOTHROW](#)  
*Get the flags.*
- unsigned [l4\\_msgtag\\_has\\_error](#) ([l4\\_msgtag\\_t](#)) [L4\\_NOTHROW](#)  
*Test for error indicator flag.*
- unsigned [l4\\_msgtag\\_is\\_page\\_fault](#) ([l4\\_msgtag\\_t](#)) [L4\\_NOTHROW](#)  
*Test for page-fault protocol.*
- unsigned [l4\\_msgtag\\_is\\_exception](#) ([l4\\_msgtag\\_t](#)) [L4\\_NOTHROW](#)  
*Test for exception protocol.*
- unsigned [l4\\_msgtag\\_is\\_sigma0](#) ([l4\\_msgtag\\_t](#)) [L4\\_NOTHROW](#)

*Test for sigma0 protocol.*

- unsigned [l4\\_msgtag\\_is\\_io\\_page\\_fault](#) ([l4\\_msgtag\\_t](#) t) [L4\\_NOTHROW](#)

*Test for IO-page-fault protocol.*

- unsigned [l4\\_is\\_invalid\\_cap](#) ([l4\\_cap\\_idx\\_t](#) c) [L4\\_NOTHROW](#)

*Test if a capability selector is the invalid capability.*

- unsigned [l4\\_is\\_valid\\_cap](#) ([l4\\_cap\\_idx\\_t](#) c) [L4\\_NOTHROW](#)

*Test if a capability selector is a valid selector.*

- unsigned [l4\\_capability\\_equal](#) ([l4\\_cap\\_idx\\_t](#) c1, [l4\\_cap\\_idx\\_t](#) c2) [L4\\_NOTHROW](#)

*Test if the capability indices of two capability selectors are equal.*

- [l4\\_cap\\_idx\\_t](#) [l4\\_capability\\_next](#) ([l4\\_cap\\_idx\\_t](#) c) [L4\\_NOTHROW](#)

*Get the next capability selector after c.*

## 17.278.1 Detailed Description

Common [L4](#) ABI Data Types.

Definition in file [types.h](#).

## 17.278.2 Function Documentation

### 17.278.2.1 [l4\\_capability\\_next\(\)](#)

```
l4_cap_idx_t l4_capability_next (
 l4_cap_idx_t c) [inline]
```

Get the next capability selector after c.

#### Parameters

|                   |                                                                        |
|-------------------|------------------------------------------------------------------------|
| <a href="#">c</a> | The capability selector for which the next selector shall be computed. |
|-------------------|------------------------------------------------------------------------|

#### Returns

The next capability selector after c.

Definition at line [457](#) of file [types.h](#).

References [L4\\_CAP\\_OFFSET](#), [L4\\_INLINE](#), and [L4\\_NOTHROW](#).

## 17.279 types.h

[Go to the documentation of this file.](#)

```

00001 /*****
00007 */
00008 * (c) 2008-2013 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00009 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00010 * Björn Döbel <doebel@os.inf.tu-dresden.de>,
00011 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00012 * economic rights: Technische Universität Dresden (Germany)
00013 *
00014 * License: see LICENSE.spdx (in this directory or the directories above)
00015 */
00016 /*****
00017 #pragma once
00018
00019 #include <l4/sys/l4int.h>
00020 #include <l4/sys/compiler.h>
00021 #include <l4/sys/consts.h>
00022
00023
00031
00038 enum L4_msgtag_protocol
00039 {
00040 L4_PROTO_NONE = 0,
00041 L4_PROTO_ALLOW_SYSCALL = 1,
00042 L4_PROTO_PF_EXCEPTION = 1,
00043
00044 L4_PROTO_IRQ = -1L,
00045 L4_PROTO_PAGE_FAULT = -2L,
00046 // -3L unused
00047 // -4L unused
00048 L4_PROTO_EXCEPTION = -5L,
00049 L4_PROTO_SIGMA0 = -6L,
00050 L4_PROTO_IO_PAGE_FAULT = -8L,
00051 L4_PROTO_THREAD_GROUP = -9L,
00052 L4_PROTO_KOBJECT = -10L,
00053 L4_PROTO_TASK = -11L,
00054 L4_PROTO_THREAD = -12L,
00055 L4_PROTO_LOG = -13L,
00056 L4_PROTO_SCHEDULER = -14L,
00057 L4_PROTO_FACTORY = -15L,
00058 L4_PROTO_VM = -16L,
00059 L4_PROTO_DMA_SPACE = -17L,
00060 L4_PROTO_IRQ_SENDER = -18L,
00061 // -19L unused
00062 L4_PROTO_SEMAPHORE = -20L,
00063 L4_PROTO_META = -21L,
00064 L4_PROTO_IOMMU = -22L,
00065 L4_PROTO_DEBUGGER = -23L,
00066 L4_PROTO_SMCCC = -24L,
00067 L4_PROTO_VCPU_CONTEXT = -25L,
00068 };
00069
00070 enum L4_varg_type
00071 {
00072 L4_VARG_TYPE_NIL = 0x00,
00073 L4_VARG_TYPE_UMWORD = 0x01,
00074 L4_VARG_TYPE_MWORD = 0x81,
00075 L4_VARG_TYPE_STRING = 0x02,
00076 L4_VARG_TYPE_FPAGE = 0x03,
00077
00078 L4_VARG_TYPE_SIGN = 0x80,
00079 };
00080
00081
00086 enum L4_msgtag_flags
00087 {
00088 // flags for received IPC
00093 L4_MSGTAG_ERROR = 0x8000,
00094
00095 // flags for sending IPC
00105 L4_MSGTAG_TRANSFER_FPU = 0x1000,
00114 L4_MSGTAG_SCHEDULE = 0x2000,
00127 L4_MSGTAG_PROPAGATE = 0x4000,
00128
00133 L4_MSGTAG_FLAGS = 0xf000,
00134 };
00135
00136
00153 typedef struct l4_msgtag_t
00154 {
00155 l4_mword_t raw;
00156 #ifdef __cplusplus
00158 long label() const L4_NOTHROW

```

```

00159 {
00160 #if defined(__cplusplus) && (__cplusplus >= 202002L)
00161 return raw >> 16;
00162 #else
00163 return raw < 0 ? ~(~raw >> 16) : raw >> 16;
00164 #endif
00165 }
00167 void label(long v) L4_NOTHROW { raw = (raw & 0xffff) | ((l4_umword_t)v << 16); }
00169 unsigned words() const L4_NOTHROW { return raw & 0x3f; }
00171 unsigned items() const L4_NOTHROW { return (raw >> 6) & 0x3f; }
00178 unsigned flags() const L4_NOTHROW { return raw & 0xf000; }
00180 bool is_page_fault() const L4_NOTHROW { return label() == L4_PROTO_PAGE_FAULT; }
00182 bool is_exception() const L4_NOTHROW { return label() == L4_PROTO_EXCEPTION; }
00184 bool is_sigma0() const L4_NOTHROW { return label() == L4_PROTO_SIGMA0; }
00186 bool is_io_page_fault() const L4_NOTHROW { return label() == L4_PROTO_IO_PAGE_FAULT; }
00191 bool has_error() const L4_NOTHROW { return raw & L4_MSGTAG_ERROR; }
00192 #endif
00193 } l4_msgtag_t;
00194
00195
00196
00208 L4_INLINE l4_msgtag_t l4_msgtag(long label, unsigned words, unsigned items,
00209 unsigned flags) L4_NOTHROW;
00210
00219 L4_INLINE long l4_msgtag_label(l4_msgtag_t t) L4_NOTHROW;
00220
00229 L4_INLINE unsigned l4_msgtag_words(l4_msgtag_t t) L4_NOTHROW;
00230
00239 L4_INLINE unsigned l4_msgtag_items(l4_msgtag_t t) L4_NOTHROW;
00240
00251 L4_INLINE unsigned l4_msgtag_flags(l4_msgtag_t t) L4_NOTHROW;
00252
00265 L4_INLINE unsigned l4_msgtag_has_error(l4_msgtag_t t) L4_NOTHROW;
00266
00275 L4_INLINE unsigned l4_msgtag_is_page_fault(l4_msgtag_t t) L4_NOTHROW;
00276
00285 L4_INLINE unsigned l4_msgtag_is_exception(l4_msgtag_t t) L4_NOTHROW;
00286
00295 L4_INLINE unsigned l4_msgtag_is_sigma0(l4_msgtag_t t) L4_NOTHROW;
00296
00305 L4_INLINE unsigned l4_msgtag_is_io_page_fault(l4_msgtag_t t) L4_NOTHROW;
00306
00319
00336 typedef unsigned long l4_cap_idx_t;
00337
00347 L4_INLINE unsigned l4_is_invalid_cap(l4_cap_idx_t c) L4_NOTHROW;
00348
00358 L4_INLINE unsigned l4_is_valid_cap(l4_cap_idx_t c) L4_NOTHROW;
00359
00373 L4_INLINE unsigned l4_capability_equal(l4_cap_idx_t c1, l4_cap_idx_t c2) L4_NOTHROW;
00374
00383 L4_INLINE l4_cap_idx_t l4_capability_next(l4_cap_idx_t c) L4_NOTHROW;
00384
00385 /* ***** */
00386 /* Implementation */
00387
00388 L4_INLINE unsigned
00389 l4_is_invalid_cap(l4_cap_idx_t c) L4_NOTHROW
00390 { return c & L4_INVALID_CAP_BIT; }
00391
00392 L4_INLINE unsigned
00393 l4_is_valid_cap(l4_cap_idx_t c) L4_NOTHROW
00394 { return !(c & L4_INVALID_CAP_BIT); }
00395
00396 L4_INLINE unsigned
00397 l4_capability_equal(l4_cap_idx_t c1, l4_cap_idx_t c2) L4_NOTHROW
00398 { return (c1 >> L4_CAP_SHIFT) == (c2 >> L4_CAP_SHIFT); }
00399
00400
00404 L4_INLINE
00405 l4_msgtag_t l4_msgtag(long label, unsigned words, unsigned items,
00406 unsigned flags) L4_NOTHROW
00407 {
00408 return (l4_msgtag_t){ (l4_mword_t)((l4_umword_t)label << 16)
00409 | (l4_mword_t)(words & 0x3f)
00410 | (l4_mword_t)((items & 0x3f) << 6)
00411 | (l4_mword_t)(flags & 0xf000)};
00412 }
00413
00414
00415
00416 L4_INLINE
00417 long l4_msgtag_label(l4_msgtag_t t) L4_NOTHROW
00418 {
00419 #if defined(__cplusplus) && (__cplusplus >= 202002L)
00420 return t.raw >> 16;
00421 #else

```



```

00422 return t.raw < 0 ? ~(t.raw >> 16) : t.raw >> 16;
00423 #endif
00424 }
00425
00426 L4_INLINE
00427 unsigned l4_msgtag_words(l4_msgtag_t t) L4_NOTHROW
00428 { return t.raw & 0x3f; }
00429
00430 L4_INLINE
00431 unsigned l4_msgtag_items(l4_msgtag_t t) L4_NOTHROW
00432 { return (t.raw >> 6) & 0x3f; }
00433
00434 L4_INLINE
00435 unsigned l4_msgtag_flags(l4_msgtag_t t) L4_NOTHROW
00436 { return t.raw & 0xf000; }
00437
00438
00439 L4_INLINE
00440 unsigned l4_msgtag_has_error(l4_msgtag_t t) L4_NOTHROW
00441 { return t.raw & L4_MSGTAG_ERROR; }
00442
00443
00444
00445 L4_INLINE unsigned l4_msgtag_is_page_fault(l4_msgtag_t t) L4_NOTHROW
00446 { return l4_msgtag_label(t) == L4_PROTO_PAGE_FAULT; }
00447
00448 L4_INLINE unsigned l4_msgtag_is_exception(l4_msgtag_t t) L4_NOTHROW
00449 { return l4_msgtag_label(t) == L4_PROTO_EXCEPTION; }
00450
00451 L4_INLINE unsigned l4_msgtag_is_sigma0(l4_msgtag_t t) L4_NOTHROW
00452 { return l4_msgtag_label(t) == L4_PROTO_SIGMA0; }
00453
00454 L4_INLINE unsigned l4_msgtag_is_io_page_fault(l4_msgtag_t t) L4_NOTHROW
00455 { return l4_msgtag_label(t) == L4_PROTO_IO_PAGE_FAULT; }
00456
00457 L4_INLINE l4_cap_idx_t l4_capability_next(l4_cap_idx_t c) L4_NOTHROW
00458 { return c + L4_CAP_OFFSET; }
00459
00460 #include <l4/sys/__l4_fpage.h>
00461 #include <l4/sys/__timeout.h>

```

## 17.280 virtio\_client.h

```

00001 /*
00002 * Copyright (C) 2018-2024 Kernkonzept GmbH.
00003 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/cxx/ref_ptr>
00010 #include <l4/cxx/unique_ptr_list>
00011 #include <l4/cxx/utis>
00012 #include <l4/sys/cache.h>
00013
00014 #include <l4/sys/task>
00015
00016 #include <l4/l4virtio/server/virtio-block>
00017
00018 #include <l4/libblock-device/debug.h>
00019 #include <l4/libblock-device/device.h>
00020 #include <l4/libblock-device/types.h>
00021 #include <l4/libblock-device/request.h>
00022
00023 namespace Block_device {
00024
00025 template <typename DEV>
00026 class Virtio_client
00027 : public L4virtio::Svr::Block_dev_base<Mem_region_info>,
00028 public L4::Epiface_t<Virtio_client<DEV>, L4virtio::Device>
00029 {
00030 protected:
00031 class Generic_pending_request : public Pending_request
00032 {
00033 protected:
00034 int check_error(int result)
00035 {
00036 if (result < 0 && result != -L4_EBUSY)
00037 client->handle_request_error(result, this);
00038
00039 return result;
00040 }
00041 };
00042 };
00043
00044 }

```

```

00041
00042 public:
00043 explicit Generic_pending_request(Virtio_client *c, cxx::unique_ptr<Request> &&req)
00044 : request(cxx::move(req)), client(c)
00045 {}
00046
00047 void fail_request() override
00048 {
00049 client->finalize_request(cxx::move(request), 0, L4VIRTIO_BLOCK_S_IOERR);
00050 }
00051
00052 cxx::unique_ptr<Request> request;
00053 Virtio_client *client;
00054 };
00055
00056 struct Pending_inout_request : public Generic_pending_request
00057 {
00058 Inout_block blocks;
00059 L4Re::Dma_space::Direction dir;
00060
00061 explicit Pending_inout_request(Virtio_client *c,
00062 cxx::unique_ptr<Request> &&req)
00063 : Generic_pending_request(c, cxx::move(req))
00064 {
00065 dir = this->request->header().type == L4VIRTIO_BLOCK_T_OUT
00066 ? L4Re::Dma_space::Direction::To_device
00067 : L4Re::Dma_space::Direction::From_device;
00068 }
00069
00070 ~Pending_inout_request() override
00071 {
00072 this->client->release_dma(this);
00073 }
00074
00075 int handle_request() override
00076 { return this->check_error(this->client->inout_request(this)); }
00077 };
00078
00079 struct Pending_flush_request : public Generic_pending_request
00080 {
00081 using Generic_pending_request::Generic_pending_request;
00082
00083 int handle_request() override
00084 { return this->check_error(this->client->flush_request(this)); }
00085 };
00086
00087 struct Pending_cmd_request : public Generic_pending_request
00088 {
00089 Inout_block blocks;
00090
00091 using Generic_pending_request::Generic_pending_request;
00092
00093 int handle_request() override
00094 {
00095 return this->check_error(this->client->discard_cmd_request(this, 0));
00096 }
00097 };
00098
00099 public:
00100 using Device_type = DEV;
00101
00102 Virtio_client(cxx::Ref_ptr<Device_type> const &dev, unsigned numds, bool readonly)
00103 : L4virtio::Svr::Block_dev_base<Mem_region_info>(L4VIRTIO_VENDOR_KK, 0x100,
00104 dev->capacity() > 9,
00105 dev->is_read_only()
00106 || readonly),
00107 _client_invalidate_cb(nullptr),
00108 _client_idle_cb(nullptr),
00109 _numds(numds),
00110 _device(dev),
00111 _in_flight(0)
00112 {
00113 reset_client();
00114 init_discard_info(0);
00115 }
00116
00117 void reset_device() override
00118 {
00119 if (_client_invalidate_cb)
00120 _client_invalidate_cb(false);
00121 _device->reset();
00122 _negotiated_features.raw = 0;
00123 }
00124
00125 void reset_client()
00126 {
00127 init_mem_info(_numds);

```

```

00142 set_seg_max(_device->max_segments());
00143 set_size_max(_device->max_size());
00144 set_flush();
00145 set_config_wce(0); // starting in write-through mode
00146 _shutdown_state = Shutdown_type::Running;
00147 _negotiated_features.raw = 0;
00148 }
00149
00150 bool queue_stopped() override
00151 { return _shutdown_state == Shutdown_type::Client_gone; }
00152
00153 // make these interfaces public so that a request scheduler can invoke them
00154 using L4virtio::Svr::Block_dev_base<Mem_region_info>::check_for_new_requests;
00155 using L4virtio::Svr::Block_dev_base<Mem_region_info>::get_request;
00156
00157 // make it possible for the request scheduler to register a direct callback
00158 void set_client_invalidate_cb(std::function<void(bool)> &&cb)
00159 {
00160 _client_invalidate_cb = cb;
00161 }
00162
00163 void set_client_idle_cb(std::function<void()> &&cb)
00164 {
00165 _client_idle_cb = cb;
00166 }
00167
00168 // make it possible for the request scheduler to register a device notify IRQ
00169 void set_device_notify_irq(L4::Cap<L4::Irq> irq)
00170 {
00171 _device_notify_irq = irq;
00172 }
00173
00174 L4::Cap<L4::Irq> device_notify_irq() const override
00175 {
00176 return _device_notify_irq;
00177 }
00178
00184 cxx::unique_ptr<Pending_request> start_request(cxx::unique_ptr<Request> &&req)
00185 {
00186 auto trace = Dbg::trace("virtio");
00187
00188 cxx::unique_ptr<Pending_request> pending;
00189
00190 if (_shutdown_state != Shutdown_type::Running)
00191 {
00192 trace.printf("Failing requests as the client is shutting down\n");
00193 this->finalize_request(cxx::move(req), 0, L4VIRTIO_BLOCK_S_IOERR);
00194 return pending;
00195 }
00196
00197 trace.printf("request received: type 0x%x, sector 0x%llx\n",
00198 req->header().type, req->header().sector);
00199 switch (req->header().type)
00200 {
00201 case L4VIRTIO_BLOCK_T_OUT:
00202 case L4VIRTIO_BLOCK_T_IN:
00203 {
00204 auto p = cxx::make_unique<Pending_inout_request>(this, cxx::move(req));
00205 int ret = build_inout_blocks(p.get());
00206 if (ret == L4_EOK)
00207 pending.reset(p.release());
00208 else
00209 handle_request_error(ret, p.get());
00210 break;
00211 }
00212 case L4VIRTIO_BLOCK_T_FLUSH:
00213 {
00214 auto p = cxx::make_unique<Pending_flush_request>(this, cxx::move(req));
00215 int ret = check_flush_request(p.get());
00216 if (ret == L4_EOK)
00217 pending.reset(p.release());
00218 else
00219 handle_request_error(ret, p.get());
00220 break;
00221 }
00222 case L4VIRTIO_BLOCK_T_WRITE_ZEROES:
00223 case L4VIRTIO_BLOCK_T_DISCARD:
00224 {
00225 auto p = cxx::make_unique<Pending_cmd_request>(this, cxx::move(req));
00226 int ret = build_discard_cmd_blocks(p.get());
00227 if (ret == L4_EOK)
00228 pending.reset(p.release());
00229 else
00230 handle_request_error(ret, p.get());
00231 break;
00232 }
00233 default:

```

```

00234 finalize_request(cxx::move(req), 0, L4VIRTIO_BLOCK_S_UNSUPP);
00235 break;
00236 }
00237
00238 return pending;
00239 }
00240
00241 void task_finished(Generic_pending_request *preq, int error, l4_size_t sz)
00242 {
00243 _in_flight--;
00244
00245 // move on to the next request
00246
00247 // Only finalize if the client is still alive
00248 if (_shutdown_state != Client_gone)
00249 finalize_request(cxx::move(preq->request), sz, error);
00250
00251 // New requests might be schedulable
00252 if (_client_idle_cb)
00253 _client_idle_cb();
00254
00255 // pending request can be dropped
00256 cxx::unique_ptr<Pending_request> ureq(preq);
00257 }
00258
00262 void shutdown_event(Shutdown_type type)
00263 {
00264 // If the client is already in the Client_gone state, it means that it was
00265 // already shutdown and this is another go at its removal. This situation
00266 // can occur because at the time of its previous removal attempt there were
00267 // still I/O requests in progress.
00268 if (_shutdown_state == Client_gone)
00269 return;
00270
00271 // Transitions from System_shutdown are also not allowed, the initiator
00272 // should take care of graceful handling of this.
00273 l4_assert(_shutdown_state != System_shutdown);
00274 // If we are transitioning from System_suspend, it must be only to Running,
00275 // the initiator should handle this gracefully.
00276 l4_assert(_shutdown_state != System_suspend
00277 || type == Shutdown_type::Running);
00278
00279 // Update shutdown state of the client
00280 _shutdown_state = type;
00281
00282 if (type == Shutdown_type::Client_shutdown)
00283 {
00284 reset();
00285 reset_client();
00286 // Client_shutdown must transit to the Running state
00287 l4_assert(_shutdown_state == Shutdown_type::Running);
00288 }
00289
00290 if (type != Shutdown_type::Running)
00291 {
00292 if (_client_invalidate_cb)
00293 _client_invalidate_cb(type != Shutdown_type::Client_gone);
00294 _device->reset();
00295 }
00296 }
00297
00310 L4::Cap<void> register_obj(L4::Registry_iface *registry,
00311 char const *service = 0)
00312 {
00313 L4::Cap<void> ret;
00314 if (service)
00315 ret = registry->register_obj(this, service);
00316 else
00317 ret = registry->register_obj(this);
00318 L4Re::chkcap(ret);
00319
00320 return ret;
00321 }
00322
00323 L4::Cap<void> register_obj(L4::Registry_iface *registry,
00324 L4::Cap<L4::Rcv_endpoint> ep)
00325 {
00326 return L4Re::chkcap(registry->register_obj(this, ep));
00327 }
00328
00334 void unregister_obj(L4::Registry_iface *registry)
00335 {
00336 registry->unregister_obj(this);
00337 }
00338
00339 bool busy() const
00340 {

```

```

00341 return _in_flight != 0;
00342 }
00343
00344 Notification_domain const *notification_domain() const
00345 { return _device->notification_domain(); }
00346
00347 protected:
00348 L4::Ipc_svr::Server_iface *server_iface() const override
00349 {
00350 return this->L4::Epiface::server_iface();
00351 }
00352
00353 private:
00354 void release_dma(Pending_inout_request *req)
00355 {
00356 // unmap DMA regions
00357 Inout_block *cur = &req->blocks;
00358 while (cur)
00359 {
00360 if (cur->num_sectors)
00361 _device->dma_unmap(cur->dma_addr, cur->num_sectors, req->dir);
00362 cur = cur->next.get();
00363 }
00364 }
00365
00366 int build_inout_blocks(Pending_inout_request *preq)
00367 {
00368 auto *req = preq->request.get();
00369 l4_size_t sps = _device->sector_size() >> 9;
00370 l4_uint64_t current_sector = req->header().sector / sps;
00371 l4_uint64_t sectors = _device->capacity() / _device->sector_size();
00372 auto dir = preq->dir;
00373
00374 l4_uint32_t flags = 0;
00375 if (req->header().type == L4VIRTIO_BLOCK_T_OUT)
00376 {
00377 // If RO was offered, every write must fail
00378 if (device_features().ro())
00379 return -L4_EIO;
00380
00381 // Figure out whether the write has a write-through or write-back semantics
00382 if (_negotiated_features.config_wce())
00383 {
00384 if (get_writeback() == 1)
00385 flags = Block_device::Inout_f_wb;
00386 }
00387 else if (_negotiated_features.flush())
00388 flags = Block_device::Inout_f_wb;
00389 }
00390
00391 // Check alignment of the first sector
00392 if (current_sector * sps != req->header().sector)
00393 return -L4_EIO;
00394
00395 Inout_block *last_blk = nullptr;
00396
00397 size_t seg = 0;
00398
00399 while (req->has_more())
00400 {
00401 Request::Data_block b;
00402
00403 if (++seg > _device->max_segments())
00404 return -L4_EIO;
00405
00406 try
00407 {
00408 b = req->next_block();
00409 }
00410 catch (L4virtio::Svr::Bad_descriptor const &e)
00411 {
00412 Dbg::warn().printf("Descriptor error: %s\n", e.message());
00413 return -L4_EIO;
00414 }
00415
00416 l4_size_t off = b.mem->ds_offset() + (l4_addr_t) b.addr
00417 - (l4_addr_t) b.mem->local_base();
00418
00419 l4_size_t sz = b.len / _device->sector_size();
00420
00421 if (sz * _device->sector_size() != b.len)
00422 {
00423 Dbg::warn().printf("Bad block size 0x%x\n", b.len);
00424 return -L4_EIO;
00425 }
00426
00427 // Check bounds

```

```

00428 if (sz > sectors)
00429 return -L4_EIO;
00430 if (current_sector > sectors - sz)
00431 return -L4_EIO;
00432
00433 Inout_block *blk;
00434 if (last_blk)
00435 {
00436 last_blk->next = cxx::make_unique<Inout_block>();
00437 blk = last_blk->next.get();
00438 }
00439 else
00440 blk = &preg->blocks;
00441
00442 L4Re::Dma_space::Dma_addr phys;
00443 long ret = _device->dma_map(b.mem, off, sz, dir, &phys);
00444 if (ret < 0)
00445 return ret;
00446
00447 blk->dma_addr = phys;
00448 blk->virt_addr = b.addr;
00449 blk->num_sectors = sz;
00450 current_sector += sz;
00451 blk->flags = flags;
00452
00453 last_blk = blk;
00454 }
00455
00456 return L4_EOK;
00457 }
00458
00459 void maintain_cache_before_req(Pending_inout_request const *preg)
00460 {
00461 if (preg->dir == L4Re::Dma_space::None)
00462 return;
00463 for (Inout_block const *cur = &preg->blocks; cur; cur = cur->next.get())
00464 {
00465 l4_addr_t vstart = (l4_addr_t)cur->virt_addr;
00466 if (vstart)
00467 {
00468 l4_size_t vsize = cur->num_sectors * _device->sector_size();
00469 if (preg->dir == L4Re::Dma_space::From_device)
00470 l4_cache_inv_data(vstart, vstart + vsize);
00471 else if (preg->dir == L4Re::Dma_space::To_device)
00472 l4_cache_clean_data(vstart, vstart + vsize);
00473 else // L4Re::Dma_space::Bidirectional
00474 l4_cache_flush_data(vstart, vstart + vsize);
00475 }
00476 }
00477 }
00478
00479 void maintain_cache_after_req(Pending_inout_request const *preg)
00480 {
00481 if (preg->dir == L4Re::Dma_space::None)
00482 return;
00483 for (Inout_block const *cur = &preg->blocks; cur; cur = cur->next.get())
00484 {
00485 l4_addr_t vstart = (l4_addr_t)cur->virt_addr;
00486 if (vstart)
00487 {
00488 l4_size_t vsize = cur->num_sectors * _device->sector_size();
00489 if (preg->dir != L4Re::Dma_space::To_device)
00490 l4_cache_inv_data(vstart, vstart + vsize);
00491 }
00492 }
00493 }
00494
00495 int inout_request(Pending_inout_request *preg)
00496 {
00497 auto *req = preg->request.get();
00498 l4_uint64_t sector = req->header().sector / (_device->sector_size() >> 9);
00499
00500 maintain_cache_before_req(preg);
00501 int res = _device->inout_data(
00502 sector, preg->blocks,
00503 [this, preg](int error, l4_size_t sz) {
00504 maintain_cache_after_req(preg);
00505 task_finished(preg, error, sz);
00506 },
00507 preg->dir);
00508
00509 // request successfully submitted to device
00510 if (res >= 0)
00511 _in_flight++;
00512
00513 return res;
00514 }

```

```

00515
00516 int check_flush_request(Pending_flush_request *preq)
00517 {
00518 if (!_negotiated_features.flush())
00519 return -L4_ENOSYS;
00520
00521 auto *req = preq->request.get();
00522
00523 // sector must be zero for FLUSH
00524 if (req->header().sector)
00525 return -L4_ENOSYS;
00526
00527 return L4_EOK;
00528 }
00529
00530 int flush_request(Pending_flush_request *preq)
00531 {
00532 int res = _device->flush([this, preq](int error, l4_size_t sz) {
00533 task_finished(preq, error, sz);
00534 });
00535
00536 // request successfully submitted to device
00537 if (res >= 0)
00538 _in_flight++;
00539
00540 return res;
00541 }
00542
00543 bool check_features(void) override
00544 {
00545 _negotiated_features = negotiated_features();
00546 return true;
00547 }
00548
00549 template <typename T = Device_type>
00550 void init_discard_info(long) {}
00551
00552 template <typename T = Device_type>
00553 auto init_discard_info(int)
00554 -> decltype(((T*)0)->discard_info(), void())
00555 {
00556 _di = _device->discard_info();
00557
00558 // Convert sector sizes to virtio 512-byte sectors.
00559 size_t sps = _device->sector_size() >> 9;
00560 if (_di.max_discard_sectors)
00561 set_discard(_di.max_discard_sectors * sps, _di.max_discard_seg,
00562 _di.discard_sector_alignment * sps);
00563 if (_di.max_write_zeroes_sectors)
00564 set_write_zeroes(_di.max_write_zeroes_sectors * sps,
00565 _di.max_write_zeroes_seg, _di.write_zeroes_may_unmap);
00566 }
00567
00568 int build_discard_cmd_blocks(Pending_cmd_request *preq)
00569 {
00570 auto *req = preq->request.get();
00571 bool discard = (req->header().type == L4VIRTIO_BLOCK_T_DISCARD);
00572
00573 if (this->device_features().ro())
00574 return -L4_EIO;
00575
00576 // sector is used only for inout requests, it must be zero for WzD
00577 if (req->header().sector)
00578 return -L4_ENOSYS;
00579
00580 if (discard)
00581 {
00582 if (!_negotiated_features.discard())
00583 return -L4_ENOSYS;
00584 }
00585 else
00586 {
00587 if (!_negotiated_features.write_zeroes())
00588 return -L4_ENOSYS;
00589 }
00590
00591 auto *d = _device.get();
00592
00593 size_t seg = 0;
00594 size_t max_seg = discard ? _di.max_discard_seg : _di.max_write_zeroes_seg;
00595
00596 l4_size_t sps = d->sector_size() >> 9;
00597 l4_uint64_t sectors = d->capacity() / d->sector_size();
00598
00599 Inout_block *last_blk = nullptr;
00600
00601 while (req->has_more())

```

```

00602 {
00603 Request::Data_block b;
00604
00605 try
00606 {
00607 b = req->next_block();
00608 }
00609 catch (L4virtio::Svr::Bad_descriptor const &e)
00610 {
00611 Dbg::warn().printf("Descriptor error: %s\n", e.message());
00612 return -L4_EIO;
00613 }
00614
00615 auto *payload = reinterpret_cast<l4virtio_block_discard_t *>(b.addr);
00616
00617 size_t items = b.len / sizeof(payload[0]);
00618 if (items * sizeof(payload[0]) != b.len)
00619 return -L4_EIO;
00620
00621 if (seg + items > max_seg)
00622 return -L4_EIO;
00623 seg += items;
00624
00625 for (auto i = 0u; i < items; i++)
00626 {
00627 auto p = cxx::access_once<l4virtio_block_discard_t>(&payload[i]);
00628
00629 // Check sector size alignment. Discard sector alignment is not
00630 // strictly enforced as it is merely a hint to the driver.
00631 if (p.sector % sps != 0)
00632 return -L4_EIO;
00633 if (p.num_sectors % sps != 0)
00634 return -L4_EIO;
00635
00636 // Convert to the device sector size
00637 p.sector /= sps;
00638 p.num_sectors /= sps;
00639
00640 // Check bounds
00641 if (p.num_sectors > sectors)
00642 return -L4_EIO;
00643 if (p.sector > sectors - p.num_sectors)
00644 return -L4_EIO;
00645
00646 if (p.flags & L4VIRTIO_BLOCK_DISCARD_F_RESERVED)
00647 return -L4_ENOSYS;
00648
00649 Inout_block *blk;
00650 if (last_blk)
00651 {
00652 last_blk->next = cxx::make_unique<Inout_block>();
00653 blk = last_blk->next.get();
00654 }
00655 else
00656 blk = &preq->blocks;
00657
00658 blk->sector = p.sector;
00659 blk->num_sectors = p.num_sectors;
00660
00661 if (discard)
00662 {
00663 if (p.flags & L4VIRTIO_BLOCK_DISCARD_F_UNMAP)
00664 return -L4_ENOSYS;
00665 if (p.num_sectors > _di.max_discard_sectors)
00666 return -L4_EIO;
00667 }
00668 else
00669 {
00670 if (p.flags & L4VIRTIO_BLOCK_DISCARD_F_UNMAP
00671 && _di.write_zeroes_may_unmap)
00672 blk->flags = Inout_f_unmap;
00673 if (p.num_sectors > _di.max_write_zeroes_sectors)
00674 return -L4_EIO;
00675 }
00676
00677 last_blk = blk;
00678 }
00679 }
00680
00681 return L4_EOK;
00682 }
00683
00684 template <typename T = Device_type>
00685 int discard_cmd_request(Pending_cmd_request *, long)
00686 { return -L4_EIO; }
00687
00688 template <typename T = Device_type>

```



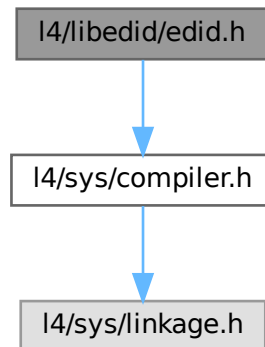
```

00689 auto discard_cmd_request(Pending_cmd_request *preq, int)
00690 -> decltype(((T*)0)->discard_info(), int())
00691 {
00692 auto *req = preq->request.get();
00693 bool discard = (req->header().type == L4VIRTIO_BLOCK_T_DISCARD);
00694
00695 int res = _device->discard(
00696 0, preq->blocks,
00697 [this, preq](int error, l4_size_t sz) { task_finished(preq, error, sz); },
00698 discard);
00699
00700 // request successfully submitted to device
00701 if (res >= 0)
00702 _in_flight++;
00703
00704 return res;
00705 }
00706
00707 // only use on errors that are not busy
00708 void handle_request_error(int error, Generic_pending_request *pending)
00709 {
00710 auto trace = Dbg::trace("virtio");
00711
00712 if (error == -L4_ENOSYS)
00713 {
00714 trace.printf("Unsupported operation.\n");
00715 finalize_request(cxx::move(pending->request), 0,
00716 L4VIRTIO_BLOCK_S_UNSUPP);
00717 }
00718 else
00719 {
00720 trace.printf("Got IO error: %d\n", error);
00721 finalize_request(cxx::move(pending->request), 0, L4VIRTIO_BLOCK_S_IOERR);
00722 }
00723 }
00724
00725 protected:
00726 L4::Cap<L4::Irq> _device_notify_irq;
00727 std::function<void(bool)> _client_invalidate_cb;
00728 std::function<void()> _client_idle_cb;
00729 unsigned _numds;
00730 Shutdown_type _shutdown_state;
00731 cxx::Ref_ptr<Device_type> _device;
00732 Device_discard_feature::Discard_info _di;
00733
00734 L4virtio::Svr::Block_features _negotiated_features;
00735
00736 unsigned _in_flight;
00737 };
00738
00739 } //name space

```

## 17.281 I4/libedid/edid.h File Reference

```
#include <l4/sys/compiler.h>
Include dependency graph for edid.h:
```



### Enumerations

- enum [Libedid\\_consts](#) { [Libedid\\_block\\_size](#) = 128 }  
*EDID constants.*

### Functions

- int [libedid\\_check\\_header](#) (const unsigned char \*edid)  
*Check for valid EDID header.*
- int [libedid\\_checksum](#) (const unsigned char \*edid)  
*Calculates the EDID checksum.*
- unsigned [libedid\\_version](#) (const unsigned char \*edid)  
*Returns the EDID version number.*
- unsigned [libedid\\_revision](#) (const unsigned char \*edid)  
*Returns the EDID revision number.*
- void [libedid\\_pnp\\_id](#) (const unsigned char \*edid, unsigned char \*id)  
*Extracts the display's PnP ID.*
- void [libedid\\_preferred\\_resolution](#) (const unsigned char \*edid, unsigned \*w, unsigned \*h)  
*Extract the display's preferred mode.*
- unsigned [libedid\\_num\\_ext\\_blocks](#) (const unsigned char \*edid)  
*Get the number of EDID extension blocks.*
- unsigned [libedid\\_dump\\_standard\\_timings](#) (const unsigned char \*edid)  
*Dump the standard timings to stdout.*
- void [libedid\\_dump](#) (const unsigned char \*edid)  
*Dump raw EDID data to stdout.*

## 17.282 edid.h

[Go to the documentation of this file.](#)

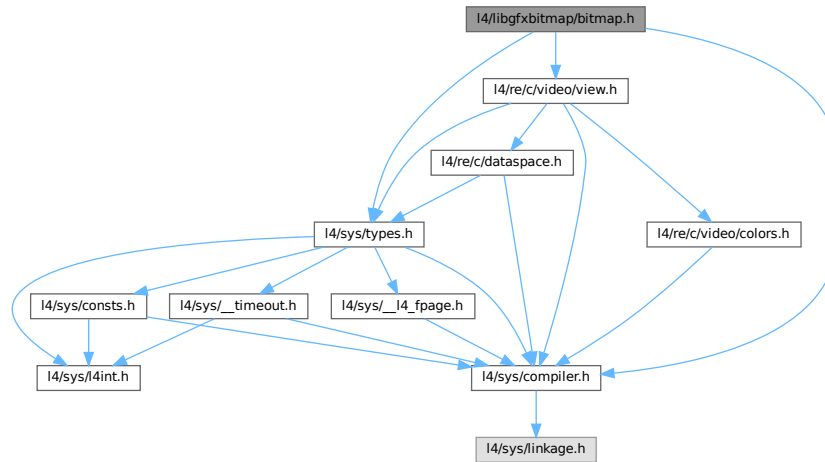
```
00001
00004 /*
00005 * (c) 2014 Matthias Lange <matthias.lange@kernkonzept.com>
00006 *
00007 * This file is part of TUD:OS and distributed under the terms of the
00008 * GNU Lesser General Public License 2.1.
00009 * Please see the COPYING-LGPL-2.1 file for details.
00010 */
00011 #pragma once
00012
00013 #include <l4/sys/compiler.h>
00014
00019
00023 enum Libedid_consts
00024 {
00025 Libedid_block_size = 128,
00026 };
00027
00028 __BEGIN_DECLS
00029
00037 int libedid_check_header(const unsigned char *edid);
00038
00046 int libedid_checksum(const unsigned char *edid);
00047
00055 unsigned libedid_version(const unsigned char *edid);
00056
00064 unsigned libedid_revision(const unsigned char *edid);
00065
00072 void libedid_pnp_id(const unsigned char *edid, unsigned char *id);
00073
00081 void libedid_prefered_resolution(const unsigned char *edid,
00082 unsigned *w, unsigned *h);
00083
00091 unsigned libedid_num_ext_blocks(const unsigned char *edid);
00092
00100 unsigned libedid_dump_standard_timings(const unsigned char *edid);
00101
00107 void libedid_dump(const unsigned char *edid);
00108
00110
00111 __END_DECLS
```

## 17.283 l4/libgfxbitmap/bitmap.h File Reference

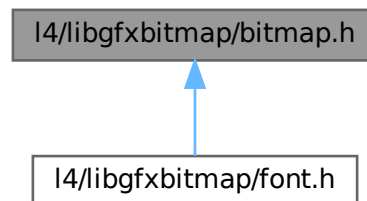
Bitmap renderer header file.

```
#include <l4/sys/compiler.h>
#include <l4/sys/types.h>
#include <l4/re/c/video/view.h>
```

Include dependency graph for `bitmap.h`:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct `gfxbitmap_offset`  
offsets in `pmap[]` and `bmap[]`

## Param macros for `bmap_*`

Bitmap type - start least or start most significant bit

- `#define pSLIM_BMAP_START_MSB 0x02`  
*'pbm'-style: "The bits are stored eight per byte, high bit first low bit last." \*/ #define pSLIM\_BMAP\_START\_LSB /\*!< the other way round*
- typedef unsigned int `gfxbitmap_color_t`  
Standard color type.
- typedef unsigned int `gfxbitmap_color_pix_t`

*Specific color type.*

- [gfxbitmap\\_color\\_pix\\_t](#) [gfxbitmap\\_convert\\_color](#) ([l4re\\_video\\_view\\_info\\_t](#) \*vi, [gfxbitmap\\_color\\_t](#) rgb)

*Convert a color.*

- void [gfxbitmap\\_fill](#) ([l4\\_uint8\\_t](#) \*vfb, [l4re\\_video\\_view\\_info\\_t](#) \*vi, int x, int y, int w, int h, [gfxbitmap\\_color\\_pix\\_t](#) color)

*Fill a rectangular area with a color.*

- void [gfxbitmap\\_bmap](#) ([l4\\_uint8\\_t](#) \*vfb, [l4re\\_video\\_view\\_info\\_t](#) \*vi, [l4\\_int16\\_t](#) x, [l4\\_int16\\_t](#) y, [l4\\_uint32\\_t](#) w, [l4\\_uint32\\_t](#) h, [l4\\_uint8\\_t](#) \*bmap, [gfxbitmap\\_color\\_pix\\_t](#) fgc, [gfxbitmap\\_color\\_pix\\_t](#) bgc, struct [gfxbitmap\\_offset](#) \*offset, [l4\\_uint8\\_t](#) mode)

*Fill a rectangular area with a bicolor bitmap pattern.*

- void [gfxbitmap\\_set](#) ([l4\\_uint8\\_t](#) \*vfb, [l4re\\_video\\_view\\_info\\_t](#) \*vi, [l4\\_int16\\_t](#) x, [l4\\_int16\\_t](#) y, [l4\\_uint32\\_t](#) w, [l4\\_uint32\\_t](#) h, [l4\\_uint32\\_t](#) xoffs, [l4\\_uint32\\_t](#) yoffs, [l4\\_uint8\\_t](#) \*pmap, struct [gfxbitmap\\_offset](#) \*offset, [l4\\_uint32\\_t](#) pwidth)

*Set area from source area.*

- void [gfxbitmap\\_copy](#) ([l4\\_uint8\\_t](#) \*dest, [l4\\_uint8\\_t](#) \*src, [l4re\\_video\\_view\\_info\\_t](#) \*vi, int x, int y, int w, int h, int dx, int dy)

*Copy a rectangular area.*

## 17.283.1 Detailed Description

Bitmap renderer header file.

Definition in file [bitmap.h](#).

## 17.283.2 Typedef Documentation

### 17.283.2.1 [gfxbitmap\\_color\\_pix\\_t](#)

```
typedef unsigned int gfxbitmap_color_pix_t
```

Specific color type.

This color type is specific for a particular framebuffer, it can be use to write pixel on a framebuffer. Use [gfxbitmap\\_convert\\_color](#) to convert from [gfxbitmap\\_color\\_t](#) to [gfxbitmap\\_color\\_pix\\_t](#).

Definition at line 65 of file [bitmap.h](#).

### 17.283.2.2 [gfxbitmap\\_color\\_t](#)

```
typedef unsigned int gfxbitmap_color_t
```

Standard color type.

It's a RGB type with 8bits for each channel, regardless of the framebuffer used.

Definition at line 56 of file [bitmap.h](#).

### 17.283.3 Function Documentation

#### 17.283.3.1 gfxbitmap\_bmap()

```
void gfxbitmap_bmap (
 l4_uint8_t * vfb,
 l4re_video_view_info_t * vi,
 l4_int16_t x,
 l4_int16_t y,
 l4_uint32_t w,
 l4_uint32_t h,
 l4_uint8_t * bmap,
 gfxbitmap_color_pix_t fg,
 gfxbitmap_color_pix_t bg,
 struct gfxbitmap_offset * offset,
 l4_uint8_t mode)
```

Fill a rectangular area with a bicolor bitmap pattern.

#### Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>vfb</i>    | Frame buffer.                       |
| <i>vi</i>     | Frame buffer information structure. |
| <i>x</i>      | X position of area.                 |
| <i>y</i>      | Y position of area.                 |
| <i>w</i>      | Width of area.                      |
| <i>h</i>      | Height of area.                     |
| <i>bmap</i>   | Bitmap pattern.                     |
| <i>fg</i>     | Foreground color.                   |
| <i>bg</i>     | Background color.                   |
| <i>offset</i> | Offsets.                            |
| <i>mode</i>   | Mode                                |

See also

[pSLIM\\_BMAP\\_START\\_MSB](#) and [#pSLIM\\_BMAP\\_START\\_LSB](#).

#### 17.283.3.2 gfxbitmap\_convert\_color()

```
gfxbitmap_color_pix_t gfxbitmap_convert_color (
 l4re_video_view_info_t * vi,
 gfxbitmap_color_t rgb)
```

Convert a color.

Converts a given color in standard format to the format used in the framebuffer.

### 17.283.3.3 gfxbitmap\_copy()

```
void gfxbitmap_copy (
 l4_uint8_t * dest,
 l4_uint8_t * src,
 l4re_video_view_info_t * vi,
 int x,
 int y,
 int w,
 int h,
 int dx,
 int dy)
```

Copy a rectangular area.

#### Parameters

|             |                                     |
|-------------|-------------------------------------|
| <i>dest</i> | Destination frame buffer.           |
| <i>src</i>  | Source frame buffer.                |
| <i>vi</i>   | Frame buffer information structure. |
| <i>x</i>    | Source X position of area.          |
| <i>y</i>    | Source Y position of area.          |
| <i>w</i>    | Width of area.                      |
| <i>h</i>    | Height of area.                     |
| <i>dx</i>   | Source X position of area.          |
| <i>dy</i>   | Source Y position of area.          |

References [L4\\_END\\_DECLS](#).

### 17.283.3.4 gfxbitmap\_fill()

```
void gfxbitmap_fill (
 l4_uint8_t * vfb,
 l4re_video_view_info_t * vi,
 int x,
 int y,
 int w,
 int h,
 gfxbitmap_color_pix_t color)
```

Fill a rectangular area with a color.

#### Parameters

|            |                                     |
|------------|-------------------------------------|
| <i>vfb</i> | Frame buffer.                       |
| <i>vi</i>  | Frame buffer information structure. |
| <i>x</i>   | X position of area.                 |
| <i>y</i>   | Y position of area.                 |

|              |                 |
|--------------|-----------------|
| <i>w</i>     | Width of area.  |
| <i>h</i>     | Height of area. |
| <i>color</i> | Color of area.  |

### 17.283.3.5 gfxbitmap\_set()

```
void gfxbitmap_set (
 14_uint8_t * vfb,
 14re_video_view_info_t * vi,
 14_int16_t x,
 14_int16_t y,
 14_uint32_t w,
 14_uint32_t h,
 14_uint32_t xoffs,
 14_uint32_t yoffs,
 14_uint8_t * pmap,
 struct gfxbitmap_offset * offset,
 14_uint32_t pwidth)
```

Set area from source area.

#### Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>vfb</i>    | Frame buffer.                       |
| <i>vi</i>     | Frame buffer information structure. |
| <i>x</i>      | X position of area.                 |
| <i>y</i>      | Y position of area.                 |
| <i>w</i>      | Width of area.                      |
| <i>h</i>      | Height of area.                     |
| <i>pmap</i>   | Source.                             |
| <i>xoffs</i>  | X offset.                           |
| <i>yoffs</i>  | Y offset.                           |
| <i>offset</i> | Offsets.                            |
| <i>pwidth</i> | Width of source in bytes.           |

## 17.284 bitmap.h

[Go to the documentation of this file.](#)

```
00001
00005 /*
00006 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 * This file is part of TUD:OS and distributed under the terms of the
00009 * GNU Lesser General Public License 2.1.
00010 * Please see the COPYING-LGPL-2.1 file for details.
00011 */
00012 #pragma once
00013
00014 #include <14/sys/compiler.h>
00015 #include <14/sys/types.h>
00016 #include <14/re/c/video/view.h>
00017
```



```

00027
00032
00033 L4_BEGIN_DECLS
00039 #define pSLIM_BMAP_START_MSB 0x02
00042 #define pSLIM_BMAP_START_LSB 0x01
00044
00049
00056 typedef unsigned int gfxbitmap_color_t;
00057
00065 typedef unsigned int gfxbitmap_color_pix_t;
00066
00068 struct gfxbitmap_offset
00069 {
00070 l4_uint32_t preskip_x;
00071 l4_uint32_t preskip_y;
00072 l4_uint32_t endskip_x;
00073 };
00074
00081 gfxbitmap_color_pix_t
00082 gfxbitmap_convert_color(l4re_video_view_info_t *vi, gfxbitmap_color_t rgb);
00083
00095 void
00096 gfxbitmap_fill(l4_uint8_t *vfb, l4re_video_view_info_t *vi,
00097 int x, int y, int w, int h, gfxbitmap_color_pix_t color);
00098
00116 void
00117 gfxbitmap_bmap(l4_uint8_t *vfb, l4re_video_view_info_t *vi,
00118 l4_int16_t x, l4_int16_t y, l4_uint32_t w,
00119 l4_uint32_t h, l4_uint8_t *bmap,
00120 gfxbitmap_color_pix_t fg, gfxbitmap_color_pix_t bg,
00121 struct gfxbitmap_offset *offset, l4_uint8_t mode);
00122
00138 void
00139 gfxbitmap_set(l4_uint8_t *vfb, l4re_video_view_info_t *vi,
00140 l4_int16_t x, l4_int16_t y, l4_uint32_t w,
00141 l4_uint32_t h, l4_uint32_t xoffs, l4_uint32_t yoffs,
00142 l4_uint8_t *pmap, struct gfxbitmap_offset *offset,
00143 l4_uint32_t pwidth);
00144
00158 void
00159 gfxbitmap_copy(l4_uint8_t *dest, l4_uint8_t *src, l4re_video_view_info_t *vi,
00160 int x, int y, int w, int h, int dx, int dy);
00162
00163 L4_END_DECLS

```

## 17.285 l4/libgfxbitmap/font.h File Reference

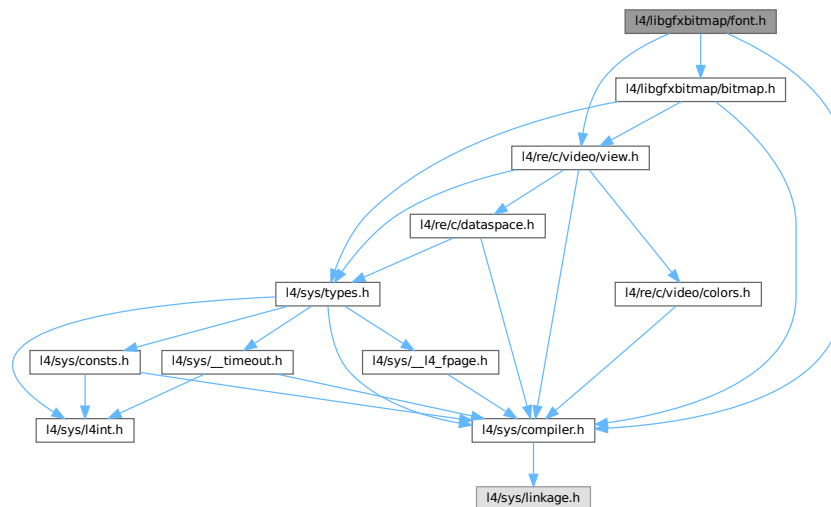
Bitmap font renderer header file.

```

#include <l4/sys/compiler.h>
#include <l4/re/c/video/view.h>
#include <l4/libgfxbitmap/bitmap.h>

```

Include dependency graph for font.h:



## Macros

- `#define GFXBITMAP_DEFAULT_FONT (void *)0`  
Constant to use for the default font.

## Enumerations

- enum  
Constant for length field.

## Functions

- int `gfxbitmap_font_init` (void)  
Initialize the library.
- `gfxbitmap_font_t` `gfxbitmap_font_get` (const char \*name)  
Get a font descriptor.
- unsigned `gfxbitmap_font_width` (`gfxbitmap_font_t` font)  
Get the font width.
- unsigned `gfxbitmap_font_height` (`gfxbitmap_font_t` font)  
Get the font height.
- void \* `gfxbitmap_font_data` (`gfxbitmap_font_t` font, unsigned c)  
Get bitmap font data for a specific character.
- void `gfxbitmap_font_text` (void \*fb, `l4re_video_view_info_t` \*vi, `gfxbitmap_font_t` font, const char \*text, unsigned len, unsigned x, unsigned y, `gfxbitmap_color_pix_t` fg, `gfxbitmap_color_pix_t` bg)  
Render a string to a framebuffer.
- void `gfxbitmap_font_text_scale` (void \*fb, `l4re_video_view_info_t` \*vi, `gfxbitmap_font_t` font, const char \*text, unsigned len, unsigned x, unsigned y, `gfxbitmap_color_pix_t` fg, `gfxbitmap_color_pix_t` bg, int scale\_x, int scale\_y)  
Render a string to a framebuffer, including scaling.

## Variables

- [L4\\_BEGIN\\_DECLS](#) typedef void \* **gfxbitmap\_font\_t**  
*Font.*

## 17.285.1 Detailed Description

Bitmap font renderer header file.

Definition in file [font.h](#).

## 17.285.2 Enumeration Type Documentation

### 17.285.2.1 anonymous enum

`anonymous enum`

Constant for length field.

Use this if the function should call strlen on the text argument itself.

Definition at line 38 of file [font.h](#).

## 17.285.3 Function Documentation

### 17.285.3.1 gfxbitmap\_font\_data()

```
void * gfxbitmap_font_data (
 gfxbitmap_font_t font,
 unsigned c)
```

Get bitmap font data for a specific character.

#### Parameters

|             |            |
|-------------|------------|
| <i>font</i> | Font.      |
| <i>c</i>    | Character. |

#### Returns

Pointer to bmap data, NULL on error.

References [gfxbitmap\\_font\\_t](#), and [L4\\_CV](#).

### 17.285.3.2 gfxbitmap\_font\_get()

```
gfxbitmap_font_t gfxbitmap_font_get (
 const char * name)
```

Get a font descriptor.

#### Parameters

---

|             |                   |
|-------------|-------------------|
| <i>name</i> | Name of the font. |
|-------------|-------------------|

**Returns**

A (opaque) font descriptor, or NULL if font could not be found.

References [gfxbitmap\\_font\\_t](#), and [L4\\_CV](#).

**17.285.3.3 gfxbitmap\_font\_height()**

```
unsigned gfxbitmap_font_height (
 gfxbitmap_font_t font)
```

Get the font height.

**Parameters**

|             |       |
|-------------|-------|
| <i>font</i> | Font. |
|-------------|-------|

**Returns**

Font height, 0 if font height could not be retrieved.

References [gfxbitmap\\_font\\_t](#), and [L4\\_CV](#).

**17.285.3.4 gfxbitmap\_font\_init()**

```
int gfxbitmap_font_init (
 void)
```

Initialize the library.

This function must be called before any other font function of this library.

**Returns**

0 on success, other on error

References [L4\\_CV](#).

**17.285.3.5 gfxbitmap\_font\_text()**

```
void gfxbitmap_font_text (
 void * fb,
 l4re_video_view_info_t * vi,
 gfxbitmap_font_t font,
 const char * text,
 unsigned len,
 unsigned x,
 unsigned y,
 gfxbitmap_color_pix_t fg,
 gfxbitmap_color_pix_t bg)
```

Render a string to a framebuffer.

**Parameters**

|             |                                          |
|-------------|------------------------------------------|
| <i>fb</i>   | Pointer to frame buffer.                 |
| <i>vi</i>   | Frame buffer info structure.             |
| <i>font</i> | Font.                                    |
| <i>text</i> | Text string.                             |
| <i>len</i>  | Length of the text string.               |
| <i>x</i>    | Horizontal position in the frame buffer. |
| <i>y</i>    | Vertical position in the frame buffer.   |
| <i>fg</i>   | Foreground color.                        |
| <i>bg</i>   | Background color.                        |

References [gfxbitmap\\_font\\_t](#), and [L4\\_CV](#).

### 17.285.3.6 gfxbitmap\_font\_text\_scale()

```
void gfxbitmap_font_text_scale (
 void * fb,
 l4re_video_view_info_t * vi,
 gfxbitmap_font_t font,
 const char * text,
 unsigned len,
 unsigned x,
 unsigned y,
 gfxbitmap_color_pix_t fg,
 gfxbitmap_color_pix_t bg,
 int scale_x,
 int scale_y)
```

Render a string to a framebuffer, including scaling.

#### Parameters

|                |                                          |
|----------------|------------------------------------------|
| <i>fb</i>      | Pointer to frame buffer.                 |
| <i>vi</i>      | Frame buffer info structure.             |
| <i>font</i>    | Font.                                    |
| <i>text</i>    | Text string.                             |
| <i>len</i>     | Length of the text string.               |
| <i>x</i>       | Horizontal position in the frame buffer. |
| <i>y</i>       | Vertical position in the frame buffer.   |
| <i>fg</i>      | Foreground color.                        |
| <i>bg</i>      | Background color.                        |
| <i>scale_x</i> | Horizontal scale factor.                 |
| <i>scale_y</i> | Vertical scale factor.                   |

References [gfxbitmap\\_font\\_t](#), and [L4\\_END\\_DECLS](#).

### 17.285.3.7 gfxbitmap\_font\_width()

```
unsigned gfxbitmap_font_width (
 gfxbitmap_font_t font)
```

Get the font width.

#### Parameters

|             |       |
|-------------|-------|
| <i>font</i> | Font. |
|-------------|-------|

#### Returns

Font width, 0 if font width could not be retrieved.

References [gfxbitmap\\_font\\_t](#), and [L4\\_CV](#).

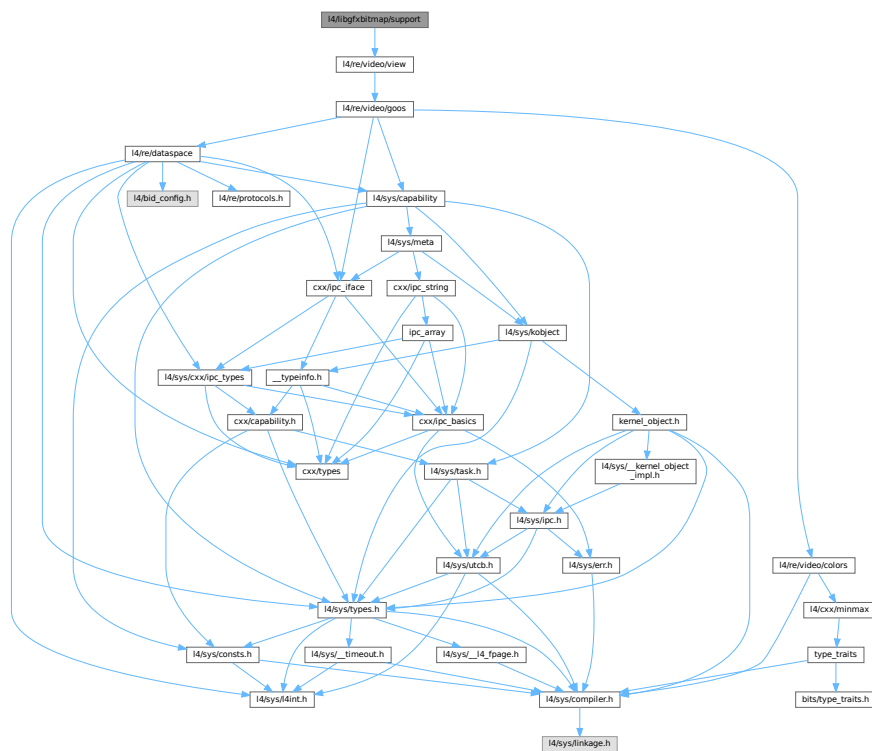
## 17.286 font.h

[Go to the documentation of this file.](#)

```
00001
00005 /*
00006 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 * This file is part of TUD:OS and distributed under the terms of the
00009 * GNU Lesser General Public License 2.1.
00010 * Please see the COPYING-LGPL-2.1 file for details.
00011 */
00012 #pragma once
00013
00014 #include <l4/sys/compiler.h>
00015 #include <l4/re/c/video/view.h>
00016 #include <l4/libgfxbitmap/bitmap.h>
00017
00022
00027
00031 #define GFXBITMAP_DEFAULT_FONT (void *)0
00032
00038 enum { GFXBITMAP_USE_STRLEN = ~0U };
00039
00040 L4_BEGIN_DECLS
00041
00043 typedef void *gfxbitmap_font_t;
00044
00053 L4_CV int gfxbitmap_font_init(void);
00054
00062 L4_CV gfxbitmap_font_t gfxbitmap_font_get(const char *name);
00063
00070 L4_CV unsigned
00071 gfxbitmap_font_width(gfxbitmap_font_t font);
00072
00079 L4_CV unsigned
00080 gfxbitmap_font_height(gfxbitmap_font_t font);
00081
00089 L4_CV void *
00090 gfxbitmap_font_data(gfxbitmap_font_t font, unsigned c);
00091
00105 L4_CV void
00106 gfxbitmap_font_text(void *fb, l4re_video_view_info_t *vi,
00107 gfxbitmap_font_t font, const char *text, unsigned len,
00108 unsigned x, unsigned y,
00109 gfxbitmap_color_pix_t fg, gfxbitmap_color_pix_t bg);
00110
00126 L4_CV void
00127 gfxbitmap_font_text_scale(void *fb, l4re_video_view_info_t *vi,
00128 gfxbitmap_font_t font, const char *text, unsigned len,
00129 unsigned x, unsigned y,
00130 gfxbitmap_color_pix_t fg, gfxbitmap_color_pix_t bg,
00131 int scale_x, int scale_y);
00132 L4_END_DECLS
```

Terminal support functionality.

Include dependency graph for support:



Terminal support functionality.

2009

Adam Lackorzynski [adam@os.inf.tu-dresden.de](mailto:adam@os.inf.tu-dresden.de)

Definition in file [support](#).

## 17.288 support

[Go to the documentation of this file.](#)

```

00001 /* vim:set ft=cpp: */
00009 /*
00010 * (c) 2009 Author(s)
00011 * economic rights: Technische Universität Dresden (Germany)
00012 * This file is part of TUD:OS and distributed under the terms of the
00013 * GNU Lesser General Public License 2.1.
00014 * Please see the COPYING-LGPL-2.1 file for details.
00015 */
00016 #ifndef __LIBTERM_SUPPORT_H__
00017 #define __LIBTERM_SUPPORT_H__
00018
00019 #include <l4/re/video/view>
00020
00021 void
00022 libterm_init_colors(L4Re::Video::View::Info *fbi);
00023
00024 int
00025 libterm_get_color(int mode, int color);
00026
00027 #endif

```

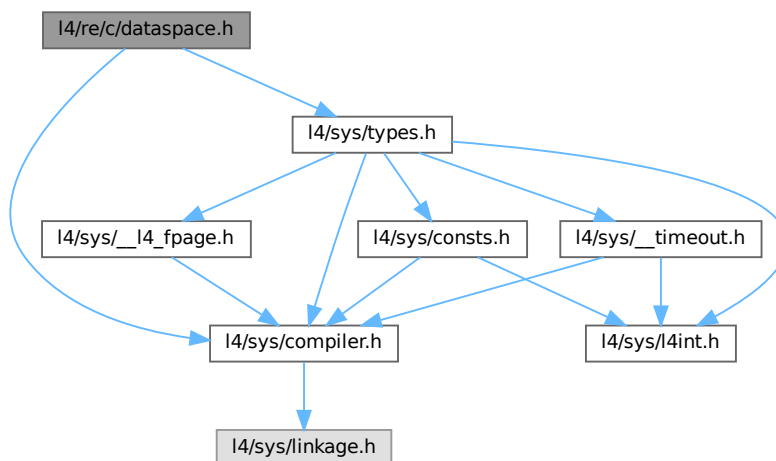
## 17.289 l4/re/c/dataspace.h File Reference

Data space C interface.

```
#include <l4/sys/compiler.h>
```

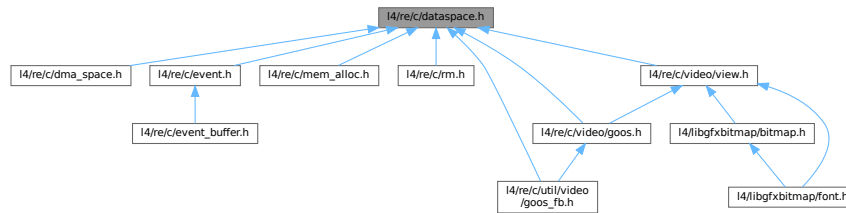
```
#include <l4/sys/types.h>
```

Include dependency graph for dataspace.h:





This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [I4re\\_ds\\_stats\\_t](#)  
*Information about the data space.*

## Enumerations

- enum [I4re\\_ds\\_map\\_flags](#) { }  
*Flags to specify the memory mapping type of a request.*

## Functions

- long [I4re\\_ds\\_clear](#) ([I4re\\_ds\\_t](#) ds, [I4re\\_ds\\_offset\\_t](#) offset, [I4re\\_ds\\_size\\_t](#) size) [L4\\_NOTHROW](#)  
*Clear parts of a dataspace.*
- long [I4re\\_ds\\_allocate](#) ([I4re\\_ds\\_t](#) ds, [I4re\\_ds\\_offset\\_t](#) offset, [I4re\\_ds\\_size\\_t](#) size) [L4\\_NOTHROW](#)  
*Allocate a range in the dataspace.*
- int [I4re\\_ds\\_copy\\_in](#) ([I4re\\_ds\\_t](#) ds, [I4re\\_ds\\_offset\\_t](#) dst\_offs, [I4re\\_ds\\_t](#) src, [I4re\\_ds\\_offset\\_t](#) src\_offs, [I4re\\_ds\\_size\\_t](#) size) [L4\\_NOTHROW](#)  
*Copy contents from another dataspace.*
- [I4re\\_ds\\_size\\_t](#) [I4re\\_ds\\_size](#) ([I4re\\_ds\\_t](#) ds) [L4\\_NOTHROW](#)  
*Get size of a dataspace.*
- [I4re\\_ds\\_flags\\_t](#) [I4re\\_ds\\_flags](#) ([I4re\\_ds\\_t](#) ds) [L4\\_NOTHROW](#)  
*Get flags of the dataspace.*
- int [I4re\\_ds\\_info](#) ([I4re\\_ds\\_t](#) ds, [I4re\\_ds\\_stats\\_t](#) \*stats) [L4\\_NOTHROW](#)  
*Get information on the dataspace.*
- int [I4re\\_ds\\_map\\_info](#) ([I4re\\_ds\\_t](#) ds, [I4\\_addr\\_t](#) \*start\_addr, [I4\\_addr\\_t](#) \*end\_addr) [L4\\_NOTHROW](#)  
*Get mapping range of dataspace.*

## Variables

- [L4\\_BEGIN\\_DECLS](#) typedef [I4\\_cap\\_idx\\_t](#) [I4re\\_ds\\_t](#)  
*Dataspace type.*

## 17.289.1 Detailed Description

Data space C interface.

Definition in file [dataspace.h](#).

## 17.290 dataspace.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00019
00020 #include <l4/sys/compiler.h>
00021 #include <l4/sys/types.h>
00022
00023 L4_BEGIN_DECLS
00024
00029 typedef l4_cap_idx_t l4re_ds_t;
00030 typedef l4_uint64_t l4re_ds_size_t;
00031 typedef l4_uint64_t l4re_ds_offset_t;
00032 typedef l4_uint64_t l4re_ds_map_addr_t;
00033 typedef unsigned long l4re_ds_flags_t;
00034
00039 typedef struct {
00040 l4re_ds_size_t size;
00041 l4re_ds_flags_t flags;
00042 } l4re_ds_stats_t;
00043
00048 enum l4re_ds_map_flags {
00049 L4RE_DS_F_R = L4_FPAGE_RO,
00050 L4RE_DS_F_W = L4_FPAGE_W,
00051 L4RE_DS_F_X = L4_FPAGE_X,
00052 L4RE_DS_F_RW = L4_FPAGE_RW,
00053 L4RE_DS_F_RX = L4_FPAGE_RX,
00054 L4RE_DS_F_RWX = L4_FPAGE_RWX,
00055
00056 L4RE_DS_F_RIGHTS_MASK = 0x0f,
00057
00058 L4RE_DS_F_NORMAL = 0x00,
00059 L4RE_DS_F_CACHEABLE = L4RE_DS_F_NORMAL,
00060 L4RE_DS_F_BUFFERABLE = 0x10,
00061 L4RE_DS_F_UNCACHEABLE = 0x20,
00062 L4RE_DS_F_CACHING_MASK = 0x30,
00063 L4RE_DS_F_CACHING_SHIFT = 4,
00064 };
00065
00071 L4_CV int
00072 l4re_ds_map(l4re_ds_t ds,
00073 l4re_ds_offset_t offset,
00074 l4re_ds_flags_t flags,
00075 l4re_ds_map_addr_t local_addr,
00076 l4re_ds_map_addr_t min_addr,
00077 l4re_ds_map_addr_t max_addr) L4_NOTHROW;
00078
00084 L4_CV int
00085 l4re_ds_map_region(l4re_ds_t ds,
00086 l4re_ds_offset_t offset,
00087 l4re_ds_flags_t flags,
00088 l4re_ds_map_addr_t min_addr,
00089 l4re_ds_map_addr_t max_addr) L4_NOTHROW;
00090
00097 L4_CV long
00098 l4re_ds_clear(l4re_ds_t ds, l4re_ds_offset_t offset,
00099 l4re_ds_size_t size) L4_NOTHROW;
00100
00107 L4_CV long
00108 l4re_ds_allocate(l4re_ds_t ds,
00109 l4re_ds_offset_t offset,
00110 l4re_ds_size_t size) L4_NOTHROW;
00111
00118 L4_CV int
00119 l4re_ds_copy_in(l4re_ds_t ds, l4re_ds_offset_t dst_offs,
00120 l4re_ds_t src, l4re_ds_offset_t src_offs,
00121 l4re_ds_size_t size) L4_NOTHROW;
00122
00129 L4_CV l4re_ds_size_t
00130 l4re_ds_size(l4re_ds_t ds) L4_NOTHROW;
00131
00138 L4_CV l4re_ds_flags_t
00139 l4re_ds_flags(l4re_ds_t ds) L4_NOTHROW;
00140
00147 L4_CV int
00148 l4re_ds_info(l4re_ds_t ds, l4re_ds_stats_t *stats) L4_NOTHROW;

```

```

00149
00156 L4_CV int
00157 l4re_ds_map_info(l4re_ds_t ds,
00158 l4_addr_t *start_addr, l4_addr_t *end_addr) L4_NOTHROW;
00159
00160 L4_END_DECLS

```

## 17.291 l4/re/c/dma\_space.h File Reference

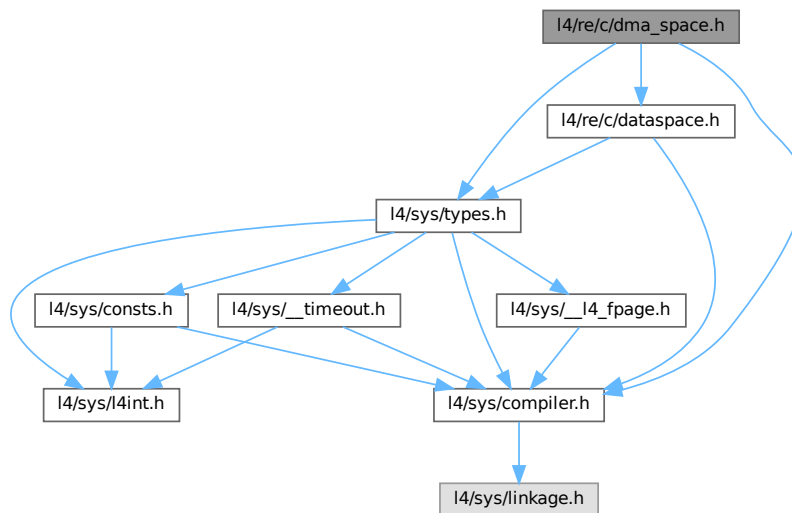
DMA space C interface.

```

#include <l4/sys/compiler.h>
#include <l4/sys/types.h>
#include <l4/re/c/dataspace.h>

```

Include dependency graph for dma\_space.h:



### Typedefs

- typedef `l4_cap_idx_t l4re_dma_space_t`  
DMA space capability type.
- typedef `l4_uint64_t l4re_dma_space_dma_addr_t`  
Data type for DMA addresses.

### Enumerations

- enum `l4re_dma_space_direction` { `L4RE_DMA_SPACE_BIDIRECTIONAL`, `L4RE_DMA_SPACE_TO_DEVICE`, `L4RE_DMA_SPACE_FROM_DEVICE`, `L4RE_DMA_SPACE_NONE` }  
Direction of the DMA transfers.
- enum `l4re_dma_space_space_attris` { `L4RE_DMA_SPACE_COHERENT` = 1 << 0, `L4RE_DMA_SPACE_PHYS_SPACE` = 1 << 1 }  
Attributes assigned to the DMA space when associated with a specific device.

## Functions

- long [l4re\\_dma\\_space\\_map](#) ([l4re\\_dma\\_space\\_t](#) dma, [l4re\\_ds\\_t](#) src, [l4re\\_ds\\_offset\\_t](#) offset, [l4\\_size\\_t](#) \*size, unsigned long attrs, enum [l4re\\_dma\\_space\\_direction](#) dir, [l4re\\_dma\\_space\\_dma\\_addr\\_t](#) \*dma\_addr) [L4\\_NOTHROW](#)  
*Map the given part of this data space into the DMA address space.*
- long [l4re\\_dma\\_space\\_unmap](#) ([l4re\\_dma\\_space\\_t](#) dma, [l4re\\_dma\\_space\\_dma\\_addr\\_t](#) dma\_addr, [l4\\_size\\_t](#) size, unsigned long attrs, enum [l4re\\_dma\\_space\\_direction](#) dir) [L4\\_NOTHROW](#)  
*Unmap the given part of this data space from the DMA address space.*
- long [l4re\\_dma\\_space\\_associate](#) ([l4re\\_dma\\_space\\_t](#) dma, [l4\\_cap\\_idx\\_t](#) dma\_task, unsigned long attr) [L4\\_NOTHROW](#)  
*Associate a (kernel) [DMA space](#) for a device to this [Dma\\_space](#).*
- long [l4re\\_dma\\_space\\_disassociate](#) ([l4re\\_dma\\_space\\_t](#) dma)  
*Disassociate the (kernel) [DMA space](#) from this [Dma\\_space](#).*

### 17.291.1 Detailed Description

DMA space C interface.

Definition in file [dma\\_space.h](#).

### 17.291.2 Enumeration Type Documentation

#### 17.291.2.1 l4re\_dma\_space\_direction

enum [l4re\\_dma\\_space\\_direction](#)

Direction of the DMA transfers.

#### Enumerator

|                                              |                                       |
|----------------------------------------------|---------------------------------------|
| <a href="#">L4RE_DMA_SPACE_BIDIRECTIONAL</a> | device reads and writes to the memory |
| <a href="#">L4RE_DMA_SPACE_TO_DEVICE</a>     | device reads the memory               |
| <a href="#">L4RE_DMA_SPACE_FROM_DEVICE</a>   | device writes to the memory           |
| <a href="#">L4RE_DMA_SPACE_NONE</a>          | device is coherently connected        |

Definition at line 27 of file [dma\\_space.h](#).

#### 17.291.2.2 l4re\_dma\_space\_space\_attrbs

enum [l4re\\_dma\\_space\\_space\\_attrbs](#)

Attributes assigned to the DMA space when associated with a specific device.

See also

[Space\\_attrbs](#)

#### Enumerator

|                           |                                                                                                                                               |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| L4RE_DMA_SPACE_COHERENT   | The device is connected coherently with the cache. This means that the map() and unmap() do not need to sync CPU caches before and after DMA. |
| L4RE_DMA_SPACE_PHYS_SPACE | The DMA space has no DMA task assigned and uses the CPUs physical memory.                                                                     |

Definition at line 38 of file [dma\\_space.h](#).

## 17.292 dma\_space.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2014 Alexander Warg <alexander.warg@kernkonzept.com>
00007 *
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010 #pragma once
00011
00017
00018 #include <l4/sys/compiler.h>
00019 #include <l4/sys/types.h>
00020 #include <l4/re/c/dataspace.h>
00021
00022 L4_BEGIN_DECLS
00023
00027 enum l4re_dma_space_direction
00028 {
00029 L4RE_DMA_SPACE_BIDIRECTIONAL,
00030 L4RE_DMA_SPACE_TO_DEVICE,
00031 L4RE_DMA_SPACE_FROM_DEVICE,
00032 L4RE_DMA_SPACE_NONE
00033 };
00034
00038 enum l4re_dma_space_space_attribs
00039 {
00040 L4RE_DMA_SPACE_COHERENT = 1 << 0,
00041 L4RE_DMA_SPACE_PHYS_SPACE = 1 << 1,
00042 };
00043
00049 typedef l4_cap_idx_t l4re_dma_space_t;
00050
00052 typedef l4_uint64_t l4re_dma_space_dma_addr_t;
00053
00060 L4_CV long
00061 l4re_dma_space_map(l4re_dma_space_t dma, l4re_ds_t src,
00062 l4re_ds_offset_t offset,
00063 l4_size_t * size, unsigned long attrs,
00064 enum l4re_dma_space_direction dir,
00065 l4re_dma_space_dma_addr_t *dma_addr) L4_NOTHROW;
00066
00067
00074 L4_CV long
00075 l4re_dma_space_unmap(l4re_dma_space_t dma, l4re_dma_space_dma_addr_t dma_addr,
00076 l4_size_t size, unsigned long attrs,
00077 enum l4re_dma_space_direction dir) L4_NOTHROW;
00078
00085 L4_CV long
00086 l4re_dma_space_associate(l4re_dma_space_t dma, l4_cap_idx_t dma_task,
00087 unsigned long attr) L4_NOTHROW;
00088
00095 L4_CV long
00096 l4re_dma_space_disassociate(l4re_dma_space_t dma);
00097
00098
00099 L4_END_DECLS

```

## 17.293 l4/re/c/event.h File Reference

Event C interface.



- `long l4re_event_get_stream_info` (const `l4_cap_idx_t` server, int idx, `l4re_event_stream_info_t` \*info) `L4_NOTHROW`  
*Get information on a stream.*
- `long l4re_event_get_stream_info_for_id` (const `l4_cap_idx_t` server, `l4_umword_t` stream\_id, `l4re_event_stream_info_t` \*info) `L4_NOTHROW`  
*Get info for a stream given a stream id.*
- `long l4re_event_get_axis_info` (const `l4_cap_idx_t` server, `l4_umword_t` id, unsigned naxes, unsigned const \*axis, `l4re_event_absinfo_t` \*info) `L4_NOTHROW`  
*Get Axis information for a stream.*

## 17.293.1 Detailed Description

Event C interface.

Definition in file [event.h](#).

## 17.294 event.h

[Go to the documentation of this file.](#)

```

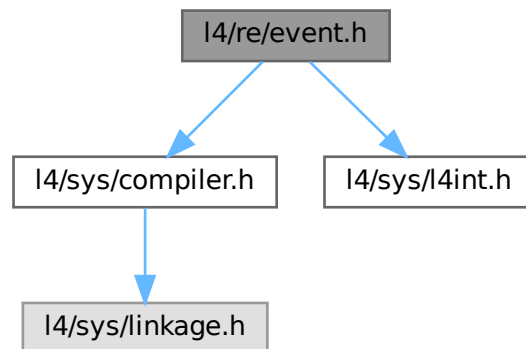
00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00019
00020 #include <l4/sys/compiler.h>
00021 #include <l4/sys/types.h>
00022 #include <l4/re/c/dataspace.h>
00023 #include <l4/re/event.h>
00024
00025 L4_BEGIN_DECLS
00026
00030 typedef struct
00031 {
00032 long long time;
00033 unsigned short type;
00034 unsigned short code;
00035 int value;
00036 l4_umword_t stream_id;
00037 } l4re_event_t;
00038
00050 L4_CV long
00051 l4re_event_get_buffer(const l4_cap_idx_t server,
00052 const l4re_ds_t ds) L4_NOTHROW;
00053
00064 L4_CV long
00065 l4re_event_get_num_streams(const l4_cap_idx_t server) L4_NOTHROW;
00066
00079 L4_CV long
00080 l4re_event_get_stream_info(const l4_cap_idx_t server,
00081 int idx, l4re_event_stream_info_t *info) L4_NOTHROW;
00082
00095 L4_CV long
00096 l4re_event_get_stream_info_for_id(const l4_cap_idx_t server,
00097 l4_umword_t stream_id,
00098 l4re_event_stream_info_t *info) L4_NOTHROW;
00099
00115 L4_CV long
00116 l4re_event_get_axis_info(const l4_cap_idx_t server, l4_umword_t id,
00117 unsigned naxes, unsigned const *axis,
00118 l4re_event_absinfo_t *info) L4_NOTHROW;
00119
00120 L4_END_DECLS

```

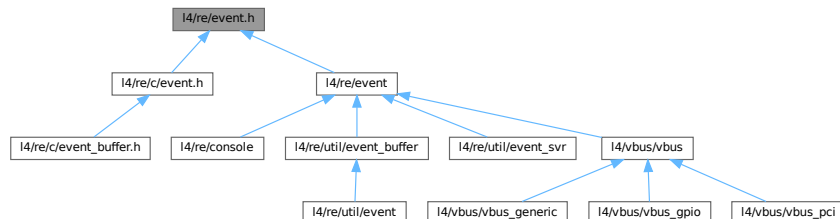
## 17.295 I4/re/event.h File Reference

Events.

```
#include <l4/sys/compiler.h>
#include <l4/sys/l4int.h>
Include dependency graph for event.h:
```



This graph shows which files directly or indirectly include this file:



### 17.295.1 Detailed Description

Events.

Definition in file [event.h](#).

## 17.296 event.h

[Go to the documentation of this file.](#)

```
00001
00005 /*
```



```

00006 * (c) 2009 Alexander Warg <warg@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #pragma once
00012
00013 #include <l4/sys/compiler.h>
00014 #include <l4/sys/l4int.h>
00015
00016 typedef struct L4_EXPORT_TYPE l4re_event_stream_id_t
00017 {
00018 l4_uint16_t bustype;
00019 l4_uint16_t vendor;
00020 l4_uint16_t product;
00021 l4_uint16_t version;
00022 } l4re_event_stream_id_t;
00023
00024 typedef struct L4_EXPORT_TYPE l4re_event_absinfo_t
00025 {
00026 l4_int32_t value;
00027 l4_int32_t min;
00028 l4_int32_t max;
00029 l4_int32_t fuzz;
00030 l4_int32_t flat;
00031 l4_int32_t resolution;
00032 } l4re_event_absinfo_t;
00033
00034 enum l4re_event_stream_max_values_t
00035 {
00036 L4RE_EVENT_EV_MAX = 0x1f,
00037 L4RE_EVENT_KEY_MAX = 0x1ff,
00038 L4RE_EVENT_REL_MAX = 0xf,
00039 L4RE_EVENT_ABS_MAX = 0x3f,
00040 L4RE_EVENT_PROP_MAX = 0x1f,
00041 L4RE_EVENT_SW_MAX = 0xf, // should be >= L4RE_SW_MAX
00042 };
00043
00044 enum l4re_event_stream_props_t
00045 {
00046 L4RE_EVENT_STREAM_CALIBRATE = 0x001,
00047 };
00048
00049
00050 #define __UNUM_B(x) ((x+1) + sizeof(unsigned long)*8 - 1) / (sizeof(unsigned long)*8)
00051
00052 typedef struct L4_EXPORT_TYPE l4re_event_stream_info_t
00053 {
00054 l4_umword_t stream_id;
00055 char name[32];
00056 char phys[32];
00057 l4re_event_stream_id_t id;
00058
00059 unsigned long propbits[__UNUM_B(L4RE_EVENT_PROP_MAX)];
00060
00061 unsigned long evbits[__UNUM_B(L4RE_EVENT_EV_MAX)];
00062 unsigned long keybits[__UNUM_B(L4RE_EVENT_KEY_MAX)];
00063 unsigned long relbits[__UNUM_B(L4RE_EVENT_REL_MAX)];
00064 unsigned long absbits[__UNUM_B(L4RE_EVENT_ABS_MAX)];
00065 unsigned long swbits[__UNUM_B(L4RE_EVENT_SW_MAX)];
00066 } l4re_event_stream_info_t;
00067
00068
00069 typedef struct L4_EXPORT_TYPE l4re_event_stream_state_t
00070 {
00071 unsigned long keybits[__UNUM_B(L4RE_EVENT_KEY_MAX)];
00072 unsigned long swbits[__UNUM_B(L4RE_EVENT_SW_MAX)];
00073 } l4re_event_stream_state_t;
00074
00075 #undef __UNUM_B
00076

```

## 17.297 event\_buffer.h

```

00001 #pragma once
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00009 #include <l4/sys/compiler.h>

```

```

00010 #include <l4/sys/linkage.h>
00011 #include <l4/re/c/event.h>
00012
00013 L4_BEGIN_DECLS
00014
00015 typedef struct l4re_event_buffer_consumer_t
00016 {
00017 unsigned long _obj_buf[8];
00018 } l4re_event_buffer_consumer_t;
00019
00020 L4_CV void
00021 l4re_event_free(l4re_event_t *e) L4_NOTHROW;
00022
00023 L4_CV long
00024 l4re_event_buffer_attach(l4re_event_buffer_consumer_t *evbuf,
00025 l4re_ds_t ds, l4_cap_idx_t rm) L4_NOTHROW;
00026
00027 L4_CV long
00028 l4re_event_buffer_detach(l4re_event_buffer_consumer_t *evbuf,
00029 l4_cap_idx_t rm) L4_NOTHROW;
00030
00031 L4_CV l4re_event_t *
00032 l4re_event_buffer_next(l4re_event_buffer_consumer_t *evbuf) L4_NOTHROW;
00033
00034 typedef L4_CV void l4re_event_buffer_cb_t(l4re_event_t *ev, void *data);
00035
00036 L4_CV void
00037 l4re_event_buffer_consumer_foreach_available_event(l4re_event_buffer_consumer_t *evbuf,
00038 void *data, l4re_event_buffer_cb_t *cb);
00039
00040
00041 L4_CV void
00042 l4re_event_buffer_consumer_process(l4re_event_buffer_consumer_t *evbuf,
00043 l4_cap_idx_t irq, l4_cap_idx_t thread, void *data,
00044 l4re_event_buffer_cb_t *cb);
00045
00046 L4_END_DECLS

```

## 17.298 l4/re/c/inhibitor.h File Reference

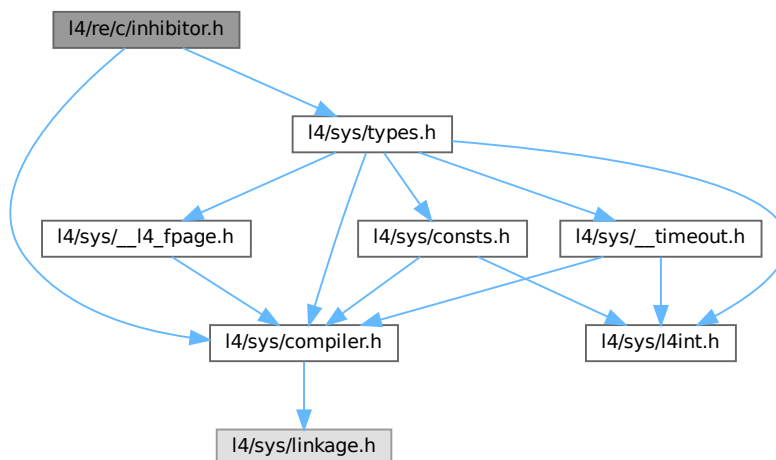
Inhibitor C interface.

```

#include <l4/sys/compiler.h>
#include <l4/sys/types.h>

```

Include dependency graph for inhibitor.h:



## Functions

- [L4\\_BEGIN\\_DECLS](#) long [l4re\\_inhibitor\\_acquire](#) ([l4\\_cap\\_idx\\_t](#) cap, [l4\\_umword\\_t](#) id, char const \*reason)  
*Acquire an inhibitor lock.*
- long [l4re\\_inhibitor\\_release](#) ([l4\\_cap\\_idx\\_t](#) cap, [l4\\_umword\\_t](#) id)  
*Release an inhibitor lock.*
- long [l4re\\_inhibitor\\_next\\_lock\\_info](#) ([l4\\_cap\\_idx\\_t](#) cap, char \*name, unsigned len, [l4\\_mword\\_t](#) current\_id)  
*Get information for the next available inhibitor lock.*

## 17.298.1 Detailed Description

Inhibitor C interface.

Definition in file [inhibitor.h](#).

## 17.298.2 Function Documentation

### 17.298.2.1 l4re\_inhibitor\_acquire()

```
L4_BEGIN_DECLS long l4re_inhibitor_acquire (
 l4_cap_idx_t cap,
 l4_umword_t id,
 char const * reason)
```

Acquire an inhibitor lock.

#### Parameters

|               |                                                                                        |
|---------------|----------------------------------------------------------------------------------------|
| <i>cap</i>    | Capability for the Inhibitor object (see <a href="#">L4Re::Inhibitor</a> )             |
| <i>id</i>     | ID of the inhibitor lock that shall be acquired.                                       |
| <i>reason</i> | Reason why the inhibitor lock is acquired. (Used for informing the user or debugging.) |

#### Returns

0 for success, <0 on error.

#### See also

[L4Re::Inhibitor::acquire\(\)](#).

References [L4\\_CV](#), and [L4\\_EXPORT](#).

### 17.298.2.2 l4re\_inhibitor\_next\_lock\_info()

```
long l4re_inhibitor_next_lock_info (
 l4_cap_idx_t cap,
 char * name,
 unsigned len,
 l4_mword_t current_id)
```

Get information for the next available inhibitor lock.

#### Parameters

|                         |                                                                                                 |
|-------------------------|-------------------------------------------------------------------------------------------------|
| <i>cap</i>              | Capability to the Inhibitor object (see <a href="#">L4Re::Inhibitor</a> )                       |
| <i>name</i>             | A pointer to a buffer for the name of the lock.                                                 |
| <i>len</i>              | The length of the available buffer (usually <a href="#">L4Re::Inhibitor::Name_max</a> is used). |
| <i>current↔<br/>_id</i> | The ID of the last available lock, use -1 to get the first lock.                                |

### Return values

|                   |                                                                                                                     |
|-------------------|---------------------------------------------------------------------------------------------------------------------|
| <i>&gt;0</i>      | The ID of the next available lock if there is one (in this case name shall contain the name of the inhibitor lock). |
| <i>-L4_ENODEV</i> | if there are no more locks.                                                                                         |
| <i>&lt;0</i>      | Any other negative failure value.                                                                                   |

### See also

[L4Re::Inhibitor::next\\_lock\\_info\(\)](#).

References [L4\\_END\\_DECLS](#).

### 17.298.2.3 l4re\_inhibitor\_release()

```
long l4re_inhibitor_release (
 l4_cap_idx_t cap,
 l4_umword_t id)
```

Release an inhibitor lock.

### Parameters

|            |                                                                             |
|------------|-----------------------------------------------------------------------------|
| <i>cap</i> | Capability for the Inhibitor object (see <a href="#">L4Re::Inhibitor</a> ). |
| <i>id</i>  | ID of inhibitor that shall be released.                                     |

### Returns

0 for success, <0 on error.

### See also

[L4Re::Inhibitor::release\(\)](#).

References [L4\\_CV](#), and [L4\\_EXPORT](#).

## 17.299 inhibitor.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * (c) 2014 Steffen Liebergeld <steffen.liebergeld@kernkonzept.com>
00003 *
00004 * License: see LICENSE.spdx (in this directory or the directories above)
00005 */
00006 #pragma once
00007
00013 #include <l4/sys/compiler.h>
00014 #include <l4/sys/types.h>
00015
00016 L4_BEGIN_DECLS
00017
00028 L4_CV long L4_EXPORT
00029 l4re_inhibitor_acquire(l4_cap_idx_t cap, l4_umword_t id,
00030 char const *reason);
00031
00039 L4_CV long L4_EXPORT
00040 l4re_inhibitor_release(l4_cap_idx_t cap, l4_umword_t id);
00041
00057 L4_CV long L4_EXPORT
00058 l4re_inhibitor_next_lock_info(l4_cap_idx_t cap, char *name,
00059 unsigned len, l4_mword_t current_id);
00060
00061 L4_END_DECLS
00062

```

## 17.300 l4/re/c/log.h File Reference

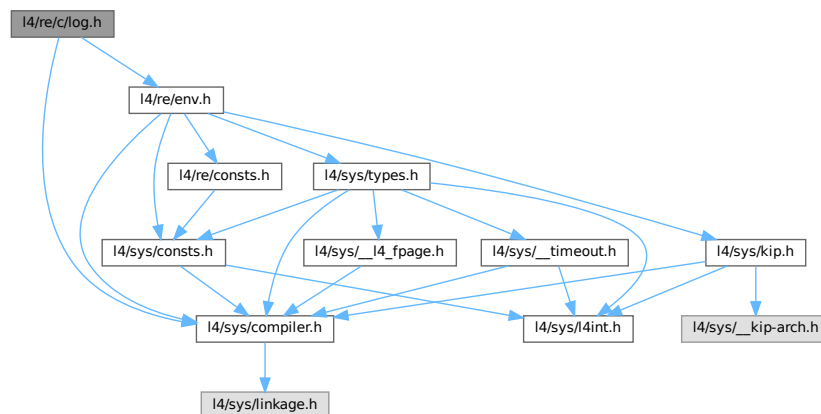
Log C interface.

```

#include <l4/re/env.h>
#include <l4/sys/compiler.h>

```

Include dependency graph for log.h:



### Functions

- `L4_BEGIN_DECLS` void `l4re_log_print` (char const \*string) `L4_NOTHROW`  
Write a null terminated string to the default log.
- void `l4re_log_printn` (char const \*string, int len) `L4_NOTHROW`  
Write a string of a given length to the default log.

- void `l4re_log_print_srv` (const `l4_cap_idx_t` logcap, char const \*string) `L4_NOTHROW`  
Write a null terminated string to a log.
- void `l4re_log_printn_srv` (const `l4_cap_idx_t` logcap, char const \*string, int len) `L4_NOTHROW`  
Write a string of a given length to a log.

## 17.300.1 Detailed Description

Log C interface.

Definition in file [log.h](#).

## 17.301 log.h

[Go to the documentation of this file.](#)

```

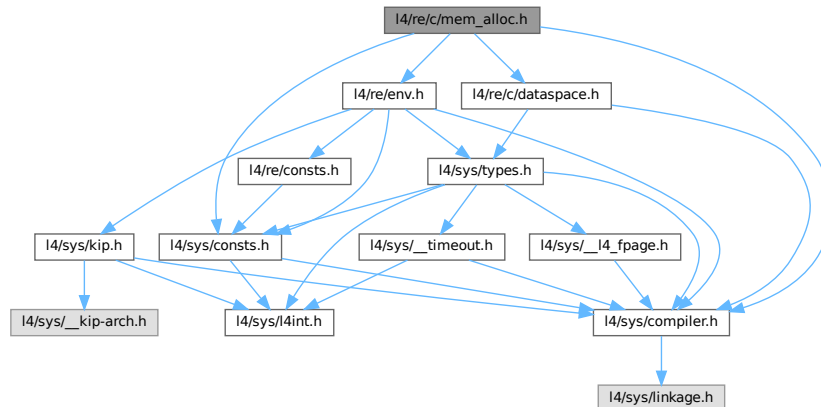
00001
00005 /*
00006 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #pragma once
00012
00018
00019 #include <l4/re/env.h>
00020 #include <l4/sys/compiler.h>
00021
00022 L4_BEGIN_DECLS
00023
00032 L4_CV L4_INLINE void
00033 l4re_log_print(char const *string) L4_NOTHROW;
00034
00044 L4_CV L4_INLINE void
00045 l4re_log_printn(char const *string, int len) L4_NOTHROW;
00046
00047
00048
00049
00059 L4_CV void
00060 l4re_log_print_srv(const l4_cap_idx_t logcap,
00061 char const *string) L4_NOTHROW;
00062
00073 L4_CV void
00074 l4re_log_printn_srv(const l4_cap_idx_t logcap,
00075 char const *string, int len) L4_NOTHROW;
00076
00077
00078 /***** Implementations *****/
00079
00080 L4_CV L4_INLINE void
00081 l4re_log_print(char const *string) L4_NOTHROW
00082 {
00083 l4re_log_print_srv(l4re_global_env->log, string);
00084 }
00085
00086 L4_CV L4_INLINE void
00087 l4re_log_printn(char const *string, int len) L4_NOTHROW
00088 {
00089 l4re_log_printn_srv(l4re_global_env->log, string, len);
00090 }
00091
00092 L4_END_DECLS

```

## 17.302 l4/re/c/mem\_alloc.h File Reference

Memory allocator C interface.

```
#include <l4/re/env.h>
#include <l4/sys/compiler.h>
#include <l4/sys/consts.h>
#include <l4/re/c/dataspace.h>
Include dependency graph for mem_alloc.h:
```



### Enumerations

- enum [l4re\\_ma\\_flags](#)  
*Flags for requesting memory at the memory allocator.*

### Functions

- long [l4re\\_ma\\_alloc](#) (long size, [l4re\\_ds\\_t](#) const mem, unsigned long flags) [L4\\_NOTHROW](#)  
*Allocate memory.*
- long [l4re\\_ma\\_alloc\\_align](#) (long size, [l4re\\_ds\\_t](#) const mem, unsigned long flags, unsigned long align) [L4\\_NOTHROW](#)  
*Allocate memory.*
- long [l4re\\_ma\\_alloc\\_align\\_srv](#) ([l4\\_cap\\_idx\\_t](#) srv, long size, [l4re\\_ds\\_t](#) const mem, unsigned long flags, unsigned long align) [L4\\_NOTHROW](#)  
*Allocate memory.*

### 17.302.1 Detailed Description

Memory allocator C interface.

Definition in file [mem\\_alloc.h](#).

## 17.303 mem\_alloc.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #pragma once
00012
00013 #include <l4/re/env.h>
00014 #include <l4/sys/compiler.h>
00015 #include <l4/sys/consts.h>
00016
00017 #include <l4/re/c/dataspace.h>
00018
00024
00025 L4_BEGIN_DECLS
00026
00032 enum l4re_ma_flags {
00033 L4RE_MA_CONTINUOUS = 0x01,
00034 L4RE_MA_PINNED = 0x02,
00035 L4RE_MA_SUPER_PAGES = 0x04,
00036 };
00037
00038
00069 L4_CV L4_INLINE long
00070 l4re_ma_alloc(long size, l4re_ds_t const mem,
00071 unsigned long flags) L4_NOTHROW;
00072
00106 L4_CV L4_INLINE long
00107 l4re_ma_alloc_align(long size, l4re_ds_t const mem,
00108 unsigned long flags, unsigned long align) L4_NOTHROW;
00109
00128 L4_CV long
00129 l4re_ma_alloc_align_srv(l4_cap_idx_t srv, long size,
00130 l4re_ds_t const mem, unsigned long flags,
00131 unsigned long align) L4_NOTHROW;
00132
00133 /***** Implementation *****/
00134
00135 L4_CV L4_INLINE long
00136 l4re_ma_alloc(long size, l4re_ds_t const mem,
00137 unsigned long flags) L4_NOTHROW
00138 {
00139 return l4re_ma_alloc_align_srv(l4re_global_env->mem_alloc, size, mem,
00140 flags, 0);
00141 }
00142
00143 L4_CV L4_INLINE long
00144 l4re_ma_alloc_align(long size, l4re_ds_t const mem,
00145 unsigned long flags, unsigned long align) L4_NOTHROW
00146 {
00147 return l4re_ma_alloc_align_srv(l4re_global_env->mem_alloc, size, mem,
00148 flags, align);
00149 }
00150
00151 L4_END_DECLS

```

## 17.304 l4/re/c/namespace.h File Reference

Namespace functions, C interface.

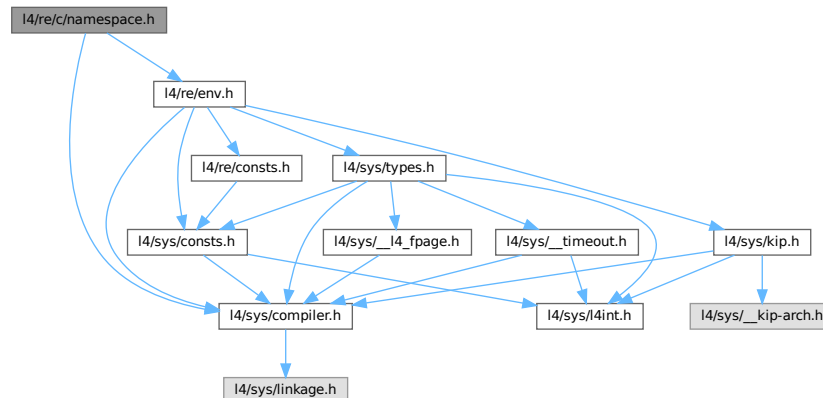
```

#include <l4/re/env.h>
#include <l4/sys/compiler.h>

```



Include dependency graph for namespace.h:



## Enumerations

- enum [l4re\\_ns\\_register\\_flags](#)  
Namespace register flags.

## Functions

- long [l4re\\_ns\\_query\\_to\\_srv](#) ([l4re\\_namespace\\_t](#) srv, char const \*name, [l4\\_cap\\_idx\\_t](#) const cap, int timeout) [L4\\_NOTHROW](#)  
Query the name space for the object named by *name*.
- long [l4re\\_ns\\_query\\_srv](#) ([l4re\\_namespace\\_t](#) srv, char const \*name, [l4\\_cap\\_idx\\_t](#) const cap) [L4\\_NOTHROW](#)  
Query the name space for the object named by *name*.
- long [l4re\\_ns\\_register\\_obj\\_srv](#) ([l4re\\_namespace\\_t](#) srv, char const \*name, [l4\\_cap\\_idx\\_t](#) const obj, unsigned flags) [L4\\_NOTHROW](#)  
Register an object with a name.

## Variables

- [L4\\_BEGIN\\_DECLS](#) typedef [l4\\_cap\\_idx\\_t](#) [l4re\\_namespace\\_t](#)  
Namespace type.

### 17.304.1 Detailed Description

Namespace functions, C interface.

Definition in file [namespace.h](#).

## 17.305 namespace.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00019
00020 #include <l4/re/env.h>
00021 #include <l4/sys/compiler.h>
00022
00029 enum l4re_ns_register_flags {
00030 L4RE_NS_REGISTER_RO = L4_FPAGE_RO,
00031 L4RE_NS_REGISTER_DIR = 0x10,
00032 L4RE_NS_REGISTER_RW = L4_FPAGE_RX,
00033 L4RE_NS_REGISTER_RWS = L4_FPAGE_RWX,
00034 L4RE_NS_REGISTER_S = L4_FPAGE_W,
00035 };
00036
00037 L4_BEGIN_DECLS
00038
00043 typedef l4_cap_idx_t l4re_namespace_t;
00044
00045
00046
00055 L4_CV long
00056 l4re_ns_query_to_srv(l4re_namespace_t srv, char const *name,
00057 l4_cap_idx_t const cap, int timeout) L4_NOTHROW;
00058
00075 L4_CV L4_INLINE long
00076 l4re_ns_query_srv(l4re_namespace_t srv, char const *name,
00077 l4_cap_idx_t const cap) L4_NOTHROW;
00078
00086 L4_CV long
00087 l4re_ns_register_obj_srv(l4re_namespace_t srv, char const *name,
00088 l4_cap_idx_t const obj, unsigned flags) L4_NOTHROW;
00089
00090
00091
00092 /***** Implementation *****/
00093
00094 L4_CV L4_INLINE long
00095 l4re_ns_query_srv(l4re_namespace_t srv, char const *name,
00096 l4_cap_idx_t const cap) L4_NOTHROW
00097 {
00098 return l4re_ns_query_to_srv(srv, name, cap, 40000);
00099 }
00100
00101
00102 L4_END_DECLS

```

## 17.306 l4/re/c/parent.h File Reference

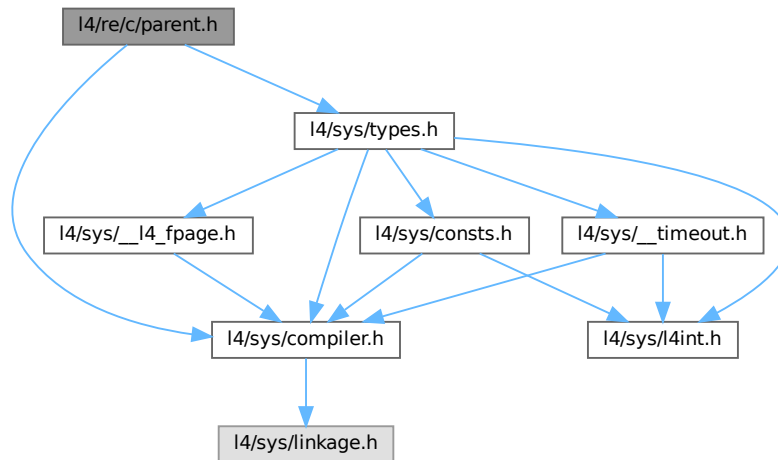
Parent C interface.

```

#include <l4/sys/compiler.h>
#include <l4/sys/types.h>

```

Include dependency graph for parent.h:



## Functions

- [L4\\_BEGIN\\_DECLS](#) long [l4re\\_parent\\_signal](#) ([l4\\_cap\\_idx\\_t](#) parent, unsigned long sig, unsigned long val)  
Send a signal using the parent protocol.

## 17.306.1 Detailed Description

Parent C interface.

Definition in file [parent.h](#).

## 17.306.2 Function Documentation

### 17.306.2.1 l4re\_parent\_signal()

```

L4_BEGIN_DECLS long l4re_parent_signal (
 l4_cap_idx_t parent,
 unsigned long sig,
 unsigned long val)

```

Send a signal using the parent protocol.

## Parameters

|               |                                               |
|---------------|-----------------------------------------------|
| <i>parent</i> | Capability index of parent to send signal to. |
| <i>sig</i>    | Signal to send                                |
| <i>val</i>    | Value of the signal                           |

## Return values

|    |           |
|----|-----------|
| 0  | Success   |
| <0 | IPC error |

See also

[L4Re::Parent::signal](#)

References [L4\\_END\\_DECLS](#).

## 17.307 parent.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * Copyright (C) 2024 Kernkonzept GmbH.
00003 * Author(s): Marcus Haehnel <marcus.haehnel@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00012
00013 #pragma once
00014
00019
00020 #include <l4/sys/compiler.h>
00021 #include <l4/sys/types.h>
00022
00023 L4_BEGIN_DECLS
00024
00037 L4_CV long L4_EXPORT
00038 l4re_parent_signal(l4_cap_idx_t parent, unsigned long sig, unsigned long val);
00039
00040 L4_END_DECLS

```

## 17.308 l4/re/c/rm.h File Reference

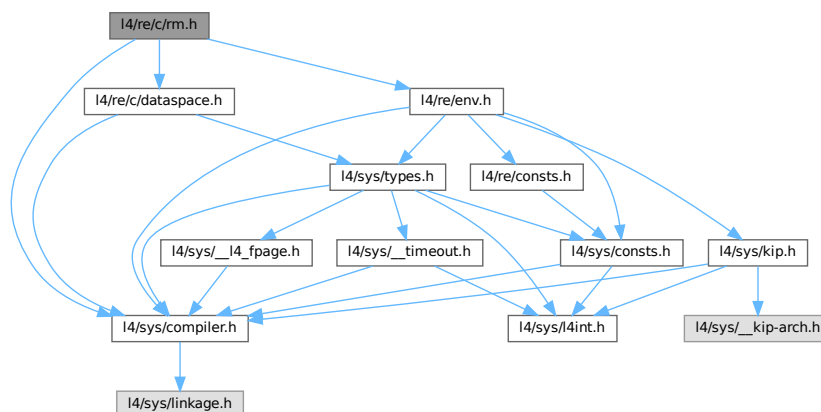
Region map interface, C interface.

```

#include <l4/re/env.h>
#include <l4/re/c/dataspace.h>
#include <l4/sys/compiler.h>

```

Include dependency graph for rm.h:



## Enumerations

- enum `l4re_rm_flags_values` {  
`L4RE_RM_F_R` = `L4RE_DS_F_R` , `L4RE_RM_F_W` = `L4RE_DS_F_W` , `L4RE_RM_F_X` = `L4RE_DS_F_X`  
, `L4RE_RM_F_RX` = `L4RE_DS_F_RX` ,  
`L4RE_RM_F_RW` = `L4RE_DS_F_RW` , `L4RE_RM_F_RWX` = `L4RE_DS_F_RWX` , `L4RE_RM_F_KERNEL`  
= `0x100` , `L4RE_RM_F_DETACH_FREE` = `0x200` ,  
`L4RE_RM_F_PAGER` = `0x400` , `L4RE_RM_F_RESERVED` = `0x800` , `L4RE_RM_CACHING_SHIFT` = `4` ,  
`L4RE_RM_F_CACHING` = `L4RE_DS_F_CACHING_MASK` ,  
`L4RE_RM_REGION_FLAGS` = `0xffff` , `L4RE_RM_F_CACHE_NORMAL` = `L4RE_DS_F_NORMAL` ,  
`L4RE_RM_F_CACHE_BUFFERED` = `L4RE_DS_F_BUFFERABLE` , `L4RE_RM_F_CACHE_UNCACHED`  
= `L4RE_DS_F_UNCACHABLE` ,  
`L4RE_RM_F_SEARCH_ADDR` = `0x020000` , `L4RE_RM_F_IN_AREA` = `0x040000` , `L4RE_RM_F_EAGER_MAP`  
= `0x080000` , `L4RE_RM_F_NO_EAGER_MAP` = `0x100000` ,  
`L4RE_RM_F_ATTACH_FLAGS` = `0x1f0000` }

*Flags for region operations.*

## Functions

- int `l4re_rm_reserve_area` (`l4_addr_t` \*start, unsigned long size, `l4re_rm_flags_t` flags, unsigned char align)  
`L4_NOTHROW`  
*Reserve the given area in the region map.*
- int `l4re_rm_free_area` (`l4_addr_t` addr) `L4_NOTHROW`  
*Free an area from the region map.*
- int `l4re_rm_attach` (void \*\*start, unsigned long size, `l4re_rm_flags_t` flags, `l4re_ds_t` mem, `l4re_rm_offset_t` offs, unsigned char align) `L4_NOTHROW`  
*Attach a data space to a region.*
- int `l4re_rm_detach` (void \*addr) `L4_NOTHROW`  
*Detach and unmap a region from the address space in the current task.*
- int `l4re_rm_detach_ds` (void \*addr, `l4re_ds_t` \*ds) `L4_NOTHROW`  
*Detach and unmap a region and return affected dataspace in the current task.*
- int `l4re_rm_detach_unmap` (`l4_addr_t` addr, `l4_cap_idx_t` task) `L4_NOTHROW`  
*Detach and unmap in specified task.*
- int `l4re_rm_detach_ds_unmap` (void \*addr, `l4re_ds_t` \*ds, `l4_cap_idx_t` task) `L4_NOTHROW`  
*Detach and unmap in specified task.*
- int `l4re_rm_find` (`l4_addr_t` \*addr, unsigned long \*size, `l4re_rm_offset_t` \*offset, `l4re_rm_flags_t` \*flags, `l4re_ds_t` \*m) `L4_NOTHROW`  
*Find a region given an address and size.*
- int `l4re_rm_get_info` (`l4_addr_t` addr, char \*name, unsigned int len, `l4re_rm_offset_t` \*backing\_offset) `L4_NOTHROW`  
*Return auxiliary information of a region.*
- void `l4re_rm_show_lists` (void) `L4_NOTHROW`  
*Dump region map internal data structures.*
- int `l4re_rm_reserve_area_srv` (`l4_cap_idx_t` rm, `l4_addr_t` \*start, unsigned long size, `l4re_rm_flags_t` flags, unsigned char align) `L4_NOTHROW`
- int `l4re_rm_free_area_srv` (`l4_cap_idx_t` rm, `l4_addr_t` addr) `L4_NOTHROW`
- int `l4re_rm_attach_srv` (`l4_cap_idx_t` rm, void \*\*start, unsigned long size, `l4re_rm_flags_t` flags, `l4re_ds_t` mem, `l4re_rm_offset_t` offs, unsigned char align) `L4_NOTHROW`
- int `l4re_rm_detach_srv` (`l4_cap_idx_t` rm, `l4_addr_t` addr, `l4re_ds_t` \*ds, `l4_cap_idx_t` task) `L4_NOTHROW`
- int `l4re_rm_find_srv` (`l4_cap_idx_t` rm, `l4_addr_t` \*addr, unsigned long \*size, `l4re_rm_offset_t` \*offset, `l4re_rm_flags_t` \*flags, `l4re_ds_t` \*m) `L4_NOTHROW`
- int `l4re_rm_get_info_srv` (`l4_cap_idx_t` rm, `l4_addr_t` addr, char \*name, unsigned int len, `l4re_rm_offset_t` \*backing\_offset) `L4_NOTHROW`
- void `l4re_rm_show_lists_srv` (`l4_cap_idx_t` rm) `L4_NOTHROW`  
*Dump region map internal data structures.*

## 17.308.1 Detailed Description

Region map interface, C interface.

Definition in file [rm.h](#).

## 17.309 rm.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00019
00020 #include <l4/re/env.h>
00021 #include <l4/re/c/dataspace.h>
00022 #include <l4/sys/compiler.h>
00023
00024 L4_BEGIN_DECLS
00025
00030 enum l4re_rm_flags_values {
00031 L4RE_RM_F_R = L4RE_DS_F_R,
00032 L4RE_RM_F_W = L4RE_DS_F_W,
00033 L4RE_RM_F_X = L4RE_DS_F_X,
00034 L4RE_RM_F_RX = L4RE_DS_F_RX,
00035 L4RE_RM_F_RW = L4RE_DS_F_RW,
00036 L4RE_RM_F_RWX = L4RE_DS_F_RWX,
00037
00038 L4RE_RM_F_KERNEL = 0x100,
00039 L4RE_RM_F_DETACH_FREE = 0x200,
00040 L4RE_RM_F_PAGER = 0x400,
00041 L4RE_RM_F_RESERVED = 0x800,
00042
00043 L4RE_RM_CACHING_SHIFT = 4,
00044
00046 L4RE_RM_F_CACHING = L4RE_DS_F_CACHING_MASK,
00047
00048 L4RE_RM_REGION_FLAGS = 0xffff,
00049
00051 L4RE_RM_F_CACHE_NORMAL = L4RE_DS_F_NORMAL,
00052
00054 L4RE_RM_F_CACHE_BUFFERED = L4RE_DS_F_BUFFERABLE,
00055
00057 L4RE_RM_F_CACHE_UNCACHED = L4RE_DS_F_UNCACHEABLE,
00058
00059 L4RE_RM_F_SEARCH_ADDR = 0x020000,
00060 L4RE_RM_F_IN_AREA = 0x040000,
00061 L4RE_RM_F_EAGER_MAP = 0x080000,
00062 L4RE_RM_F_NO_EAGER_MAP = 0x100000,
00063 L4RE_RM_F_ATTACH_FLAGS = 0x1f0000,
00064 };
00065
00066 typedef l4_uint32_t l4re_rm_flags_t;
00067 typedef l4_uint64_t l4re_rm_offset_t;
00068
00077 L4_CV L4_INLINE int
00078 l4re_rm_reserve_area(l4_addr_t *start, unsigned long size,
00079 l4re_rm_flags_t flags, unsigned char align) L4_NOTHROW;
00080
00089 L4_CV L4_INLINE int
00090 l4re_rm_free_area(l4_addr_t addr) L4_NOTHROW;
00091
00142 L4_CV L4_INLINE int
00143 l4re_rm_attach(void **start, unsigned long size, l4re_rm_flags_t flags,
00144 l4re_ds_t mem, l4re_rm_offset_t offs,
00145 unsigned char align) L4_NOTHROW;
00146
00147
00165 L4_CV L4_INLINE int
00166 l4re_rm_detach(void *addr) L4_NOTHROW;
00167
00187 L4_CV L4_INLINE int

```

```

00188 l4re_rm_detach_ds(void *addr, l4re_ds_t *ds) L4_NOTHROW;
00189
00201 L4_CV L4_INLINE int
00202 l4re_rm_detach_unmap(l4_addr_t addr, l4_cap_idx_t task) L4_NOTHROW;
00203
00218 L4_CV L4_INLINE int
00219 l4re_rm_detach_ds_unmap(void *addr, l4re_ds_t *ds,
00220 l4_cap_idx_t task) L4_NOTHROW;
00221
00222
00229 L4_CV L4_INLINE int
00230 l4re_rm_find(l4_addr_t *addr, unsigned long *size,
00231 l4re_rm_offset_t *offset,
00232 l4re_rm_flags_t *flags, l4re_ds_t *m) L4_NOTHROW;
00233
00241 L4_CV L4_INLINE int
00242 l4re_rm_get_info(l4_addr_t addr,
00243 char *name, unsigned int len,
00244 l4re_rm_offset_t *backing_offset) L4_NOTHROW;
00245
00246
00253 L4_CV L4_INLINE void
00254 l4re_rm_show_lists(void) L4_NOTHROW;
00255
00256
00257 /*
00258 * Variants of functions that also take a capability of the region map
00259 * service.
00260 */
00261
00262
00267 L4_CV int
00268 l4re_rm_reserve_area_srv(l4_cap_idx_t rm, l4_addr_t *start, unsigned long size,
00269 l4re_rm_flags_t flags, unsigned char align) L4_NOTHROW;
00270
00275 L4_CV int
00276 l4re_rm_free_area_srv(l4_cap_idx_t rm, l4_addr_t addr) L4_NOTHROW;
00277
00282 L4_CV int
00283 l4re_rm_attach_srv(l4_cap_idx_t rm, void **start, unsigned long size,
00284 l4re_rm_flags_t flags, l4re_ds_t mem,
00285 l4re_rm_offset_t offs,
00286 unsigned char align) L4_NOTHROW;
00287
00288
00293 L4_CV int
00294 l4re_rm_detach_srv(l4_cap_idx_t rm, l4_addr_t addr,
00295 l4re_ds_t *ds, l4_cap_idx_t task) L4_NOTHROW;
00296
00297
00302 L4_CV int
00303 l4re_rm_find_srv(l4_cap_idx_t rm, l4_addr_t *addr,
00304 unsigned long *size, l4re_rm_offset_t *offset,
00305 l4re_rm_flags_t *flags, l4re_ds_t *m) L4_NOTHROW;
00306
00307
00312 L4_CV int
00313 l4re_rm_get_info_srv(l4_cap_idx_t rm, l4_addr_t addr,
00314 char *name, unsigned int len,
00315 l4re_rm_offset_t *backing_offset) L4_NOTHROW;
00316
00321 L4_CV void
00322 l4re_rm_show_lists_srv(l4_cap_idx_t rm) L4_NOTHROW;
00323
00324
00325 /***** Implementations *****/
00326
00327 L4_CV L4_INLINE int
00328 l4re_rm_reserve_area(l4_addr_t *start, unsigned long size,
00329 l4re_rm_flags_t flags, unsigned char align) L4_NOTHROW
00330 {
00331 return l4re_rm_reserve_area_srv(l4re_global_env->rm, start, size,
00332 flags, align);
00333 }
00334
00335 L4_CV L4_INLINE int
00336 l4re_rm_free_area(l4_addr_t addr) L4_NOTHROW
00337 {
00338 return l4re_rm_free_area_srv(l4re_global_env->rm, addr);
00339 }
00340
00341 L4_CV L4_INLINE int
00342 l4re_rm_attach(void **start, unsigned long size, l4re_rm_flags_t flags,
00343 l4re_ds_t mem, l4re_rm_offset_t offs,
00344 unsigned char align) L4_NOTHROW
00345 {
00346 return l4re_rm_attach_srv(l4re_global_env->rm, start, size,

```

```

00347 flags, mem, offs, align);
00348 }
00349
00350
00351 L4_CV L4_INLINE int
00352 l4re_rm_detach(void *addr) L4_NOTHROW
00353 {
00354 return l4re_rm_detach_srv(l4re_global_env->rm,
00355 (l4_addr_t)addr, 0, L4_BASE_TASK_CAP);
00356 }
00357
00358 L4_CV L4_INLINE int
00359 l4re_rm_detach_unmap(l4_addr_t addr, l4_cap_idx_t task) L4_NOTHROW
00360 {
00361 return l4re_rm_detach_srv(l4re_global_env->rm, addr, 0, task);
00362 }
00363
00364 L4_CV L4_INLINE int
00365 l4re_rm_detach_ds(void *addr, l4re_ds_t *ds) L4_NOTHROW
00366 {
00367 return l4re_rm_detach_srv(l4re_global_env->rm, (l4_addr_t)addr,
00368 ds, L4_BASE_TASK_CAP);
00369 }
00370
00371 L4_CV L4_INLINE int
00372 l4re_rm_detach_ds_unmap(void *addr, l4re_ds_t *ds, l4_cap_idx_t task) L4_NOTHROW
00373 {
00374 return l4re_rm_detach_srv(l4re_global_env->rm, (l4_addr_t)addr,
00375 ds, task);
00376 }
00377
00378 L4_CV L4_INLINE int
00379 l4re_rm_find(l4_addr_t *addr, unsigned long *size,
00380 l4re_rm_offset_t *offset,
00381 l4re_rm_flags_t *flags, l4re_ds_t *m) L4_NOTHROW
00382 {
00383 return l4re_rm_find_srv(l4re_global_env->rm, addr, size, offset, flags, m);
00384 }
00385
00386 L4_CV L4_INLINE void
00387 l4re_rm_show_lists(void) L4_NOTHROW
00388 {
00389 l4re_rm_show_lists_srv(l4re_global_env->rm);
00390 }
00391
00392
00393
00394 L4_CV L4_INLINE int
00395 l4re_rm_get_info(l4_addr_t addr, char *name, unsigned int len,
00396 l4re_rm_offset_t *backing_offset) L4_NOTHROW
00397 {
00398 return l4re_rm_get_info_srv(l4re_global_env->rm, addr, name, len,
00399 backing_offset);
00400 }
00401
00402 L4_END_DECLS

```

## 17.310 l4re/c/util/cap\_alloc.h File Reference

Capability allocator C interface.

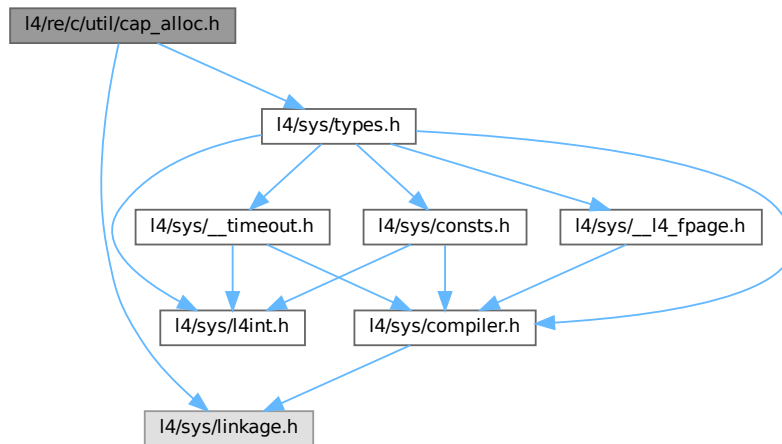
```

#include <l4/sys/types.h>
#include <l4/sys/linkage.h>

```



Include dependency graph for cap\_alloc.h:



## Functions

- `L4_BEGIN_DECLS l4_cap_idx_t l4re_util_cap_alloc (void) L4_NOTHROW`  
Get free capability index at capability allocator.
- `void l4re_util_cap_free (l4_cap_idx_t cap) L4_NOTHROW`  
Return capability index to capability allocator.
- `void l4re_util_cap_free_um (l4_cap_idx_t cap) L4_NOTHROW`  
Return capability index to capability allocator, and unmaps the object.
- `long l4re_util_cap_last (void) L4_NOTHROW`  
Return last capability index the allocator can return.

### 17.310.1 Detailed Description

Capability allocator C interface.

Definition in file [cap\\_alloc.h](#).

## 17.311 cap\_alloc.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00019
00020 #include <l4/sys/types.h>
00021 #include <l4/sys/linkage.h>

```

```

00022
00023 L4_BEGIN_DECLS
00024
00029 L4_CV l4_cap_idx_t
00030 l4re_util_cap_alloc(void) L4_NOTHROW;
00031
00036 L4_CV void
00037 l4re_util_cap_free(l4_cap_idx_t cap) L4_NOTHROW;
00038
00044 L4_CV void
00045 l4re_util_cap_free_um(l4_cap_idx_t cap) L4_NOTHROW;
00046
00052 L4_CV long
00053 l4re_util_cap_last(void) L4_NOTHROW;
00054
00055 L4_END_DECLS

```

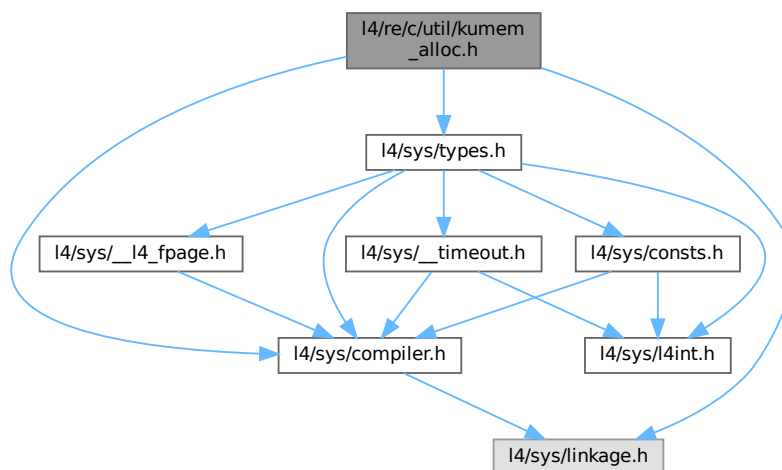
## 17.312 l4/re/c/util/kumem\_alloc.h File Reference

Kumem allocator utility C interface.

```

#include <l4/sys/compiler.h>
#include <l4/sys/types.h>
#include <l4/sys/linkage.h>
Include dependency graph for kumem_alloc.h:

```



### Functions

- [L4\\_BEGIN\\_DECLS](#) `int l4re_util_kumem_alloc (l4_addr_t *mem, unsigned pages_order, l4\_cap\_idx\_t task, l4\_cap\_idx\_t rm) L4\_NOTHROW`  
Allocate state area.

### 17.312.1 Detailed Description

Kumem allocator utility C interface.

Definition in file [kumem\\_alloc.h](#).

## 17.312.2 Function Documentation

### 17.312.2.1 l4re\_util\_kumem\_alloc()

```
L4_BEGIN_DECLS int l4re_util_kumem_alloc (
 l4_addr_t * mem,
 unsigned pages_order,
 l4_cap_idx_t task,
 l4_cap_idx_t rm)
```

Allocate state area.

#### Parameters

|     |                    |                                            |
|-----|--------------------|--------------------------------------------|
| out | <i>mem</i>         | Pointer to memory that has been allocated. |
|     | <i>pages_order</i> | Size to allocate, in log2 pages.           |
|     | <i>task</i>        | Task to use for allocation.                |
|     | <i>rm</i>          | Region manager to use for allocation.      |

#### Return values

|    |                       |
|----|-----------------------|
| 0  | for success           |
| <0 | error code on failure |

#### Note

The amount of kernel-user memory that can be allocated at once is limited by the used kernel implementation. The minimum allocatable amount is one page. A portable implementation should not depend on allocations greater than 16KiB to succeed.

#### Examples

[examples/sys/aliens/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

References [L4\\_END\\_DECLS](#), and [L4\\_NOTHROW](#).

## 17.313 kumem\_alloc.h

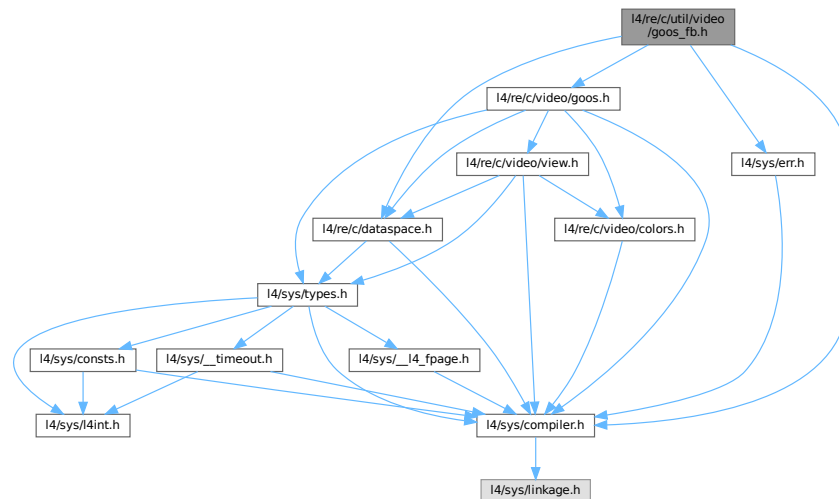
[Go to the documentation of this file.](#)

```
00001
00005 /*
00006 * (c) 2010 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00019
00020 #include <l4/sys/compiler.h>
00021 #include <l4/sys/types.h>
00022 #include <l4/sys/linkage.h>
00023
00024 L4_BEGIN_DECLS
00025
00029 L4_CV int
00030 l4re_util_kumem_alloc(l4_addr_t *mem, unsigned pages_order,
00031 l4_cap_idx_t task, l4_cap_idx_t rm) L4_NOTHROW;
00032
00033 L4_END_DECLS
```

## 17.314 l4/re/c/util/video/goos\_fb.h File Reference

Framebuffer utility functionality.

```
#include <l4/sys/compiler.h>
#include <l4/re/c/video/goos.h>
#include <l4/sys/err.h>
#include <l4/re/c/dataspace.h>
Include dependency graph for goos_fb.h:
```



### 17.314.1 Detailed Description

Framebuffer utility functionality.

Definition in file [goos\\_fb.h](#).

## 17.315 goos\_fb.h

[Go to the documentation of this file.](#)

```
00001
00005 /*
00006 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #pragma once
00012
00013 #include <l4/sys/compiler.h>
00014 #include <l4/re/c/video/goos.h>
00015 #include <l4/sys/err.h>
00016 #include <l4/re/c/dataspace.h>
00017
00018 L4_BEGIN_DECLS
00019
00020 typedef struct
00021 {
00022 unsigned long _obj_buf[6];
00023 } l4re_util_video_goos_fb_t;
```

```

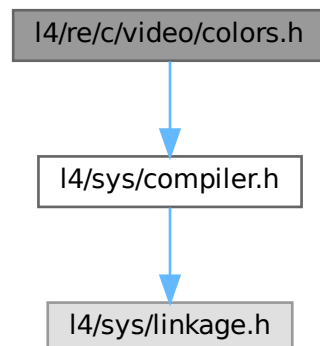
00024
00025 L4_CV int
00026 l4re_util_video_goos_fb_setup_name(l4re_util_video_goos_fb_t *goosfb,
00027 char const *name) L4_NOTHROW;
00028
00029 L4_CV void
00030 l4re_util_video_goos_fb_destroy(l4re_util_video_goos_fb_t *goosfb) L4_NOTHROW;
00031
00032 L4_CV int
00033 l4re_util_video_goos_fb_view_info(l4re_util_video_goos_fb_t *goosfb,
00034 l4re_video_view_info_t *info) L4_NOTHROW;
00035
00036 L4_CV void *
00037 l4re_util_video_goos_fb_attach_buffer(l4re_util_video_goos_fb_t *goosfb) L4_NOTHROW;
00038
00039 L4_CV int
00040 l4re_util_video_goos_fb_refresh(l4re_util_video_goos_fb_t *goosfb,
00041 int x, int y, int w, int h) L4_NOTHROW;
00042
00043 L4_CV l4re_ds_t
00044 l4re_util_video_goos_fb_buffer(l4re_util_video_goos_fb_t *goosfb) L4_NOTHROW;
00045
00046 L4_CV l4_cap_idx_t
00047 l4re_util_video_goos_fb_goos(l4re_util_video_goos_fb_t *goosfb) L4_NOTHROW;
00048
00049 L4_END_DECLS

```

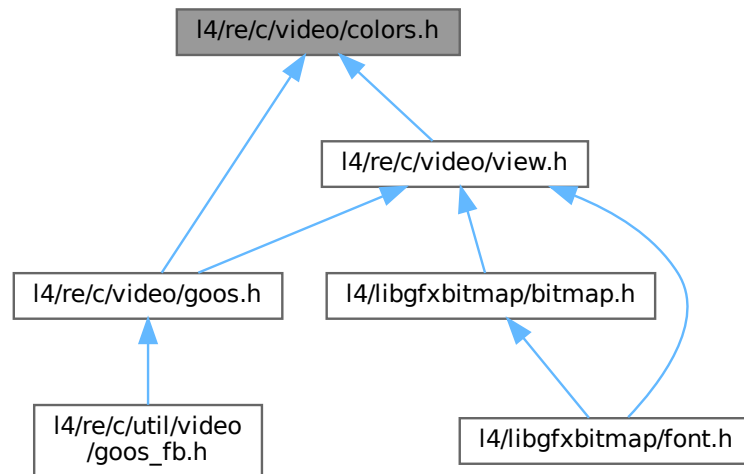
## 17.316 l4/re/c/video/colors.h File Reference

#include <l4/sys/compiler.h>

Include dependency graph for colors.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [l4re\\_video\\_color\\_component\\_t](#)  
*Color component structure.*
- struct [l4re\\_video\\_pixel\\_info\\_t](#)  
*Pixel\_info structure.*

## Typedefs

- typedef struct l4re\_video\_color\_component\_t **l4re\_video\_color\_component\_t**  
*Color component structure.*
- typedef struct l4re\_video\_pixel\_info\_t **l4re\_video\_pixel\_info\_t**  
*Pixel\_info structure.*

## 17.316.1 Detailed Description

### Note

The C interface of L4Re::Video does *NOT* reflect the full C++ interface on purpose. Use the C++ interface where possible.

Definition in file [colors.h](#).

## 17.317 colors.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00014 #include <l4/sys/compiler.h>
00015
00020 typedef struct l4re_video_color_component_t
00021 {
00022 unsigned char size;
00023 unsigned char shift;
00024 } __attribute__((packed)) l4re_video_color_component_t;
00025
00030 typedef struct l4re_video_pixel_info_t
00031 {
00032 l4re_video_color_component_t r, g, b, a;
00033 unsigned char bytes_per_pixel;
00034 } l4re_video_pixel_info_t;
00035
00036 L4_BEGIN_DECLS
00037
00038 L4_INLINE L4_CV int
00039 l4re_video_bits_per_pixel(l4re_video_pixel_info_t *p) L4_NOTHROW;
00040
00041 /* *****
00042 * Implementations */
00043
00044 L4_INLINE L4_CV int
00045 l4re_video_bits_per_pixel(l4re_video_pixel_info_t *p) L4_NOTHROW
00046 {
00047 return p->r.size + p->b.size + p->g.size + p->a.size;
00048 }
00049
00050 L4_END_DECLS

```

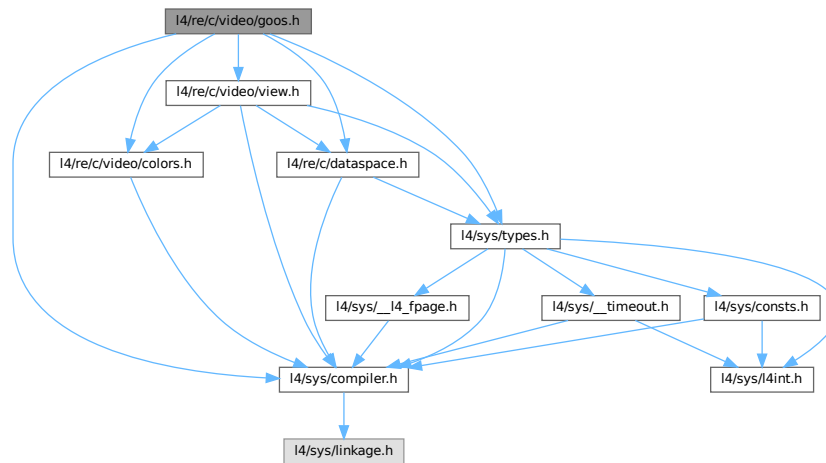
## 17.318 l4/re/c/video/goos.h File Reference

```

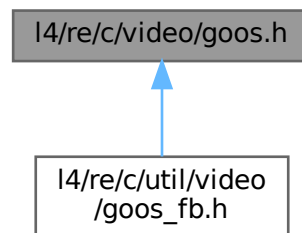
#include <l4/sys/compiler.h>
#include <l4/sys/types.h>
#include <l4/re/c/dataspace.h>
#include <l4/re/c/video/colors.h>
#include <l4/re/c/video/view.h>

```

Include dependency graph for goos.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [l4re\\_video\\_goos\\_info\\_t](#)  
*Goos information structure.*

## Typedefs

- typedef [l4\\_cap\\_idx\\_t](#) [l4re\\_video\\_goos\\_t](#)  
*Goos object type.*

## Enumerations

- enum [l4re\\_video\\_goos\\_info\\_flags\\_t](#) { [F\\_l4re\\_video\\_goos\\_auto\\_refresh](#) = 0x01 , [F\\_l4re\\_video\\_goos\\_pointer](#) = 0x02 , [F\\_l4re\\_video\\_goos\\_dynamic\\_views](#) = 0x04 , [F\\_l4re\\_video\\_goos\\_dynamic\\_buffers](#) = 0x08 }  
*Flags of information on the goos.*



## Functions

- `L4_BEGIN_DECLS` `int` `l4re_video_goos_info` (`l4re_video_goos_t` goos, `l4re_video_goos_info_t` \*ginfo) `L4_NOTHROW`  
*Get information on a goos.*
- `int` `l4re_video_goos_refresh` (`l4re_video_goos_t` goos, `int` x, `int` y, `int` w, `int` h) `L4_NOTHROW`  
*Flush a rectangle of pixels of the goos screen.*
- `int` `l4re_video_goos_create_buffer` (`l4re_video_goos_t` goos, `unsigned long` size, `l4_cap_idx_t` buffer) `L4_NOTHROW`  
*Create a new buffer (memory buffer) for pixel data.*
- `int` `l4re_video_goos_delete_buffer` (`l4re_video_goos_t` goos, `unsigned` idx) `L4_NOTHROW`  
*Delete a pixel buffer.*
- `int` `l4re_video_goos_get_static_buffer` (`l4re_video_goos_t` goos, `unsigned` idx, `l4_cap_idx_t` buffer) `L4_NOTHROW`  
*Get the data-space capability of the static pixel buffer.*
- `int` `l4re_video_goos_create_view` (`l4re_video_goos_t` goos, `l4re_video_view_t` \*view) `L4_NOTHROW`  
*Create a new view (.*
- `int` `l4re_video_goos_delete_view` (`l4re_video_goos_t` goos, `l4re_video_view_t` \*view) `L4_NOTHROW`  
*Delete a view.*
- `int` `l4re_video_goos_get_view` (`l4re_video_goos_t` goos, `unsigned` idx, `l4re_video_view_t` \*view) `L4_NOTHROW`  
*Get a view for the given index.*

### 17.318.1 Detailed Description

#### Note

The C interface of L4Re::Video does *NOT* reflect the full C++ interface on purpose. Use the C++ where possible.

Definition in file [goos.h](#).

## 17.319 goos.h

[Go to the documentation of this file.](#)

```

00001
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include <l4/sys/compiler.h>
00011 #include <l4/sys/types.h>
00012 #include <l4/re/c/dataspace.h>
00013 #include <l4/re/c/video/colors.h>
00014 #include <l4/re/c/video/view.h>
00015
00016
00017
00018
00019
00020
00021
00022
00023
00024
00025
00026
00027
00028
00029
00030
00031
00032
00033
00034
00035
00036
00037
00038
00039
00040
00041
00042
00043
00044
00045
00046
00047
00048
00049
00050
00051
00052
00053
00054
00055
00056
00057
00058
00059
00060
00061
00062
00063
00064
00065
00066
00067
00068
00069
00070
00071
00072
00073
00074
00075
00076
00077
00078
00079
00080
00081
00082
00083
00084
00085
00086
00087
00088
00089
00090
00091
00092
00093
00094
00095
00096
00097
00098
00099
00100
00101
00102
00103
00104
00105
00106
00107
00108
00109
00110
00111
00112
00113
00114
00115
00116
00117
00118
00119
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
00130
00131
00132
00133
00134
00135
00136
00137
00138
00139
00140
00141
00142
00143
00144
00145
00146
00147
00148
00149
00150
00151
00152
00153
00154
00155
00156
00157
00158
00159
00160
00161
00162
00163
00164
00165
00166
00167
00168
00169
00170
00171
00172
00173
00174
00175
00176
00177
00178
00179
00180
00181
00182
00183
00184
00185
00186
00187
00188
00189
00190
00191
00192
00193
00194
00195
00196
00197
00198
00199
00200
00201
00202
00203
00204
00205
00206
00207
00208
00209
00210
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232
00233
00234
00235
00236
00237
00238
00239
00240
00241
00242
00243
00244
00245
00246
00247
00248
00249
00250
00251
00252
00253
00254
00255
00256
00257
00258
00259
00260
00261
00262
00263
00264
00265
00266
00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000

```

```

00041 typedef struct
00042 {
00043 unsigned long width;
00044 unsigned long height;
00045 unsigned flags;
00046 unsigned num_static_views;
00047 unsigned num_static_buffers;
00048 l4re_video_pixel_info_t pixel_info;
00049 } l4re_video_goos_info_t;
00050
00055 typedef l4_cap_idx_t l4re_video_goos_t;
00056
00057 L4_BEGIN_DECLS
00058
00070 L4_CV int
00071 l4re_video_goos_info(l4re_video_goos_t goos,
00072 l4re_video_goos_info_t *ginfo) L4_NOTHROW;
00073
00083 L4_CV int
00084 l4re_video_goos_refresh(l4re_video_goos_t goos, int x, int y, int w,
00085 int h) L4_NOTHROW;
00086
00098 L4_CV int
00099 l4re_video_goos_create_buffer(l4re_video_goos_t goos, unsigned long size,
00100 l4_cap_idx_t buffer) L4_NOTHROW;
00101
00109 L4_CV int
00110 l4re_video_goos_delete_buffer(l4re_video_goos_t goos, unsigned idx) L4_NOTHROW;
00111
00122 L4_CV int
00123 l4re_video_goos_get_static_buffer(l4re_video_goos_t goos, unsigned idx,
00124 l4_cap_idx_t buffer) L4_NOTHROW;
00125
00132 L4_CV int
00133 l4re_video_goos_create_view(l4re_video_goos_t goos,
00134 l4re_video_view_t *view) L4_NOTHROW;
00135
00143 L4_CV int
00144 l4re_video_goos_delete_view(l4re_video_goos_t goos,
00145 l4re_video_view_t *view) L4_NOTHROW;
00146
00147
00159 L4_CV int
00160 l4re_video_goos_get_view(l4re_video_goos_t goos, unsigned idx,
00161 l4re_video_view_t *view) L4_NOTHROW;
00162
00163 L4_END_DECLS

```

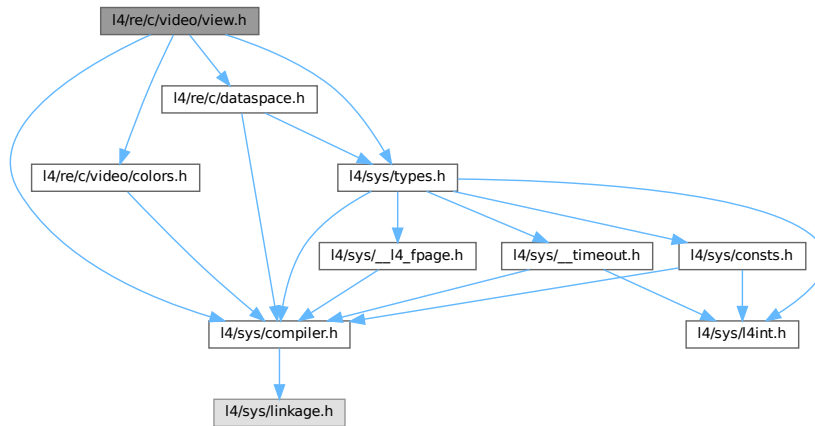
## 17.320 l4re/c/video/view.h File Reference

```

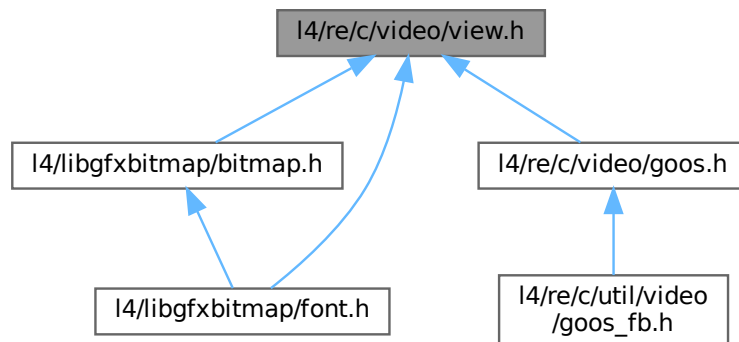
#include <l4/sys/compiler.h>
#include <l4/sys/types.h>
#include <l4/re/c/dataspace.h>
#include <l4/re/c/video/colors.h>

```

Include dependency graph for view.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [l4re\\_video\\_view\\_info\\_t](#)  
View information structure.
- struct [l4re\\_video\\_view\\_t](#)  
C representation of a goos view.

## Typedefs

- typedef struct l4re\_video\_view\_info\_t [l4re\\_video\\_view\\_info\\_t](#)  
View information structure.
- typedef struct l4re\_video\_view\_t [l4re\\_video\\_view\\_t](#)  
C representation of a goos view.

## Enumerations

- enum `l4re_video_view_info_flags_t` {  
`F_l4re_video_view_none` = 0x00 , `F_l4re_video_view_set_buffer` = 0x01 , `F_l4re_video_view_set_buffer_offset` = 0x02 , `F_l4re_video_view_set_bytes_per_line` = 0x04 ,  
`F_l4re_video_view_set_pixel` = 0x08 , `F_l4re_video_view_set_position` = 0x10 , `F_l4re_video_view_dyn_allocated` = 0x20 , `F_l4re_video_view_set_background` = 0x40 ,  
`F_l4re_video_view_set_flags` = 0x80 , **`F_l4re_video_view_fully_dynamic`** , `F_l4re_video_view_above` = 0x01000 , `F_l4re_video_view_flags_mask` = 0xff000 }

*Flags of information on a view.*

## Functions

- `L4_BEGIN_DECLS` int `l4re_video_view_refresh` (`l4re_video_view_t` \*view, int x, int y, int w, int h) `L4_NOTHROW`  
*Flush the given rectangle of pixels of the given view.*
- int `l4re_video_view_get_info` (`l4re_video_view_t` \*view, `l4re_video_view_info_t` \*info) `L4_NOTHROW`  
*Retrieve information about the given view.*
- int `l4re_video_view_set_info` (`l4re_video_view_t` \*view, `l4re_video_view_info_t` \*info) `L4_NOTHROW`  
*Set properties of the view.*
- int `l4re_video_view_set_viewport` (`l4re_video_view_t` \*view, int x, int y, int w, int h, unsigned long bofs) `L4_NOTHROW`  
*Set the viewport parameters of a view.*
- int `l4re_video_view_stack` (`l4re_video_view_t` \*view, `l4re_video_view_t` \*pivot, int behind) `L4_NOTHROW`  
*Change the stacking order in the stack of visible views.*

## 17.320.1 Detailed Description

### Note

The C interface of `L4Re::Video` does *NOT* reflect the full C++ interface on purpose. Use the C++ where possible.

Definition in file [view.h](#).

## 17.321 view.h

[Go to the documentation of this file.](#)

```
00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00014 #include <l4/sys/compiler.h>
00015 #include <l4/sys/types.h>
00016 #include <l4/re/c/dataspace.h>
00017 #include <l4/re/c/video/colors.h>
00018
00023 enum l4re_video_view_info_flags_t
00024 {
00025 F_l4re_video_view_none = 0x00,
00026 F_l4re_video_view_set_buffer = 0x01,
00027 F_l4re_video_view_set_buffer_offset = 0x02,
```

```

00028 F_l4re_video_view_set_bytes_per_line = 0x04,
00029 F_l4re_video_view_set_pixel = 0x08,
00030 F_l4re_video_view_set_position = 0x10,
00031 F_l4re_video_view_dyn_allocated = 0x20,
00032 F_l4re_video_view_set_background = 0x40,
00033 F_l4re_video_view_set_flags = 0x80,
00034 F_l4re_video_view_fully_dynamic = F_l4re_video_view_set_buffer
00035 | F_l4re_video_view_set_buffer_offset
00036 | F_l4re_video_view_set_bytes_per_line
00037 | F_l4re_video_view_set_pixel
00038 | F_l4re_video_view_set_position
00039 | F_l4re_video_view_dyn_allocated,
00040
00041 F_l4re_video_view_above = 0x01000,
00042 F_l4re_video_view_flags_mask = 0xff000,
00043 };
00044
00049 typedef struct l4re_video_view_info_t
00050 {
00051 unsigned flags;
00052 unsigned view_index;
00053 unsigned long xpos, ypos, width, height;
00054 unsigned long buffer_offset;
00055 unsigned long bytes_per_line;
00056 l4re_video_pixel_info_t pixel_info;
00057 unsigned buffer_index;
00058 } l4re_video_view_info_t;
00059
00060
00068 typedef struct l4re_video_view_t
00069 {
00070 l4_cap_idx_t goos;
00071 unsigned idx;
00072 } l4re_video_view_t;
00073
00074
00075 L4_BEGIN_DECLS
00076
00087 L4_CV int
00088 l4re_video_view_refresh(l4re_video_view_t *view, int x, int y, int w,
00089 int h) L4_NOTHROW;
00090
00098 L4_CV int
00099 l4re_video_view_get_info(l4re_video_view_t *view,
00100 l4re_video_view_info_t *info) L4_NOTHROW;
00101
00113 L4_CV int
00114 l4re_video_view_set_info(l4re_video_view_t *view,
00115 l4re_video_view_info_t *info) L4_NOTHROW;
00116
00133 L4_CV int
00134 l4re_video_view_set_viewport(l4re_video_view_t *view, int x, int y, int w,
00135 int h, unsigned long bofs) L4_NOTHROW;
00136
00147 L4_CV int
00148 l4re_video_view_stack(l4re_video_view_t *view, l4re_video_view_t *pivot,
00149 int behind) L4_NOTHROW;
00150
00151 L4_END_DECLS
00152

```

## 17.322 console

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009
00010 #pragma once
00011
00012 #include <l4/re/video/goos>
00013 #include <l4/re/event>
00014
00015 namespace L4Re {
00016
00022
00023
00028 class L4_EXPORT Console :
00029 public L4::Kobject_2t<Console, Video::Goos, Event, L4::PROTO_EMPTY>

```



## Data Structures

- class [L4Re::Dataspace](#)  
*Interface for memory-like objects.*
- struct [L4Re::Dataspace::F](#)  
*Dataspace flags definitions.*
- struct [L4Re::Dataspace::Stats](#)  
*Information about the dataspace.*

## Namespaces

- namespace [L4Re](#)  
*L4Re C++ Interfaces.*

### 17.323.1 Detailed Description

Dataspace interface.

Definition in file [dataspace](#).

## 17.324 dataspace

[Go to the documentation of this file.](#)

```

00001 // -*- Mode: C++ -*-
00002 // vim:ft=cpp
00007 /*
00008 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00009 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00010 * Björn Döbel <doebel@os.inf.tu-dresden.de>,
00011 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00012 * economic rights: Technische Universität Dresden (Germany)
00013 *
00014 * License: see LICENSE.spdx (in this directory or the directories above)
00015 */
00016
00017 #pragma once
00018
00019 #include <l4/bid_config.h>
00020 #include <l4/sys/types.h>
00021 #include <l4/sys/l4int.h>
00022 #include <l4/sys/capability>
00023 #include <l4/re/protocols.h>
00024 #include <l4/sys/cxx/ipc_types>
00025 #include <l4/sys/cxx/ipc_iface>
00026 #include <l4/sys/cxx/types>
00027
00028 namespace L4Re
00029 {
00030
00031 // MISSING:
00032 // * size support in map, mapped size in reply
00033
00050 class L4_EXPORT Dataspace :
00051 public L4::Kobject_t<Dataspace, L4::Kobject, L4RE_PROTO_DATASPACE,
00052 L4::Type_info::Demand_t<1> >
00053 {
00054 public:
00055
00057 struct F
00058 {
00059 enum
00060 {
00061 Caching_shift = 4,
00062 };
00063

```

```

00070 enum Flags
00071 {
00072 R = L4_FPAGE_RO,
00073 Ro = L4_FPAGE_RO,
00074 RW = L4_FPAGE_RW,
00075 W = L4_FPAGE_W,
00076 X = L4_FPAGE_X,
00077 RX = L4_FPAGE_RX,
00078 RWX = L4_FPAGE_RWX,
00079 Rights_mask = 0x0f,
00080
00081 Normal = 0x00,
00082 Cacheable = Normal,
00083 Bufferable = 0x10,
00084 Uncacheable = 0x20,
00085 Caching_mask = 0x30,
00086 };
00087
00088 L4_TYPES_FLAGS_OPS_DEF(Flags);
00089 };
00090
00091 struct Flags : L4::Types::Flags_ops_t<Flags>
00092 {
00093 unsigned long raw;
00094 Flags() = default;
00095 explicit constexpr Flags(unsigned long f) : raw(f) {}
00096 constexpr Flags(F::Flags f) : raw(f) {}
00097 constexpr bool r() const { return raw & L4_FPAGE_RO; }
00098 constexpr bool w() const { return raw & L4_FPAGE_W; }
00099 constexpr bool x() const { return raw & L4_FPAGE_X; }
00100
00101 constexpr unsigned long fpage_rights() const
00102 { return raw & 0xf; }
00103 };
00104
00105 typedef l4_uint64_t Size;
00106 typedef l4_uint64_t Offset;
00107 typedef l4_uint64_t Map_addr;
00108
00109 struct Stats
00110 {
00111 Size size;
00112 Flags flags;
00113 };
00114
00115 long map(Offset offset, Flags flags, Map_addr local_addr,
00116 Map_addr min_addr, Map_addr max_addr,
00117 L4::Cap<L4::Task> dst = L4::Cap<L4::Task>::Invalid) const noexcept;
00118
00119 long map_region(Offset offset, Flags flags,
00120 Map_addr min_addr, Map_addr max_addr,
00121 L4::Cap<L4::Task> dst = L4::Cap<L4::Task>::Invalid) const noexcept;
00122
00123 L4_RPC(long, clear, (Offset offset, Size size));
00124
00125 L4_RPC(long, allocate, (Offset offset, Size size));
00126
00127 L4_RPC(long, copy_in, (Offset dst_offs, L4::Ipc::Cap<Dataspace> src,
00128 Offset src_offs, Size size));
00129
00130 Size size() const noexcept;
00131
00132 Flags flags() const noexcept;
00133
00134 L4_RPC(long, info, (Stats *stats));
00135
00136 L4_RPC_NF(long, map, (Offset offset, Map_addr spot,
00137 Flags flags, L4::Ipc::Rcv_fpage r,
00138 L4::Ipc::Snd_fpage &fp));
00139
00140 #ifdef CONFIG_MMU
00141 L4_RPC_NF(long, map_info, (l4_addr_t *start_addr, l4_addr_t *end_addr));
00142 inline long map_info([[maybe_unused]] l4_addr_t *start_addr,
00143 [[maybe_unused]] l4_addr_t *end_addr)
00144 { return 0; }
00145 #else
00146 L4_RPC(long, map_info, (l4_addr_t *start_addr, l4_addr_t *end_addr));
00147 #endif
00148
00149 private:
00150 long __map(Offset offset, unsigned char *order, Flags flags,
00151 Map_addr local_addr, L4::Cap<L4::Task> dst) const noexcept;
00152
00153 public:
00154 typedef L4::Typeid::Rpcs<map_t, clear_t, info_t, copy_in_t,

```

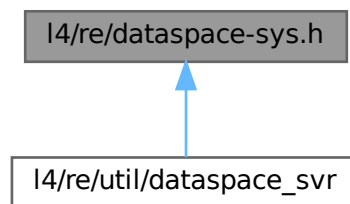


```
00319 allocate_t, map_info_t> RpcS;
00320
00321 };
00322
00323 }
00324
```

## 17.325 l4/re/dataspace-sys.h File Reference

Dataspace protocol defintion.

This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [L4Re](#)  
*L4Re C++ Interfaces.*

### Enumerations

- enum [L4Re::Dataspace\\_::Opcodes](#)  
*Data-space communication-protocol opcodes.*

### 17.325.1 Detailed Description

Dataspace protocol defintion.

Definition in file [dataspace-sys.h](#).

## 17.326 dataspace-sys.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00014 namespace L4Re
00015 {
00016 namespace Dataspace_
00017 {
00023 enum Opcodes { Map, Clear, Stats, Copy, Take, Release, Allocate };
00024 };
00025 };
00026

```

## 17.327 dbg\_events

```

00001 // vim:ft=cpp
00002
00003 #pragma once
00004
00005 #include <l4/sys/cxx/ipc_epiface>
00006 #include <l4/sys/utcb.h>
00007 #include <l4/re/remote_access>
00008 #include <l4/re/rm>
00009
00010 namespace L4Re {
00011 struct Dbg_events : L4::Kobject_t<Dbg_events, L4::Kobject, 0,
00012 L4::Type_info::Demand_t<2> >
00013 {
00014 L4_INLINE_RPC(int, request_backtrace, (l4_exc_regs_t regs,
00015 L4::Ipc::Cap<L4Re::Remote_access> raif,
00016 L4::Ipc::Cap<L4Re::Rm> rmif));
00017
00018 typedef L4::Typeid::Rpc<request_backtrace_t> Rpc;
00019 };
00020 } // L4Re

```

## 17.328 l4/re/dma\_space File Reference

```

#include <l4/sys/types.h>
#include <l4/sys/l4int.h>
#include <l4/sys/capability>
#include <l4/re/dataspace>
#include <l4/re/protocols.h>
#include <l4/sys/cxx/types>
#include <l4/sys/cxx/ipc_types>
#include <l4/sys/cxx/ipc_iface>

```

```

graph TD
 I4libblockdeviceblock_device_mgr_h["I4/libblock-device/block_device_mgr.h"] --> I4libblockdevicepart_device_h["I4/libblock-device/part_device.h"]
 I4libblockdeviceblock_device_mgr_h --> I4libblockdevicevirtio_client_h["I4/libblock-device/virtio_client.h"]
 I4libblockdeviceblock_device_mgr_h --> I4libblockdevicescheduler_h["I4/libblock-device/scheduler.h"]
 I4libblockdeviceblock_device_mgr_h --> I4libblockdeviceinout_memory_h["I4/libblock-device/inout_memory.h"]
 I4libblockdeviceblock_device_mgr_h --> I4libblockdevicedevice_h["I4/libblock-device/device.h"]
 I4libblockdeviceblock_device_mgr_h --> I4re_dma_space["I4/re/dma_space"]
 I4libblockdevicepart_device_h --> I4libblockdevicepartition_h["I4/libblock-device/partition.h"]
 I4libblockdevicevirtio_client_h --> I4libblockdevicetypes_h["I4/libblock-device/types.h"]
 I4libblockdevicescheduler_h --> I4libblockdevicedevice_h
 I4libblockdeviceinout_memory_h --> I4re_dma_space
 I4libblockdevicedevice_h --> I4re_dma_space
 I4libblockdevicedevice_h --> I4vbus_vbus["I4/vbus/vbus"]
 I4re_dma_space --> I4vbus_vbus
 I4vbus_vbus --> I4vbus_vbus_generic["I4/vbus/vbus_generic"]
 I4vbus_vbus --> I4vbus_vbus_gpio["I4/vbus/vbus_gpio"]
 I4vbus_vbus --> I4vbus_vbus_pci["I4/vbus/vbus_pci"]

```

- class `L4Re::Dma_space`  
*Managed DMA Address Space.*

- namespace **L4Re**  
*L4Re C++ Interfaces.*

## 17.329 dma\_space

[Go to the documentation of this file.](#)

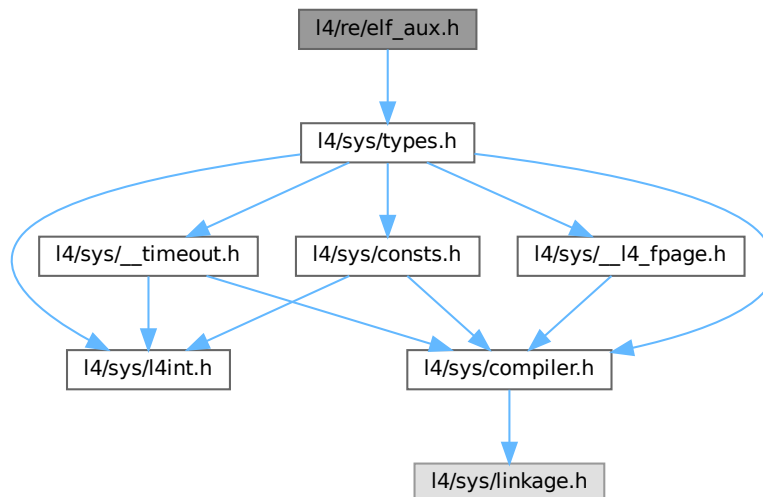
```

00001 // -*- Mode: C++ -*-
00002 // vim:ft=cpp
00006 /*
00007 * (c) 2014 Alexander Warg <alexander.warg@kernkonzept.com>
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011
00012 #pragma once
00013
00014 #include <l4/sys/types.h>
00015 #include <l4/sys/l4int.h>
00016 #include <l4/sys/capability>
00017 #include <l4/re/dataspace>
00018 #include <l4/re/protocols.h>
00019 #include <l4/sys/cxx/types>
00020 #include <l4/sys/cxx/ipc_types>
00021 #include <l4/sys/cxx/ipc_iface>
00022
00023 namespace L4Re
00024 {
00025
00052 class Dma_space :
00053 public L4::Kobject_0t< Dma_space,
00054 L4RE_PROTO_DMA_SPACE,
00055 L4::Type_info::Demand_t<1> >
00056 {
00057 public:
00059 typedef l4_uint64_t Dma_addr;
00060
00064 enum Direction
00065 {
00066 Bidirectional,
00067 To_device,
00068 From_device,
00069 None
00070 };
00071
00076 enum Attribute
00077 {
00089 No_sync
00090 };
00091
00097 typedef L4::Types::Flags<Attribute> Attributes;
00098
00104 enum Space_attrib
00105 {
00112 Coherent,
00113
00118 Phys_space
00119 };
00120
00122 typedef L4::Types::Flags<Space_attrib> Space_attribs;
00123
00159 L4_INLINE_RPC(
00160 long, map, (L4::Ipc::Cap<L4Re::Dataspace> src,
00161 L4Re::Dataspace::Offset offset,
00162 L4::Ipc::In_out<l4_size_t *> size,
00163 Attributes attrs, Direction dir,
00164 Dma_addr *dma_addr));
00165
00176 L4_INLINE_RPC(
00177 long, unmap, (Dma_addr dma_addr,
00178 l4_size_t size, Attributes attrs, Direction dir));
00179
00202 L4_INLINE_RPC(
00203 long, associate, (L4::Ipc::Opt<L4::Ipc::Cap<L4::Task> > dma_task,
00204 Space_attribs attr),
00205 L4::Ipc::Call_t<L4_CAP_FPAGE_RW>);
00206
00218 L4_INLINE_RPC(
00219 long, disassociate, (),
00220 L4::Ipc::Call_t<L4_CAP_FPAGE_RW>);
00221
00222 typedef L4::Typeid::Rpcs<map_t, unmap_t, associate_t, disassociate_t> Rpcs;
00223 };
00224
00225 }
```

## 17.330 l4/re/elf\_aux.h File Reference

Auxiliary information for binaries.

```
#include <l4/sys/types.h>
Include dependency graph for elf_aux.h:
```



### Data Structures

- struct `l4re_elf_aux_t`  
Generic header for each auxiliary vector element.
- struct `l4re_elf_aux_vma_t`  
Auxiliary vector element for a reserved virtual memory area.
- struct `l4re_elf_aux_mword_t`  
Auxiliary vector element for a single unsigned data word.

### Macros

- #define `L4RE_ELF_AUX_ELEM` `const __attribute__((used, section(".ro.l4re_elf_aux"), aligned(sizeof(l4_umword_t))))`  
Define an auxiliary vector element.
- #define `L4RE_ELF_AUX_ELEM_T` `(type, id, tag, val...)`  
Define an auxiliary vector element.

### Typedefs

- typedef struct `l4re_elf_aux_t` `l4re_elf_aux_t`  
Generic header for each auxiliary vector element.
- typedef struct `l4re_elf_aux_vma_t` `l4re_elf_aux_vma_t`  
Auxiliary vector element for a reserved virtual memory area.
- typedef struct `l4re_elf_aux_mword_t` `l4re_elf_aux_mword_t`  
Auxiliary vector element for a single unsigned data word.

## Enumerations

- enum {  
L4RE\_ELF\_AUX\_T\_NONE = 0 , L4RE\_ELF\_AUX\_T\_VMA , L4RE\_ELF\_AUX\_T\_STACK\_SIZE ,  
L4RE\_ELF\_AUX\_T\_STACK\_ADDR ,  
L4RE\_ELF\_AUX\_T\_KIP\_ADDR , L4RE\_ELF\_AUX\_T\_EX\_REGS\_FLAGS }

### 17.330.1 Detailed Description

Auxiliary information for binaries.

Definition in file [elf\\_aux.h](#).

## 17.331 elf\_aux.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2009 Alexander Warg <warg@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #pragma once
00012
00013 #include <l4/sys/types.h>
00014
00015
00028
00041 #define L4RE_ELF_AUX_ELEM const __attribute__((used, section(".rol4re_elf_aux"),
 aligned(sizeof(l4_umword_t))))
00042
00056 #define L4RE_ELF_AUX_ELEM_T(type, id, tag, val...) \
00057 static L4RE_ELF_AUX_ELEM type id = {tag, sizeof(type), val}
00058
00059 enum
00060 {
00064 L4RE_ELF_AUX_T_NONE = 0,
00065
00069 L4RE_ELF_AUX_T_VMA,
00070
00075 L4RE_ELF_AUX_T_STACK_SIZE,
00076
00081 L4RE_ELF_AUX_T_STACK_ADDR,
00082
00087 L4RE_ELF_AUX_T_KIP_ADDR,
00088
00092 L4RE_ELF_AUX_T_EX_REGS_FLAGS,
00093 };
00094
00098 typedef struct l4re_elf_aux_t
00099 {
00100 l4_umword_t type;
00101 l4_umword_t length;
00102 } l4re_elf_aux_t;
00103
00107 typedef struct l4re_elf_aux_vma_t
00108 {
00109 l4_umword_t type;
00110 l4_umword_t length;
00111 l4_umword_t start;
00112 l4_umword_t end;
00113 } l4re_elf_aux_vma_t;
00114
00118 typedef struct l4re_elf_aux_mword_t
00119 {
00120 l4_umword_t type;
00121 l4_umword_t length;
00122 l4_umword_t value;
00123 } l4re_elf_aux_mword_t;
00124

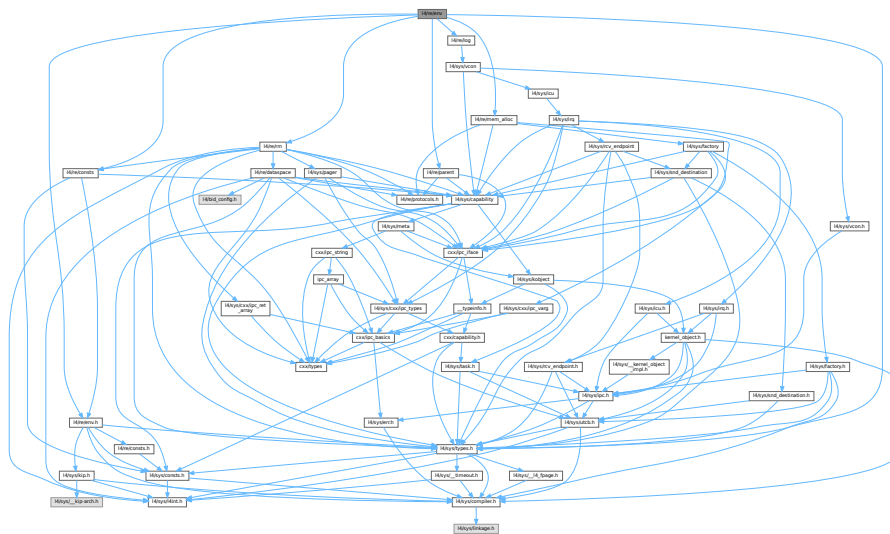
```

## 17.332 l4/re/env File Reference

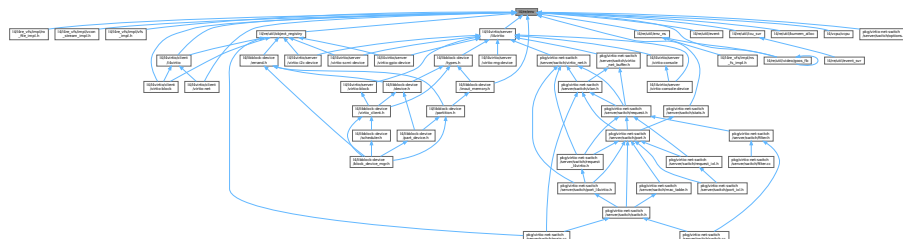
Environment interface.

```
#include <l4/sys/types.h>
#include <l4/re/rm>
#include <l4/re/parent>
#include <l4/re/mem_alloc>
#include <l4/re/log>
#include <l4/re/consts>
#include <l4/re/env.h>
```

Include dependency graph for env:



This graph shows which files directly or indirectly include this file:



### Data Structures

- class [L4Re::Env](#)  
*C++ interface of the initial environment that is provided to an [L4](#) task.*

### Namespaces

- namespace [L4](#)  
*[L4](#) low-level kernel interface.*
- namespace [L4Re](#)  
*[L4Re](#) C++ Interfaces.*

## 17.332.1 Detailed Description

Environment interface.

Definition in file [env](#).

## 17.333 env

[Go to the documentation of this file.](#)

```

00001 // -*- Mode: C++ -*-
00002 // vim:ft=cpp
00003 /*
00008 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00009 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00010 * Björn Döbel <doebel@os.inf.tu-dresden.de>
00011 * economic rights: Technische Universität Dresden (Germany)
00012 *
00013 * License: see LICENSE.spdx (in this directory or the directories above)
00014 */
00015 #pragma once
00016
00017 #include <l4/sys/types.h>
00018
00019 #include <l4/re/rm>
00020 #include <l4/re/parent>
00021 #include <l4/re/mem_alloc>
00022 #include <l4/re/log>
00023 #include <l4/re/consts>
00024
00025 #include <l4/re/env.h>
00026
00027 namespace L4 {
00028 class Scheduler;
00029 }
00030
00034 namespace L4Re
00035 {
00036 class Itas;
00037 struct Dbg_events;
00038
00078 class L4_EXPORT Env
00079 {
00080 private:
00081 l4re_env_t _env;
00082 public:
00083
00087 typedef l4re_env_cap_entry_t Cap_entry;
00088
00096 static Env const *env() noexcept
00097 { return reinterpret_cast<Env*>(l4re_global_env); }
00098
00103 L4::Cap<Parent> parent() const noexcept
00104 { return L4::Cap<Parent>(_env.parent); }
00109 L4::Cap<Mem_alloc> mem_alloc() const noexcept
00110 { return L4::Cap<Mem_alloc>(_env.mem_alloc); }
00114 L4::Cap<L4::Factory> user_factory() const noexcept
00115 { return L4::Cap<L4::Factory>(_env.mem_alloc); }
00120 L4::Cap<Rm> rm() const noexcept
00121 { return L4::Cap<Rm>(_env.rm); }
00126 L4::Cap<Log> log() const noexcept
00127 { return L4::Cap<Log>(_env.log); }
00132 L4::Cap<L4::Thread> main_thread() const noexcept
00133 { return L4::Cap<L4::Thread>(_env.main_thread); }
00138 L4::Cap<L4::Task> task() const noexcept
00139 { return L4::Cap<L4::Task>(L4RE_THIS_TASK_CAP); }
00144 L4::Cap<L4::Factory> factory() const noexcept
00145 { return L4::Cap<L4::Factory>(_env.factory); }
00152 l4_cap_idx_t first_free_cap() const noexcept
00153 { return _env.first_free_cap; }
00158 l4_fpage_t utcb_area() const noexcept
00159 { return _env.utcb_area; }
00167 l4_addr_t first_free_utcb() const noexcept
00168 { return _env.first_free_utcb; }
00169
00174 Cap_entry const *initial_caps() const noexcept
00175 { return _env.caps; }
00176
00185 Cap_entry const *get(char const *name, unsigned l) const noexcept

```



```

00186 { return l4re_env_get_cap_l(name, l, &_env); }
00187
00196 template< typename T >
00197 L4::Cap<T> get_cap(char const *name, unsigned l) const noexcept
00198 {
00199 if (Cap_entry const *e = get(name, l))
00200 return L4::Cap<T>(e->cap);
00201
00202 return L4::Cap<T>(-L4_ENOENT);
00203 }
00204
00211 template< typename T >
00212 L4::Cap<T> get_cap(char const *name) const noexcept
00213 { return get_cap<T>(name, __builtin_strlen(name)); }
00214
00219 void parent(L4::Cap<Parent> const &c) noexcept
00220 { _env.parent = c.cap(); }
00225 void mem_alloc(L4::Cap<Mem_alloc> const &c) noexcept
00226 { _env.mem_alloc = c.cap(); }
00231 void rm(L4::Cap<Rm> const &c) noexcept
00232 { _env.rm = c.cap(); }
00237 void log(L4::Cap<Log> const &c) noexcept
00238 { _env.log = c.cap(); }
00243 void main_thread(L4::Cap<L4::Thread> const &c) noexcept
00244 { _env.main_thread = c.cap(); }
00249 void factory(L4::Cap<L4::Factory> const &c) noexcept
00250 { _env.factory = c.cap(); }
00255 void first_free_cap(l4_cap_idx_t c) noexcept
00256 { _env.first_free_cap = c; }
00261 void utcb_area(l4_fpage_t utcbs) noexcept
00262 { _env.utcb_area = utcbs; }
00267 void first_free_utcb(l4_addr_t u) noexcept
00268 { _env.first_free_utcb = u; }
00269
00275 L4::Cap<L4::Scheduler> scheduler() const noexcept
00276 { return L4::Cap<L4::Scheduler>(_env.scheduler); }
00277
00282 void scheduler(L4::Cap<L4::Scheduler> const &c) noexcept
00283 { _env.scheduler = c.cap(); }
00284
00294 L4::Cap<Itas> itas() const noexcept
00295 { return L4::Cap<Itas>(_env.itas); }
00296
00301 void itas(L4::Cap<Itas> const &c) noexcept
00302 { _env.itas = c.cap(); }
00303
00310 L4::Cap<Dbg_events> dbg_events() const noexcept
00311 { return L4::Cap<Dbg_events>(_env.dbg_events); }
00312
00320 void dbg_events(L4::Cap<Dbg_events> const &dbg_events) noexcept
00321 { _env.dbg_events = dbg_events.cap(); }
00322
00327 void initial_caps(Cap_entry *first) noexcept
00328 { _env.caps = first; }
00329 };
00330 };

```

## 17.334 l4/re/env.h File Reference

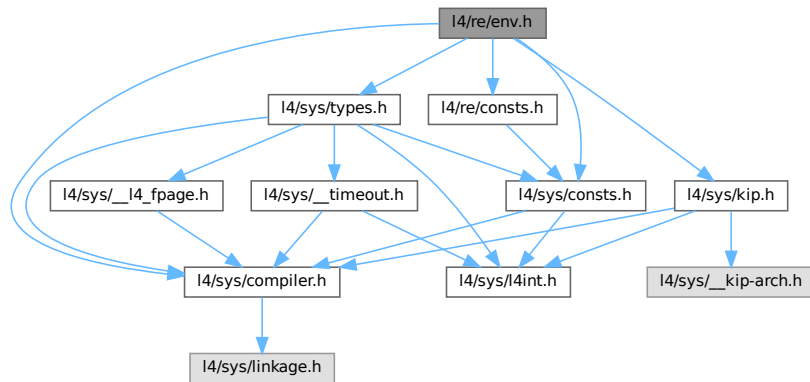
Environment interface.

```

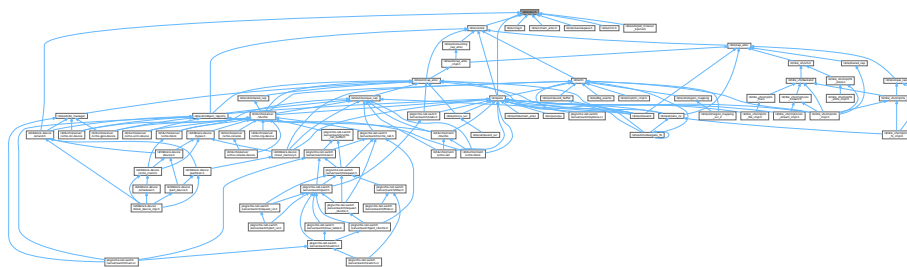
#include <l4/sys/consts.h>
#include <l4/sys/types.h>
#include <l4/sys/kip.h>
#include <l4/sys/compiler.h>
#include <l4/re/consts.h>

```

Include dependency graph for env.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct `l4re_env_cap_entry_t`  
*Entry in the `L4Re` environment array for the named initial objects.*
- struct `l4re_env_t`  
*Initial environment data structure.*

## Typedefs

- `typedef struct l4re_env_cap_entry_t l4re_env_cap_entry_t`  
*Entry in the [L4Re](#) environment array for the named initial objects.*
- `typedef struct l4re_env_t l4re_env_t`  
*Initial environment data structure.*

## Functions

- `l4re_env_t * l4re_env` (void) `L4_NOTHROW`  
*Get L4Re initial environment.*
- `l4_kernel_info_t` const \* `l4re_kip` (void) `L4_NOTHROW`  
*Get Kernel Info Page.*

- [l4\\_cap\\_idx\\_t l4re\\_env\\_get\\_cap](#) (char const \*name) [L4\\_NOTHROW](#)  
*Get the capability selector for the object named name.*
- [l4\\_cap\\_idx\\_t l4re\\_env\\_get\\_cap\\_e](#) (char const \*name, [l4re\\_env\\_t](#) const \*e) [L4\\_NOTHROW](#)  
*Get the capability selector for the object named name.*
- [l4re\\_env\\_cap\\_entry\\_t](#) const \* [l4re\\_env\\_get\\_cap\\_l](#) (char const \*name, unsigned l, [l4re\\_env\\_t](#) const \*e) [L4\\_NOTHROW](#)  
*Get the full [l4re\\_env\\_cap\\_entry\\_t](#) for the object named name.*

## 17.334.1 Detailed Description

Environment interface.

Definition in file [env.h](#).

## 17.334.2 Typedef Documentation

### 17.334.2.1 l4re\_env\_t

```
typedef struct l4re_env_t l4re_env_t
```

Initial environment data structure.

See also

[Initial environment](#)

## 17.335 env.h

[Go to the documentation of this file.](#)

```
00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00014 #include <l4/sys/consts.h>
00015 #include <l4/sys/types.h>
00016 #include <l4/sys/kip.h>
00017 #include <l4/sys/compiler.h>
00018
00019 #include <l4/re/consts.h>
00020
00034
00039 typedef struct l4re_env_cap_entry_t
00040 {
00044 l4_cap_idx_t cap;
00045
00050 l4_umword_t flags;
00051
00055 char name[16];
00056 #ifdef __cplusplus
00057
00061 l4re_env_cap_entry_t() L4_NOTHROW : cap(L4_INVALID_CAP), flags(~0UL) {}
00062
00070 l4re_env_cap_entry_t(char const *n, l4_cap_idx_t c, l4_umword_t f = 0) L4_NOTHROW
00071 : cap(c), flags(f)
00072 {
```

```

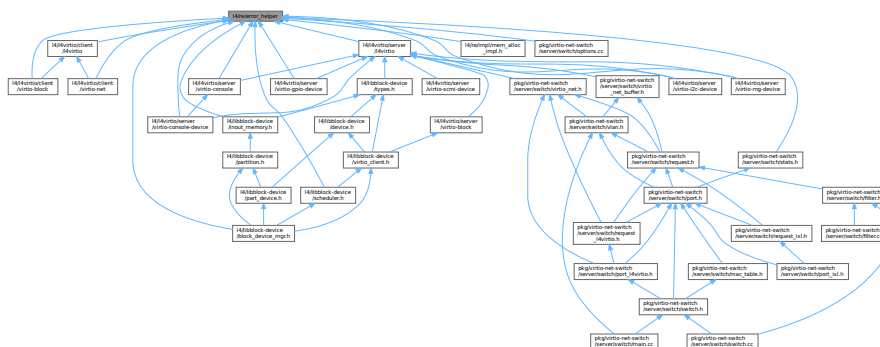
00073 for (unsigned i = 0; n && i < sizeof(name); ++i, ++n)
00074 {
00075 name[i] = *n;
00076 if (!*n)
00077 break;
00078 }
00079 }
00080
00081 static bool is_valid_name(char const *n) L4_NOTHROW
00082 {
00083 for (unsigned i = 0; *n; ++i, ++n)
00084 if (i > sizeof(name))
00085 return false;
00086
00087 return true;
00088 }
00089 #endif
00090 } l4re_env_cap_entry_t;
00091
00092
00093 typedef struct l4re_env_t
00094 {
00095 l4_cap_idx_t parent;
00096 l4_cap_idx_t rm;
00097 l4_cap_idx_t mem_alloc;
00098 l4_cap_idx_t log;
00099 l4_cap_idx_t main_thread;
00100 l4_cap_idx_t factory;
00101 l4_cap_idx_t scheduler;
00102 l4_cap_idx_t itas;
00103 l4_cap_idx_t dbg_events;
00104 l4_cap_idx_t first_free_cap;
00105 l4_fpage_t utcb_area;
00106 l4_addr_t first_free_utcb;
00107 l4re_env_cap_entry_t *caps;
00108 } l4re_env_t;
00109
00110 extern l4re_env_t *l4re_global_env;
00111
00112
00113 L4_INLINE l4re_env_t *l4re_env(void) L4_NOTHROW;
00114
00115 /*
00116 * FIXME: this seems to be at the wrong place here
00117 */
00118 L4_INLINE l4_kernel_info_t const *l4re_kip(void) L4_NOTHROW;
00119
00120
00121 L4_INLINE l4_cap_idx_t
00122 l4re_env_get_cap(char const *name) L4_NOTHROW;
00123
00124 L4_INLINE l4_cap_idx_t
00125 l4re_env_get_cap_e(char const *name, l4re_env_t const *e) L4_NOTHROW;
00126
00127 L4_INLINE l4re_env_cap_entry_t const *
00128 l4re_env_get_cap_l(char const *name, unsigned l, l4re_env_t const *e) L4_NOTHROW;
00129
00130 L4_INLINE
00131 l4re_env_t *l4re_env(void) L4_NOTHROW
00132 { return l4re_global_env; }
00133
00134 L4_INLINE
00135 l4_kernel_info_t const *l4re_kip(void) L4_NOTHROW
00136 { return l4_kip(); }
00137
00138 L4_INLINE l4re_env_cap_entry_t const *
00139 l4re_env_get_cap_l(char const *name, unsigned l, l4re_env_t const *e) L4_NOTHROW
00140 {
00141 l4re_env_cap_entry_t const *c = e->caps;
00142 for (; c && c->flags != ~0UL; ++c)
00143 {
00144 unsigned i;
00145 for (i = 0;
00146 i < sizeof(c->name) && i < l && c->name[i] && name[i] && name[i] == c->name[i];
00147 ++i)
00148 ;
00149
00150 if (i == l && (i == sizeof(c->name) || !c->name[i]))
00151 return c;
00152 }
00153 return NULL;
00154 }
00155
00156 L4_INLINE l4_cap_idx_t
00157 l4re_env_get_cap_e(char const *name, l4re_env_t const *e) L4_NOTHROW
00158 {
00159 unsigned l;

```

### 17.336 l4/re/error\_helper File Reference

```
#include <l4/sys/types.h>
#include <l4/cxx/exceptions>
#include <l4/cxx/type_traits>
#include <l4/sys/err.h>
#include <stdarg.h>
#include <stdio.h>
```

Include dependency graph for error\_helper:



## Namespaces

- namespace [L4Re](#)  
*L4Re C++ Interfaces.*

## Functions

- void [L4Re::throw\\_error](#) (long err, char const \*extra="")  
*Generate C++ exception.*
- long [L4Re::chksys](#) (long err, char const \*extra="", long ret=0)  
*Generate C++ exception on error.*
- long [L4Re::chksys](#) ([l4\\_msgtag\\_t](#) const &t, char const \*extra="", [l4\\_utcb\\_t](#) \*utcb=[l4\\_utcb](#)(), long ret=0)  
*Generate C++ exception on error.*
- long [L4Re::chksys](#) ([l4\\_msgtag\\_t](#) const &t, [l4\\_utcb\\_t](#) \*utcb, char const \*extra="")  
*Generate C++ exception on error.*
- template<typename T>  
T [L4Re::chkcap](#) (T &&cap, char const \*extra="", long err=[L4\\_ENOMEM](#))  
*Check for valid capability or raise C++ exception.*
- [l4\\_msgtag\\_t](#) [L4Re::chkipc](#) ([l4\\_msgtag\\_t](#) tag, char const \*extra="", [l4\\_utcb\\_t](#) \*utcb=[l4\\_utcb](#)())  
*Test a message tag for IPC errors.*

### 17.336.1 Detailed Description

Error helper.

Definition in file [error\\_helper](#).

## 17.337 error\_helper

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -- Mode: C++ --
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00005 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00006 * economic rights: Technische Universität Dresden (Germany)
00007 *
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010 #pragma once
00011
00012 #include <l4/sys/types.h>
00013 #include <l4/cxx/exceptions>
00014 #include <l4/cxx/type_traits>
00015 #include <l4/sys/err.h>
00016
00017 #include <stdarg.h>
00018 #include <stdio.h>
00019
00020 namespace L4Re {
00021 #ifdef __EXCEPTIONS
00022 [[noreturn]] inline void throw_error(long err, char const *extra = "")
00023 {
00024 switch (err)
00025 {
00026 case -L4_ENOENT: throw (L4::Element_not_found(extra));
00027 case -L4_ENOMEM: throw (L4::Out_of_memory(extra));
00028 case -L4_EEXIST: throw (L4::Element_already_exists(extra));
00029 case -L4_ERANGE: throw (L4::Bounds_error(extra));
00030 }
00031 }

```

```

00045 default: throw (L4::Runtime_error(err, extra));
00046 }
00047 }
00048
00049 [[noreturn]] inline void throw_error_fmt(long err, char const *const fmt, ...)
00050 __attribute__((format(printf, 2, 3)));
00051 [[noreturn]] inline void throw_error_fmt(long err, char const *const fmt, ...)
00052 {
00053 char extra[80];
00054 va_list argp;
00055 va_start(argp, fmt);
00056 vsnprintf(extra, sizeof(extra), fmt, argp);
00057 va_end(argp);
00058 throw_error(err, extra);
00059 }
00060
00071 inline
00072 long chksys(long err, char const *extra = "", long ret = 0)
00073 {
00074 if (L4_UNLIKELY(err < 0))
00075 throw_error(ret ? ret : err, extra);
00076 return err;
00077 }
00078
00092 inline
00093 long chksys(l4_msgtag_t const &t, char const *extra = "",
00094 l4_utcb_t *utcb = l4_utcb(), long ret = 0)
00095 {
00096 if (L4_UNLIKELY(t.has_error()))
00097 throw_error(ret ? ret : l4_error_u(t, utcb), extra);
00098 else if (L4_UNLIKELY(t.label() < 0))
00099 throw_error(ret ? ret: t.label(), extra);
00100 return t.label();
00101 }
00102
00115 inline
00116 long chksys(l4_msgtag_t const &t, l4_utcb_t *utcb, char const *extra = "")
00117 { return chksys(t, extra, utcb); }
00118
00119 #if 0
00120 inline
00121 long chksys(long ret, long err, char const *extra = "")
00122 {
00123 if (L4_UNLIKELY(ret < 0))
00124 throw_error(err, extra);
00125 return ret;
00126 }
00127 #endif
00128
00146 template<typename T>
00147 inline
00148 #if __cplusplus >= 201103L
00149 T chkcap(T &&cap, char const *extra = "", long err = -L4_ENOMEM)
00150 #else
00151 T chkcap(T cap, char const *extra = "", long err = -L4_ENOMEM)
00152 #endif
00153 {
00154 if (L4_UNLIKELY(!cap.is_valid()))
00155 throw_error(err ? err : cap.invalid_cap_error(), extra);
00156 #if __cplusplus >= 201103L
00157 return cxx::forward<T>(cap);
00158 #else
00159 return cap;
00160 #endif
00161 }
00162
00178 inline
00179 l4_msgtag_t
00180 chkipc(l4_msgtag_t tag, char const *extra = "",
00181 l4_utcb_t *utcb = l4_utcb())
00182 {
00183 if (L4_UNLIKELY(tag.has_error()))
00184 chksys(l4_error_u(tag, utcb), extra);
00185 return tag;
00186 }
00187
00188 #endif
00189
00190 }

```

## 17.338 event-sys.h

```

00001 /*
00002 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009 namespace L4Re
00010 {
00011 namespace Event_
00012 {
00013 enum Opcodes
00014 {
00015 Get, Get_num_streams, Get_stream_info, Get_stream_info_for_id,
00016 Get_axis_info, Get_stream_state_for_id
00017 };
00018 };
00019 };
00020
```

## 17.339 event\_enums.h

```

00001 #pragma once
00002
00003 /*
00004 *
00005 *
00006 * Constants for L4Re events ...
00007 */
00008
00009
00010 enum L4Re_events_key
00011 {
00012 L4RE_KEY_RESERVED = 0,
00013 L4RE_KEY_ESC = 1,
00014 L4RE_KEY_1 = 2,
00015 L4RE_KEY_2 = 3,
00016 L4RE_KEY_3 = 4,
00017 L4RE_KEY_4 = 5,
00018 L4RE_KEY_5 = 6,
00019 L4RE_KEY_6 = 7,
00020 L4RE_KEY_7 = 8,
00021 L4RE_KEY_8 = 9,
00022 L4RE_KEY_9 = 10,
00023 L4RE_KEY_0 = 11,
00024 L4RE_KEY_MINUS = 12,
00025 L4RE_KEY_EQUAL = 13,
00026 L4RE_KEY_BACKSPACE = 14,
00027 L4RE_KEY_TAB = 15,
00028 L4RE_KEY_Q = 16,
00029 L4RE_KEY_W = 17,
00030 L4RE_KEY_E = 18,
00031 L4RE_KEY_R = 19,
00032 L4RE_KEY_T = 20,
00033 L4RE_KEY_Y = 21,
00034 L4RE_KEY_U = 22,
00035 L4RE_KEY_I = 23,
00036 L4RE_KEY_O = 24,
00037 L4RE_KEY_P = 25,
00038 L4RE_KEY_LEFTBRACE = 26,
00039 L4RE_KEY_RIGHTBRACE = 27,
00040 L4RE_KEY_ENTER = 28,
00041 L4RE_KEY_LEFTCTRL = 29,
00042 L4RE_KEY_A = 30,
00043 L4RE_KEY_S = 31,
00044 L4RE_KEY_D = 32,
00045 L4RE_KEY_F = 33,
00046 L4RE_KEY_G = 34,
00047 L4RE_KEY_H = 35,
00048 L4RE_KEY_J = 36,
00049 L4RE_KEY_K = 37,
00050 L4RE_KEY_L = 38,
00051 L4RE_KEY_SEMICOLON = 39,
00052 L4RE_KEY_APOSTROPHE = 40,
00053 L4RE_KEY_GRAVE = 41,
00054 L4RE_KEY_LEFTSHIFT = 42,
00055 L4RE_KEY_BACKSLASH = 43,
00056 L4RE_KEY_Z = 44,
00057 L4RE_KEY_X = 45,
00058 L4RE_KEY_C = 46,

```



```
00059 L4RE_KEY_V = 47,
00060 L4RE_KEY_B = 48,
00061 L4RE_KEY_N = 49,
00062 L4RE_KEY_M = 50,
00063 L4RE_KEY_COMMA = 51,
00064 L4RE_KEY_DOT = 52,
00065 L4RE_KEY_SLASH = 53,
00066 L4RE_KEY_RIGHTSHIFT = 54,
00067 L4RE_KEY_KPASTERISK = 55,
00068 L4RE_KEY_LEFTALT = 56,
00069 L4RE_KEY_SPACE = 57,
00070 L4RE_KEY_CAPSLOCK = 58,
00071 L4RE_KEY_F1 = 59,
00072 L4RE_KEY_F2 = 60,
00073 L4RE_KEY_F3 = 61,
00074 L4RE_KEY_F4 = 62,
00075 L4RE_KEY_F5 = 63,
00076 L4RE_KEY_F6 = 64,
00077 L4RE_KEY_F7 = 65,
00078 L4RE_KEY_F8 = 66,
00079 L4RE_KEY_F9 = 67,
00080 L4RE_KEY_F10 = 68,
00081 L4RE_KEY_NUMLOCK = 69,
00082 L4RE_KEY_SCROLLLOCK = 70,
00083 L4RE_KEY_KP7 = 71,
00084 L4RE_KEY_KP8 = 72,
00085 L4RE_KEY_KP9 = 73,
00086 L4RE_KEY_KPMINUS = 74,
00087 L4RE_KEY_KP4 = 75,
00088 L4RE_KEY_KP5 = 76,
00089 L4RE_KEY_KP6 = 77,
00090 L4RE_KEY_KPPLUS = 78,
00091 L4RE_KEY_KP1 = 79,
00092 L4RE_KEY_KP2 = 80,
00093 L4RE_KEY_KP3 = 81,
00094 L4RE_KEY_KP0 = 82,
00095 L4RE_KEY_KPDOT = 83,
00096 L4RE_KEY_ZENKAKUHANKAKU = 85,
00097 L4RE_KEY_102ND = 86,
00098 L4RE_KEY_F11 = 87,
00099 L4RE_KEY_F12 = 88,
00100 L4RE_KEY_RO = 89,
00101 L4RE_KEY_KATAKANA = 90,
00102 L4RE_KEY_HIRAGANA = 91,
00103 L4RE_KEY_HENKAN = 92,
00104 L4RE_KEY_KATAKANAHIRAGANA = 93,
00105 L4RE_KEY_MUHENKAN = 94,
00106 L4RE_KEY_KPJPCOMMA = 95,
00107 L4RE_KEY_KPENTER = 96,
00108 L4RE_KEY_RIGHTCTRL = 97,
00109 L4RE_KEY_KPSLASH = 98,
00110 L4RE_KEY_SYSRQ = 99,
00111 L4RE_KEY_RIGHTALT = 100,
00112 L4RE_KEY_LINEFEED = 101,
00113 L4RE_KEY_HOME = 102,
00114 L4RE_KEY_UP = 103,
00115 L4RE_KEY_PAGEUP = 104,
00116 L4RE_KEY_LEFT = 105,
00117 L4RE_KEY_RIGHT = 106,
00118 L4RE_KEY_END = 107,
00119 L4RE_KEY_DOWN = 108,
00120 L4RE_KEY_PAGEDOWN = 109,
00121 L4RE_KEY_INSERT = 110,
00122 L4RE_KEY_DELETE = 111,
00123 L4RE_KEY_MACRO = 112,
00124 L4RE_KEY_MUTE = 113,
00125 L4RE_KEY_VOLUMEDOWN = 114,
00126 L4RE_KEY_VOLUMEUP = 115,
00127 L4RE_KEY_POWER = 116,
00128 L4RE_KEY_KPEQUAL = 117,
00129 L4RE_KEY_KPPLUSMINUS = 118,
00130 L4RE_KEY_PAUSE = 119,
00131 L4RE_KEY_KPCOMMA = 121,
00132 L4RE_KEY_HANGEUL = 122,
00133 L4RE_KEY_HANGUEL = L4RE_KEY_HANGEUL,
00134 L4RE_KEY_HANJA = 123,
00135 L4RE_KEY_YEN = 124,
00136 L4RE_KEY_LEFTMETA = 125,
00137 L4RE_KEY_RIGHTMETA = 126,
00138 L4RE_KEY_COMPOSE = 127,
00139 L4RE_KEY_STOP = 128,
00140 L4RE_KEY_AGAIN = 129,
00141 L4RE_KEY_PROPS = 130,
00142 L4RE_KEY_UNDO = 131,
00143 L4RE_KEY_FRONT = 132,
00144 L4RE_KEY_COPY = 133,
00145 L4RE_KEY_OPEN = 134,
```

|       |                          |        |
|-------|--------------------------|--------|
| 00146 | L4RE_KEY_PASTE           | = 135, |
| 00147 | L4RE_KEY_FIND            | = 136, |
| 00148 | L4RE_KEY_CUT             | = 137, |
| 00149 | L4RE_KEY_HELP            | = 138, |
| 00150 | L4RE_KEY_MENU            | = 139, |
| 00151 | L4RE_KEY_CALC            | = 140, |
| 00152 | L4RE_KEY_SETUP           | = 141, |
| 00153 | L4RE_KEY_SLEEP           | = 142, |
| 00154 | L4RE_KEY_WAKEUP          | = 143, |
| 00155 | L4RE_KEY_FILE            | = 144, |
| 00156 | L4RE_KEY_SENDFILE        | = 145, |
| 00157 | L4RE_KEY_DELETEFILE      | = 146, |
| 00158 | L4RE_KEY_XFER            | = 147, |
| 00159 | L4RE_KEY_PROG1           | = 148, |
| 00160 | L4RE_KEY_PROG2           | = 149, |
| 00161 | L4RE_KEY_WWW             | = 150, |
| 00162 | L4RE_KEY_MSDOS           | = 151, |
| 00163 | L4RE_KEY_COFFEE          | = 152, |
| 00164 | L4RE_KEY_DIRECTION       | = 153, |
| 00165 | L4RE_KEY_CYCLEWINDOWS    | = 154, |
| 00166 | L4RE_KEY_MAIL            | = 155, |
| 00167 | L4RE_KEY_BOOKMARKS       | = 156, |
| 00168 | L4RE_KEY_COMPUTER        | = 157, |
| 00169 | L4RE_KEY_BACK            | = 158, |
| 00170 | L4RE_KEY_FORWARD         | = 159, |
| 00171 | L4RE_KEY_CLOSECD         | = 160, |
| 00172 | L4RE_KEY_EJECTCD         | = 161, |
| 00173 | L4RE_KEY_EJECTCLOSECD    | = 162, |
| 00174 | L4RE_KEY_NEXTSONG        | = 163, |
| 00175 | L4RE_KEY_PLAYPAUSE       | = 164, |
| 00176 | L4RE_KEY_PREVIOUSSONG    | = 165, |
| 00177 | L4RE_KEY_STOPCD          | = 166, |
| 00178 | L4RE_KEY_RECORD          | = 167, |
| 00179 | L4RE_KEY_REWIND          | = 168, |
| 00180 | L4RE_KEY_PHONE           | = 169, |
| 00181 | L4RE_KEY_ISO             | = 170, |
| 00182 | L4RE_KEY_CONFIG          | = 171, |
| 00183 | L4RE_KEY_HOMEPAGE        | = 172, |
| 00184 | L4RE_KEY_REFRESH         | = 173, |
| 00185 | L4RE_KEY_EXIT            | = 174, |
| 00186 | L4RE_KEY_MOVE            | = 175, |
| 00187 | L4RE_KEY_EDIT            | = 176, |
| 00188 | L4RE_KEY_SCROLLUP        | = 177, |
| 00189 | L4RE_KEY_SCROLLDOWN      | = 178, |
| 00190 | L4RE_KEY_KPLEFTPAREN     | = 179, |
| 00191 | L4RE_KEY_KPRIGHTPAREN    | = 180, |
| 00192 | L4RE_KEY_NEW             | = 181, |
| 00193 | L4RE_KEY_REDO            | = 182, |
| 00194 | L4RE_KEY_F13             | = 183, |
| 00195 | L4RE_KEY_F14             | = 184, |
| 00196 | L4RE_KEY_F15             | = 185, |
| 00197 | L4RE_KEY_F16             | = 186, |
| 00198 | L4RE_KEY_F17             | = 187, |
| 00199 | L4RE_KEY_F18             | = 188, |
| 00200 | L4RE_KEY_F19             | = 189, |
| 00201 | L4RE_KEY_F20             | = 190, |
| 00202 | L4RE_KEY_F21             | = 191, |
| 00203 | L4RE_KEY_F22             | = 192, |
| 00204 | L4RE_KEY_F23             | = 193, |
| 00205 | L4RE_KEY_F24             | = 194, |
| 00206 | L4RE_KEY_PLAYCD          | = 200, |
| 00207 | L4RE_KEY_PAUSECD         | = 201, |
| 00208 | L4RE_KEY_PROG3           | = 202, |
| 00209 | L4RE_KEY_PROG4           | = 203, |
| 00210 | L4RE_KEY_SUSPEND         | = 205, |
| 00211 | L4RE_KEY_CLOSE           | = 206, |
| 00212 | L4RE_KEY_PLAY            | = 207, |
| 00213 | L4RE_KEY_FASTFORWARD     | = 208, |
| 00214 | L4RE_KEY_BASSBOOST       | = 209, |
| 00215 | L4RE_KEY_PRINT           | = 210, |
| 00216 | L4RE_KEY_HP              | = 211, |
| 00217 | L4RE_KEY_CAMERA          | = 212, |
| 00218 | L4RE_KEY_SOUND           | = 213, |
| 00219 | L4RE_KEY_QUESTION        | = 214, |
| 00220 | L4RE_KEY_EMAIL           | = 215, |
| 00221 | L4RE_KEY_CHAT            | = 216, |
| 00222 | L4RE_KEY_SEARCH          | = 217, |
| 00223 | L4RE_KEY_CONNECT         | = 218, |
| 00224 | L4RE_KEY_FINANCE         | = 219, |
| 00225 | L4RE_KEY_SPORT           | = 220, |
| 00226 | L4RE_KEY_SHOP            | = 221, |
| 00227 | L4RE_KEY_ALTERASE        | = 222, |
| 00228 | L4RE_KEY_CANCEL          | = 223, |
| 00229 | L4RE_KEY_BRIGHTNESSDOWN  | = 224, |
| 00230 | L4RE_KEY_BRIGHTNESSUP    | = 225, |
| 00231 | L4RE_KEY_MEDIA           | = 226, |
| 00232 | L4RE_KEY_SWITCHVIDEOMODE | = 227, |

```
00233 L4RE_KEY_KBDILLUMTOGGLE = 228,
00234 L4RE_KEY_KBDILLUMDOWN = 229,
00235 L4RE_KEY_KBDILLUMUP = 230,
00236 L4RE_KEY_SEND = 231,
00237 L4RE_KEY_REPLY = 232,
00238 L4RE_KEY_FORWARDMAIL = 233,
00239 L4RE_KEY_SAVE = 234,
00240 L4RE_KEY_DOCUMENTS = 235,
00241 L4RE_KEY_UNKNOWN = 240,
00242 L4RE_KEY_OK = 0x160,
00243 L4RE_KEY_SELECT = 0x161,
00244 L4RE_KEY_GOTO = 0x162,
00245 L4RE_KEY_CLEAR = 0x163,
00246 L4RE_KEY_POWER2 = 0x164,
00247 L4RE_KEY_OPTION = 0x165,
00248 L4RE_KEY_INFO = 0x166,
00249 L4RE_KEY_TIME = 0x167,
00250 L4RE_KEY_VENDOR = 0x168,
00251 L4RE_KEY_ARCHIVE = 0x169,
00252 L4RE_KEY_PROGRAM = 0x16a,
00253 L4RE_KEY_CHANNEL = 0x16b,
00254 L4RE_KEY_FAVORITES = 0x16c,
00255 L4RE_KEY_EPG = 0x16d,
00256 L4RE_KEY_PVR = 0x16e,
00257 L4RE_KEY_MHP = 0x16f,
00258 L4RE_KEY_LANGUAGE = 0x170,
00259 L4RE_KEY_TITLE = 0x171,
00260 L4RE_KEY_SUBTITLE = 0x172,
00261 L4RE_KEY_ANGLE = 0x173,
00262 L4RE_KEY_ZOOM = 0x174,
00263 L4RE_KEY_MODE = 0x175,
00264 L4RE_KEY_KEYBOARD = 0x176,
00265 L4RE_KEY_SCREEN = 0x177,
00266 L4RE_KEY_PC = 0x178,
00267 L4RE_KEY_TV = 0x179,
00268 L4RE_KEY_TV2 = 0x17a,
00269 L4RE_KEY_VCR = 0x17b,
00270 L4RE_KEY_VCR2 = 0x17c,
00271 L4RE_KEY_SAT = 0x17d,
00272 L4RE_KEY_SAT2 = 0x17e,
00273 L4RE_KEY_CD = 0x17f,
00274 L4RE_KEY_TAPE = 0x180,
00275 L4RE_KEY_RADIO = 0x181,
00276 L4RE_KEY_TUNER = 0x182,
00277 L4RE_KEY_PLAYER = 0x183,
00278 L4RE_KEY_TEXT = 0x184,
00279 L4RE_KEY_DVD = 0x185,
00280 L4RE_KEY_AUX = 0x186,
00281 L4RE_KEY_MP3 = 0x187,
00282 L4RE_KEY_AUDIO = 0x188,
00283 L4RE_KEY_VIDEO = 0x189,
00284 L4RE_KEY_DIRECTORY = 0x18a,
00285 L4RE_KEY_LIST = 0x18b,
00286 L4RE_KEY_MEMO = 0x18c,
00287 L4RE_KEY_CALENDAR = 0x18d,
00288 L4RE_KEY_RED = 0x18e,
00289 L4RE_KEY_GREEN = 0x18f,
00290 L4RE_KEY_YELLOW = 0x190,
00291 L4RE_KEY_BLUE = 0x191,
00292 L4RE_KEY_CHANNELUP = 0x192,
00293 L4RE_KEY_CHANNELDOWN = 0x193,
00294 L4RE_KEY_FIRST = 0x194,
00295 L4RE_KEY_LAST = 0x195,
00296 L4RE_KEY_AB = 0x196,
00297 L4RE_KEY_NEXT = 0x197,
00298 L4RE_KEY_RESTART = 0x198,
00299 L4RE_KEY_SLOW = 0x199,
00300 L4RE_KEY_SHUFFLE = 0x19a,
00301 L4RE_KEY_BREAK = 0x19b,
00302 L4RE_KEY_PREVIOUS = 0x19c,
00303 L4RE_KEY_DIGITS = 0x19d,
00304 L4RE_KEY_TEEN = 0x19e,
00305 L4RE_KEY_TWEN = 0x19f,
00306 L4RE_KEY_DEL_EOL = 0x1c0,
00307 L4RE_KEY_DEL_EOS = 0x1c1,
00308 L4RE_KEY_INS_LINE = 0x1c2,
00309 L4RE_KEY_DEL_LINE = 0x1c3,
00310 L4RE_KEY_FN = 0x1d0,
00311 L4RE_KEY_FN_ESC = 0x1d1,
00312 L4RE_KEY_FN_F1 = 0x1d2,
00313 L4RE_KEY_FN_F2 = 0x1d3,
00314 L4RE_KEY_FN_F3 = 0x1d4,
00315 L4RE_KEY_FN_F4 = 0x1d5,
00316 L4RE_KEY_FN_F5 = 0x1d6,
00317 L4RE_KEY_FN_F6 = 0x1d7,
00318 L4RE_KEY_FN_F7 = 0x1d8,
00319 L4RE_KEY_FN_F8 = 0x1d9,
```

```

00320 L4RE_KEY_FN_F9 = 0x1da,
00321 L4RE_KEY_FN_F10 = 0x1db,
00322 L4RE_KEY_FN_F11 = 0x1dc,
00323 L4RE_KEY_FN_F12 = 0x1dd,
00324 L4RE_KEY_FN_1 = 0x1de,
00325 L4RE_KEY_FN_2 = 0x1df,
00326 L4RE_KEY_FN_D = 0x1e0,
00327 L4RE_KEY_FN_E = 0x1e1,
00328 L4RE_KEY_FN_F = 0x1e2,
00329 L4RE_KEY_FN_S = 0x1e3,
00330 L4RE_KEY_FN_B = 0x1e4,
00331 L4RE_KEY_MAX = 0x1ff,
00332 };
00333
00334 enum L4Re_events_rel
00335 {
00336 L4RE_REL_X = 0x00,
00337 L4RE_REL_Y = 0x01,
00338 L4RE_REL_Z = 0x02,
00339 L4RE_REL_RX = 0x03,
00340 L4RE_REL_RY = 0x04,
00341 L4RE_REL_RZ = 0x05,
00342 L4RE_REL_HWHEEL = 0x06,
00343 L4RE_REL_DIAL = 0x07,
00344 L4RE_REL_WHEEL = 0x08,
00345 L4RE_REL_MISC = 0x09,
00346 L4RE_REL_MAX = 0x0f,
00347 };
00348
00349 enum L4Re_events_snd
00350 {
00351 L4RE_SND_CLICK = 0x00,
00352 L4RE_SND_BELL = 0x01,
00353 L4RE_SND_TONE = 0x02,
00354 L4RE_SND_MAX = 0x07,
00355 };
00356
00357 enum L4Re_events_rep
00358 {
00359 L4RE_REP_DELAY = 0x00,
00360 L4RE_REP_PERIOD = 0x01,
00361 L4RE_REP_MAX = 0x01,
00362 };
00363
00364 enum L4Re_events_led
00365 {
00366 L4RE_LED_NUML = 0x00,
00367 L4RE_LED_CAPSL = 0x01,
00368 L4RE_LED_SCROLLL = 0x02,
00369 L4RE_LED_COMPOSE = 0x03,
00370 L4RE_LED_KANA = 0x04,
00371 L4RE_LED_SLEEP = 0x05,
00372 L4RE_LED_SUSPEND = 0x06,
00373 L4RE_LED_MUTE = 0x07,
00374 L4RE_LED_MISC = 0x08,
00375 L4RE_LED_MAIL = 0x09,
00376 L4RE_LED_CHARGING = 0x0a,
00377 L4RE_LED_MAX = 0x0f,
00378 };
00379
00380 enum L4Re_events_btn
00381 {
00382 L4RE_BTN_MISC = 0x100,
00383 L4RE_BTN_0 = 0x100,
00384 L4RE_BTN_1 = 0x101,
00385 L4RE_BTN_2 = 0x102,
00386 L4RE_BTN_3 = 0x103,
00387 L4RE_BTN_4 = 0x104,
00388 L4RE_BTN_5 = 0x105,
00389 L4RE_BTN_6 = 0x106,
00390 L4RE_BTN_7 = 0x107,
00391 L4RE_BTN_8 = 0x108,
00392 L4RE_BTN_9 = 0x109,
00393 L4RE_BTN_MOUSE = 0x110,
00394 L4RE_BTN_LEFT = 0x110,
00395 L4RE_BTN_RIGHT = 0x111,
00396 L4RE_BTN_MIDDLE = 0x112,
00397 L4RE_BTN_SIDE = 0x113,
00398 L4RE_BTN_EXTRA = 0x114,
00399 L4RE_BTN_FORWARD = 0x115,
00400 L4RE_BTN_BACK = 0x116,
00401 L4RE_BTN_TASK = 0x117,
00402 L4RE_BTN_JOYSTICK = 0x120,
00403 L4RE_BTN_TRIGGER = 0x120,
00404 L4RE_BTN_THUMB = 0x121,
00405 L4RE_BTN_THUMB2 = 0x122,
00406 L4RE_BTN_TOP = 0x123,

```

```

00407 L4RE_BTN_TOP2 = 0x124,
00408 L4RE_BTN_PINKIE = 0x125,
00409 L4RE_BTN_BASE = 0x126,
00410 L4RE_BTN_BASE2 = 0x127,
00411 L4RE_BTN_BASE3 = 0x128,
00412 L4RE_BTN_BASE4 = 0x129,
00413 L4RE_BTN_BASE5 = 0x12a,
00414 L4RE_BTN_BASE6 = 0x12b,
00415 L4RE_BTN_DEAD = 0x12f,
00416 L4RE_BTN_GAMEPAD = 0x130,
00417 L4RE_BTN_A = 0x130,
00418 L4RE_BTN_B = 0x131,
00419 L4RE_BTN_C = 0x132,
00420 L4RE_BTN_X = 0x133,
00421 L4RE_BTN_Y = 0x134,
00422 L4RE_BTN_Z = 0x135,
00423 L4RE_BTN_TL = 0x136,
00424 L4RE_BTN_TR = 0x137,
00425 L4RE_BTN_TL2 = 0x138,
00426 L4RE_BTN_TR2 = 0x139,
00427 L4RE_BTN_SELECT = 0x13a,
00428 L4RE_BTN_START = 0x13b,
00429 L4RE_BTN_MODE = 0x13c,
00430 L4RE_BTN_THUMBL = 0x13d,
00431 L4RE_BTN_THUMBR = 0x13e,
00432 L4RE_BTN_DIGI = 0x140,
00433 L4RE_BTN_TOOL_PEN = 0x140,
00434 L4RE_BTN_TOOL_RUBBER = 0x141,
00435 L4RE_BTN_TOOL_BRUSH = 0x142,
00436 L4RE_BTN_TOOL_PENCIL = 0x143,
00437 L4RE_BTN_TOOL_AIRBRUSH = 0x144,
00438 L4RE_BTN_TOOL_FINGER = 0x145,
00439 L4RE_BTN_TOOL_MOUSE = 0x146,
00440 L4RE_BTN_TOOL_LENS = 0x147,
00441 L4RE_BTN_TOUCH = 0x14a,
00442 L4RE_BTN_STYLUS = 0x14b,
00443 L4RE_BTN_STYLUS2 = 0x14c,
00444 L4RE_BTN_TOOL_DOUBLETAP = 0x14d,
00445 L4RE_BTN_TOOL_TRIPLETAP = 0x14e,
00446 L4RE_BTN_WHEEL = 0x150,
00447 L4RE_BTN_GEAR_DOWN = 0x150,
00448 L4RE_BTN_GEAR_UP = 0x151,
00449 };
00450
00451 enum L4Re_events_sw
00452 {
00453 L4RE_SW_0 = 0x00,
00454 L4RE_SW_1 = 0x01,
00455 L4RE_SW_2 = 0x02,
00456 L4RE_SW_3 = 0x03,
00457 L4RE_SW_4 = 0x04,
00458 L4RE_SW_5 = 0x05,
00459 L4RE_SW_6 = 0x06,
00460 L4RE_SW_7 = 0x07,
00461 L4RE_SW_MAX = 0x0f,
00462 };
00463
00464 enum L4Re_events_ev
00465 {
00466 L4RE_EV_SYN = 0x00,
00467 L4RE_EV_KEY = 0x01,
00468 L4RE_EV_REL = 0x02,
00469 L4RE_EV_ABS = 0x03,
00470 L4RE_EV_MSC = 0x04,
00471 L4RE_EV_SW = 0x05,
00472 L4RE_EV_LED = 0x11,
00473 L4RE_EV_SND = 0x12,
00474 L4RE_EV_REP = 0x14,
00475 L4RE_EV_FF = 0x15,
00476 L4RE_EV_PWR = 0x16,
00477 L4RE_EV_FF_STATUS = 0x17,
00478 L4RE_EV_WINDOW = 0x18,
00479 L4RE_EV_PM = 0x1e, // power management signals
00480 L4RE_EV_MAX = 0x1f,
00481 };
00482
00483 enum L4Re_events_syn
00484 {
00485 L4RE_SYN_REPORT = 0,
00486 L4RE_SYN_CONFIG = 1,
00487 L4RE_SYN_MT_REPORT = 2,
00488
00489 L4RE_SYN_STREAM_CFG = 0x80,
00490 };
00491
00492 enum L4Re_stream_cfg
00493 {

```

```

00494 L4RE_SYN_STREAM_NEW = 0,
00495 L4RE_SYN_STREAM_CLOSE = 1,
00496 };
00497
00498 enum L4Re_events_abs
00499 {
00500 L4RE_ABS_X = 0x00,
00501 L4RE_ABS_Y = 0x01,
00502 L4RE_ABS_Z = 0x02,
00503 L4RE_ABS_RX = 0x03,
00504 L4RE_ABS_RY = 0x04,
00505 L4RE_ABS_RZ = 0x05,
00506 L4RE_ABS_THROTTLE = 0x06,
00507 L4RE_ABS_RUDDER = 0x07,
00508 L4RE_ABS_WHEEL = 0x08,
00509 L4RE_ABS_GAS = 0x09,
00510 L4RE_ABS_BRAKE = 0x0a,
00511 L4RE_ABS_HAT0X = 0x10,
00512 L4RE_ABS_HAT0Y = 0x11,
00513 L4RE_ABS_HAT1X = 0x12,
00514 L4RE_ABS_HAT1Y = 0x13,
00515 L4RE_ABS_HAT2X = 0x14,
00516 L4RE_ABS_HAT2Y = 0x15,
00517 L4RE_ABS_HAT3X = 0x16,
00518 L4RE_ABS_HAT3Y = 0x17,
00519 L4RE_ABS_PRESSURE = 0x18,
00520 L4RE_ABS_DISTANCE = 0x19,
00521 L4RE_ABS_TILT_X = 0x1a,
00522 L4RE_ABS_TILT_Y = 0x1b,
00523 L4RE_ABS_TOOL_WIDTH = 0x1c,
00524 L4RE_ABS_VOLUME = 0x20,
00525 L4RE_ABS_MISC = 0x28,
00526 L4RE_ABS_MT_TOUCH_MAJOR = 0x30,
00527 L4RE_ABS_MT_TOUCH_MINOR = 0x31,
00528 L4RE_ABS_MT_WIDTH_MAJOR = 0x32,
00529 L4RE_ABS_MT_WIDTH_MINOR = 0x33,
00530 L4RE_ABS_MT_ORIENTATION = 0x34,
00531 L4RE_ABS_MT_POSITION_X = 0x35,
00532 L4RE_ABS_MT_POSITION_Y = 0x36,
00533 L4RE_ABS_MT_TOOL_TYPE = 0x37,
00534 L4RE_ABS_MT_BLOB_ID = 0x38,
00535 L4RE_ABS_MT_TRACKING_ID = 0x39,
00536 L4RE_ABS_MT_PRESSURE = 0x3a,
00537 L4RE_ABS_MT_DISTANCE = 0x3b,
00538
00539 L4RE_ABS_MAX = 0x3f,
00540 };
00541
00542 enum L4Re_events_msc
00543 {
00544 L4RE_MSC_SERIAL = 0x00,
00545 L4RE_MSC_PULSELED = 0x01,
00546 L4RE_MSC_GESTURE = 0x02,
00547 L4RE_MSC_RAW = 0x03,
00548 L4RE_MSC_SCAN = 0x04,
00549 L4RE_MSC_MAX = 0x07,
00550 };
00551
00552 enum L4Re_events_properties
00553 {
00554 L4RE_EVENT_PROP_POINTER = 0x00,
00555 L4RE_EVENT_PROP_DIRECT = 0x01,
00556 L4RE_EVENT_PROP_BUTTONPAD = 0x02,
00557 L4RE_EVENT_PROP_SEMI_MT = 0x03,
00558 //L4RE_EVENT_PROP_MAX = 0x1f
00559 };

```

## 17.340 l4/re/impl/dataspace\_impl.h File Reference

Dataspace client stub implementation.

```

#include <l4/re/dataspace>
#include <l4/sys/cxx/ipc_client>
#include <l4/sys/cxx/consts>

```



```

00016 L4_RPC_DEF(L4Re::Dataspace::clear);
00017 L4_RPC_DEF(L4Re::Dataspace::allocate);
00018 L4_RPC_DEF(L4Re::Dataspace::copy_in);
00019 L4_RPC_DEF(L4Re::Dataspace::info);
00020 L4_RPC_DEF(L4Re::Dataspace::map_info);
00021
00022 namespace L4Re {
00023
00024
00025 long
00026 Dataspace::__map(Dataspace::Offset offset, unsigned char *size,
00027 Dataspace::Flags flags,
00028 Dataspace::Map_addr local_addr,
00029 L4::Cap<L4::Task> dst) const noexcept
00030 {
00031 Map_addr spot = local_addr & ~(~0ULL « L4_umword_t(*size));
00032 Map_addr base = local_addr & (~0ULL « L4_umword_t(*size));
00033 L4::Ipc::Rcv_fpage r = L4::Ipc::Rcv_fpage::mem(base, *size, 0, dst);
00034
00035 L4::Ipc::Snd_fpage fp;
00036 long err = map_t::call(c(), offset, spot, flags, r, fp, L4_utcb());
00037 if (L4_UNLIKELY(err < 0))
00038 return err;
00039
00040 *size = fp.rcv_order();
00041 return err;
00042 }
00043
00044 long
00045 Dataspace::map_region(Dataspace::Offset offset, Dataspace::Flags flags,
00046 Dataspace::Map_addr min_addr,
00047 Dataspace::Map_addr max_addr,
00048 L4::Cap<L4::Task> dst) const noexcept
00049 {
00050 min_addr = L4::trunc_page(min_addr);
00051 max_addr = L4::round_page(max_addr);
00052 unsigned char order = L4_LOG2_PAGESIZE;
00053
00054 long err = 0;
00055
00056 while (min_addr < max_addr)
00057 {
00058 unsigned char order_mapped;
00059 order_mapped = order
00060 = L4::max_order(order, min_addr, min_addr, max_addr, min_addr);
00061
00062 err = __map(offset, &order_mapped, flags, min_addr, dst);
00063 if (L4_UNLIKELY(err < 0))
00064 return err;
00065
00066 if (order > order_mapped)
00067 order = order_mapped;
00068
00069 min_addr += Map_addr(1) « order;
00070 offset += Map_addr(1) « order;
00071
00072 if (min_addr >= max_addr)
00073 return 0;
00074
00075 while (min_addr != L4::trunc_order(min_addr, order)
00076 || max_addr < L4::round_order(min_addr + 1, order))
00077 --order;
00078 }
00079
00080 return 0;
00081 }
00082
00083
00084 long
00085 Dataspace::map(Dataspace::Offset offset, Dataspace::Flags flags,
00086 Dataspace::Map_addr local_addr,
00087 Dataspace::Map_addr min_addr,
00088 Dataspace::Map_addr max_addr,
00089 L4::Cap<L4::Task> dst) const noexcept
00090 {
00091 min_addr = L4::trunc_page(min_addr);
00092 max_addr = L4::round_page(max_addr);
00093 local_addr = L4::trunc_page(local_addr);
00094 unsigned char order
00095 = L4::max_order(L4_LOG2_PAGESIZE, local_addr, min_addr, max_addr, local_addr);
00096
00097 return __map(offset, &order, flags, local_addr, dst);
00098 }
00099
00100 Dataspace::Size
00101 Dataspace::size() const noexcept
00102 {

```



```

00103 Stats stats = Stats();
00104 int err = info(&stats);
00105 if (err < 0)
00106 return 0;
00107 return stats.size;
00108 }
00109
00110 Dataspace::Flags
00111 Dataspace::flags() const noexcept
00112 {
00113 Stats stats = Stats();
00114 int err = info(&stats);
00115 if (err < 0)
00116 return Flags(0);
00117 return stats.flags;
00118 }
00119
00120 };

```

## 17.342 l4/re/impl/mem\_alloc\_impl.h File Reference

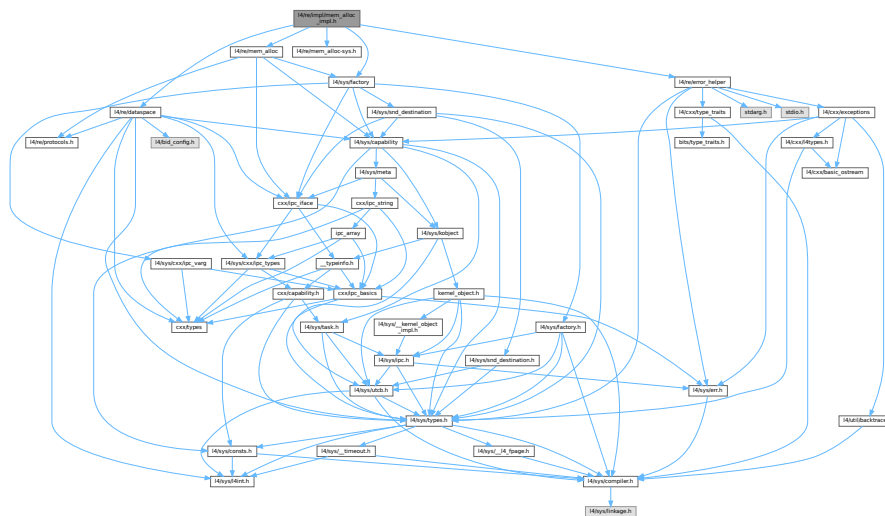
Memory allocator client stub implementation.

```

#include <l4/re/mem_alloc>
#include <l4/re/mem_alloc-sys.h>
#include <l4/re/dataspace>
#include <l4/re/error_helper>
#include <l4/sys/factory>

```

Include dependency graph for mem\_alloc\_impl.h:



### Namespaces

- namespace [L4Re](#)  
*L4Re C++ Interfaces.*

### 17.342.1 Detailed Description

Memory allocator client stub implementation.

Definition in file [mem\\_alloc\\_impl.h](#).

## 17.343 mem\_alloc\_impl.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #include <l4/re/mem_alloc>
00013 #include <l4/re/mem_alloc-sys.h>
00014 #include <l4/re/dataspace>
00015 #include <l4/re/error_helper>
00016
00017 #include <l4/sys/factory>
00018
00019
00020 namespace L4Re
00021 {
00022
00023 long
00024 Mem_alloc::alloc(long size,
00025 L4::Cap<Dataspace> mem, unsigned long flags,
00026 unsigned long align, l4_addr_t paddr) const noexcept
00027 {
00028 L4::Cap<L4::Factory> f(cap());
00029 auto call = f->create(mem, L4Re::Dataspace::Protocol);
00030 call « l4_mword_t(size)
00031 « l4_umword_t(flags)
00032 « l4_umword_t(align);
00033 if (flags & Fixed_paddr)
00034 call « l4_umword_t(paddr);
00035 return l4_error(call);
00036 }
00037
00038 };

```

## 17.344 l4/re/impl/namespace\_impl.h File Reference

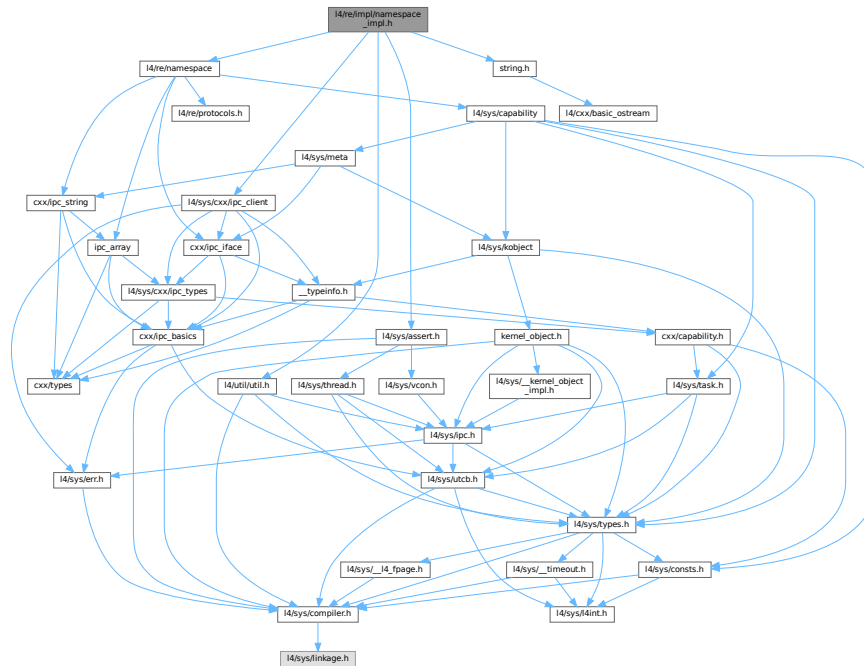
Namespace client stub implementation.

```

#include <l4/re/namespace>
#include <l4/util/util.h>
#include <l4/sys/cxx/ipc_client>
#include <l4/sys/assert.h>
#include <string.h>

```

Include dependency graph for namespace\_impl.h:



## Namespaces

- namespace [L4Re](#)  
*L4Re C++ Interfaces.*

### 17.344.1 Detailed Description

Namespace client stub implementation.

Definition in file [namespace\\_impl.h](#).

## 17.345 namespace\_impl.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #include <l4re/namespace>
00013
00014 #include <l4/util/util.h>
00015 #include <l4/sys/cxx/pc_client>
00016 #include <l4/sys/assert.h>
00017
00018 #include <string.h>
00019
00020 L4_RPC_DEF(L4Re::Namespace::query);

```

```

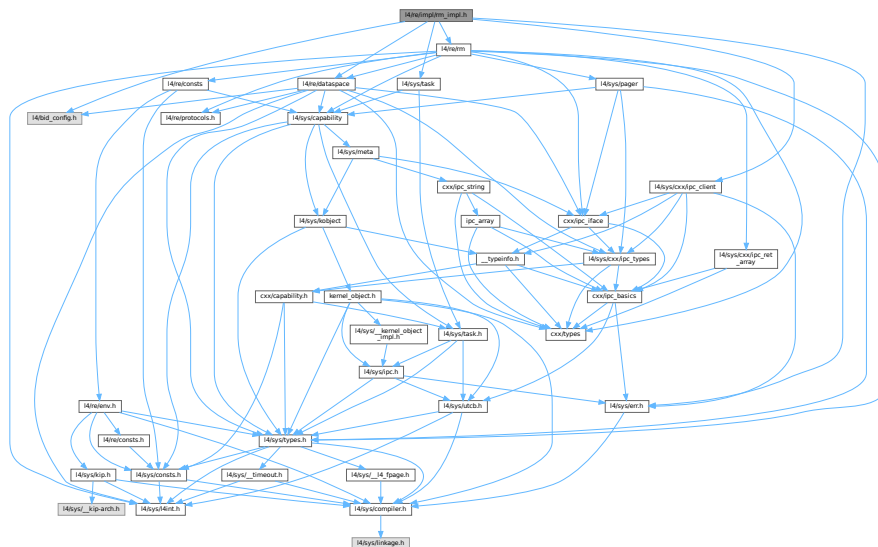
00021 L4_RPC_DEF(L4Re::Namespace::register_obj);
00022 L4_RPC_DEF(L4Re::Namespace::unlink);
00023
00024 namespace L4Re {
00025
00026 long
00027 Namespace::_query(char const *name, unsigned len,
00028 L4::Cap<void> const &target,
00029 l4_umword_t *local_id, bool iterate) const noexcept
00030 {
00031 l4_assert(target.is_valid());
00032
00033 L4::Cap<Namespace> ns = c();
00034 L4::Ipc::Array<char const, unsigned long> _name(len, name);
00035
00036 while (_name.length > 0)
00037 {
00038 L4::Ipc::Snd_fpage cap;
00039 L4::Opcode dummy;
00040 int err = query_t::call(ns, _name,
00041 L4::Ipc::Small_buf(target.cap(),
00042 local_id
00043 ? L4_RCV_ITEM_LOCAL_ID
00044 : 0),
00045 cap, dummy, _name);
00046 if (err < 0)
00047 return err;
00048
00049 bool const partly = err & Partly_resolved;
00050 if (cap.id_received())
00051 {
00052 *local_id = cap.data();
00053 return _name.length;
00054 }
00055
00056 if (partly && iterate)
00057 ns = L4::cap_cast<Namespace>(target);
00058 else
00059 return err;
00060 }
00061
00062 return _name.length;
00063 }
00064
00065 long
00066 Namespace::query(char const *name, unsigned len, L4::Cap<void> const &target,
00067 int timeout, l4_umword_t *local_id, bool iterate) const noexcept
00068 {
00069 if (L4_UNLIKELY(len == 0))
00070 return -L4_EINVAL;
00071
00072 if (L4_UNLIKELY(timeout < 0))
00073 return -L4_EINVAL;
00074
00075 long ret;
00076 long rem = timeout;
00077 long to = 0;
00078
00079 if (rem)
00080 to = 10;
00081
00082 do
00083 {
00084 ret = _query(name, len, target, local_id, iterate);
00085
00086 if (ret >= 0)
00087 return ret;
00088
00089 if (L4_UNLIKELY(ret != -L4_EAGAIN))
00090 return ret;
00091
00092 if (rem == to)
00093 return ret;
00094
00095 l4_sleep(to);
00096
00097 if (rem > 0)
00098 {
00099 rem -= to;
00100 if (rem < 0)
00101 {
00102 to = rem = 0;
00103 continue; // one final try
00104 }
00105 }
00106
00107 if (to < 100)

```

## 17.346 l4/re/impl/rm\_impl.h File Reference

```
#include <l4/bid_config.h>
#include <l4/re/rm>
#include <l4/re/dataspace>
#include <l4/sys/cxx/ipc_client>
#include <l4/sys/task>
#include <l4/sys/err.h>
```

Include dependency graph for rm\_impl.h:



- namespace **L4Re**  
*L4Re C++ Interfaces.*

Definition in file [rm\\_impl.h](#).

## 17.347 rm\_impl.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #include <l4/bid_config.h>
00013 #include <l4/re/rm>
00014 #include <l4/re/dataspace>
00015
00016 #include <l4/sys/cxx/ipc_client>
00017
00018 #include <l4/sys/task>
00019 #include <l4/sys/err.h>
00020
00021 L4_RPC_DEF(L4Re::Rm::reserve_area);
00022 L4_RPC_DEF(L4Re::Rm::free_area);
00023 L4_RPC_DEF(L4Re::Rm::attach);
00024 L4_RPC_DEF(L4Re::Rm::detach);
00025 L4_RPC_DEF(L4Re::Rm::get_regions);
00026 L4_RPC_DEF(L4Re::Rm::get_areas);
00027 L4_RPC_DEF(L4Re::Rm::find);
00028 L4_RPC_DEF(L4Re::Rm::get_info);
00029
00030 namespace L4Re
00031 {
00032
00033 long
00034 Rm::attach(l4_addr_t *start, unsigned long size, Rm::Flags flags,
00035 L4::Ipc::Cap<Dataspace> mem, Rm::Offset offs,
00036 unsigned char align, L4::Cap<L4::Task> const task,
00037 char const *name, Rm::Offset backing_offset) const noexcept
00038 {
00039 if ((flags & F::Rights_mask) == Flags(0)
00040 || (flags & (F::Reserved | F::Kernel)))
00041 mem = L4::Ipc::Cap<L4Re::Dataspace>();
00042
00043 char const n = '\0';
00044 long e = attach_t::call(c(), start, size, flags, mem, offs, align,
00045 mem.cap().cap(), name ? name : &n, backing_offset);
00046 if (e < 0)
00047 return e;
00048
00049 #ifndef CONFIG_MMU
00050 if ((flags & (F::Eager_map | F::No_eager_map)) == F::Eager_map)
00051 #else
00052 if (!(flags & F::No_eager_map) && mem.is_valid())
00053 #endif
00054 e = mem.cap()->map_region(offs, map_flags(flags), *start, *start + size,
00055 task);
00056
00057 return e;
00058 }
00059
00060 int
00061 Rm::detach(l4_addr_t start, unsigned long size, L4::Cap<Dataspace> *mem,
00062 L4::Cap<L4::Task> task, unsigned flags) const noexcept
00063 {
00064 l4_addr_t rstart = 0, rsize = 0;
00065 l4_cap_idx_t mem_cap = L4_INVALID_CAP;
00066 long e = detach_t::call(c(), start, size, flags, rstart, rsize, mem_cap);
00067 if (L4_UNLIKELY(e < 0))
00068 return e;
00069
00070 if (mem)
00071 *mem = L4::Cap<L4Re::Dataspace>(mem_cap);
00072
00073 if (!task.is_valid())
00074 return e;
00075
00076 rsize = l4_round_page(rsize);
00077 unsigned order = L4_LOG2_PAGESIZE;
00078 unsigned long sz = (1UL << order);
00079 for (unsigned long p = rstart; rsize; p += sz, rsize -= sz)
00080 {
00081 while (sz > rsize)
00082 {
00083 --order;
00084 sz >>= 1;
00085 }

```

```

00086
00087 for (;;)
00088 {
00089 unsigned long m = sz << 1;
00090 if (m > rsize)
00091 break;
00092
00093 if (p & (m - 1))
00094 break;
00095
00096 ++order;
00097 sz <= 1;
00098 }
00099
00100 task->unmap(l4_fpage(p, order, L4_FPAGE_RWX),
00101 L4_FP_ALL_SPACES);
00102 }
00103
00104 return e;
00105 }
00106 }

```

## 17.348 inhibitor

```

00001 // vim:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2014 Steffen Liebergeld <steffen.liebergeld@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/sys/capability>
00010 #include <l4/sys/cxx/ipc_iface>
00011 #include <l4/sys/cxx/ipc_string>
00012 #include <l4/re/protocols.h>
00013
00014 namespace L4Re {
00015
00038 class Inhibitor :
00039 public L4::Kobject_t<Inhibitor, L4::Kobject, L4RE_PROTO_INHIBITOR>
00040 {
00041 public:
00042 enum
00043 {
00044 Name_max = 20
00045 };
00046
00057 L4_INLINE_RPC(long, acquire, (l4_umword_t id, L4::Ipc::String<> reason));
00058
00067 L4_INLINE_RPC(long, release, (l4_umword_t id));
00068
00084 long next_lock_info(char *name, unsigned len, l4_mword_t current_id = -1,
00085 l4_utcb_t *utcb = l4_utcb())
00086 {
00087 L4::Ipc::String<char> name_buf(len, name);
00088 long r = next_lock_info_t::call(c(), ¤t_id, name_buf, utcb);
00089 if (r < 0)
00090 return r;
00091
00092 return current_id;
00093 }
00094
00095 L4_INLINE_RPC_NF(long, next_lock_info, (L4::Ipc::In_out<l4_mword_t *> current_id,
00096 L4::Ipc::String<char> &name));
00097
00098 typedef L4::Typeid::Rpc<acquire_t, release_t, next_lock_info_t> Rpcs;
00099 };
00100
00101 }

```

## 17.349 inhibitor-sys.h

```

00001 /*
00002 * (c) 2014 Steffen Liebergeld <steffen.liebergeld@kernkonzept.com>
00003 *
00004 * License: see LICENSE.spdx (in this directory or the directories above)
00005 */

```

```

00006 #pragma once
00007
00008 namespace L4Re {
00009 namespace Inhibitor_ {
00015 enum Opcodes { Acquire, Release, Next_lock_info };
00016 }
00017 }

```

## 17.350 itas

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * Copyright (C) 2025 Kernkonzept GmbH.
00004 * Author(s): Jan Klötzke <jan.kloetzke@kernkonzept.com>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00009 #pragma once
00010
00011 #include <l4/sys/cxx/ipc_iface>
00012 #include <l4/sys/cxx/ipc_types>
00013 #include <l4/sys/cxx/types>
00014 #include <l4/sys/l4int.h>
00015 #include <l4/sys/thread>
00016 #include <l4/sys/types.h>
00017
00018 #include <signal.h>
00019 #include <sys/time.h>
00020
00021 namespace L4Re
00022 {
00023
00030 class L4_EXPORT Itas :
00031 public L4::Kobject_t<Itas, L4::Kobject,
00032 L4RE_PROTO_ITAS,
00033 L4::Type_info::Demand_t<2> >
00034 {
00035 public:
00048 L4_INLINE_RPC(int, register_thread, (L4::Ipc::Cap<L4::Thread> parent,
00049 L4::Ipc::Cap<L4::Thread> thread_cap,
00050 l4_addr_t thread_utcb));
00060 L4_INLINE_RPC(int, unregister_thread, (L4::Ipc::Cap<L4::Thread> thread));
00061
00062 // sigaction.sa_flags is usually `unsigned long`, except for MIPS...
00063 enum : unsigned
00064 {
00065 Ignore_sigaction = ~0U
00066 };
00067
00079 L4_INLINE_RPC(int, sigaction, (int signum,
00080 const struct sigaction *act,
00081 struct sigaction *oldact));
00082
00095 L4_INLINE_RPC(int, sigaltstack, (L4::Ipc::Cap<L4::Thread> thread,
00096 const struct sigaltstack *ss,
00097 struct sigaltstack *oss));
00098
00113 L4_INLINE_RPC(int, sigprocmask, (L4::Ipc::Cap<L4::Thread> thread,
00114 int how, sigset_t const *set,
00115 sigset_t *oldset));
00116
00126 L4_INLINE_RPC(int, sigpending, (L4::Ipc::Cap<L4::Thread> thread,
00127 sigset_t *set));
00128
00138 L4_INLINE_RPC(int, setitimer, (int which,
00139 const struct itimerval *new_value,
00140 struct itimerval *old_value));
00141
00150 L4_INLINE_RPC(int, getitimer, (int which, struct itimerval *curr_value));
00151
00158 L4_INLINE_RPC(int, raise, (L4::Ipc::Cap<L4::Thread> thread, int sig));
00159
00160 typedef L4::Typeid::Rpc<
00161 register_thread_t, unregister_thread_t, sigaction_t, sigaltstack_t,
00162 sigprocmask_t, sigpending_t, setitimer_t, getitimer_t, raise_t
00163 > Rpc<
00164 >;
00165
00166 }

```

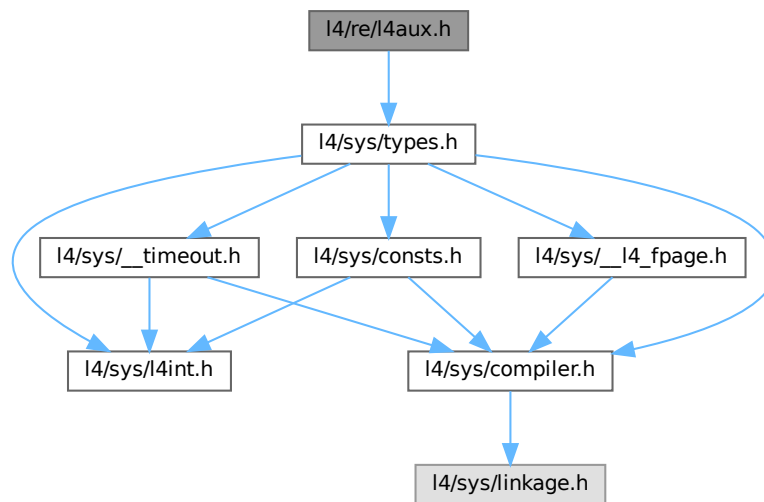


## 17.351 l4/re/l4aux.h File Reference

Auxiliary definitions.

```
#include <l4/sys/types.h>
```

Include dependency graph for l4aux.h:



### Data Structures

- struct `l4re_aux_t`  
*Auxiliary descriptor.*

### Typedefs

- typedef struct `l4re_aux_t` `l4re_aux_t`  
*Auxiliary descriptor.*

### Enumerations

- enum `l4re_aux_ldr_flags_t`  
*Flags for program loading.*

## 17.351.1 Detailed Description

Auxiliary definitions.

Definition in file `l4aux.h`.

## 17.352 l4aux.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00009 * Björn Döbel <doebel@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014
00015 #include <l4/sys/types.h>
00016
00022
00028 enum l4re_aux_ldr_flags_t
00029 {
00030 L4RE_AUX_LDR_FLAG_EAGER_MAP = 0x1,
00031 L4RE_AUX_LDR_FLAG_ALL_SEGS_COW = 0x2,
00032 L4RE_AUX_LDR_FLAG_PINNED_SEGS = 0x4,
00033 };
00034
00040 typedef struct l4re_aux_t
00041 {
00042 char const * binary;
00043 l4_cap_idx_t kip_ds;
00044 l4_umword_t dbg_lvl;
00045 l4_umword_t ldr_flags;
00046 l4_addr_t ldr_base;
00047 } l4re_aux_t;
00048

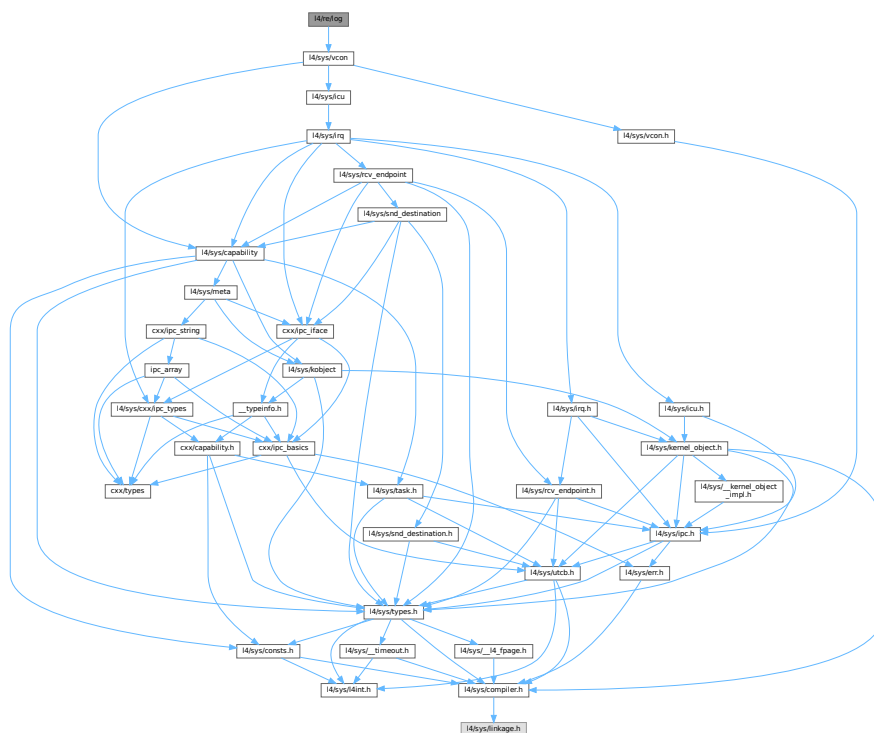
```

### 17.353 I4/re/log File Reference

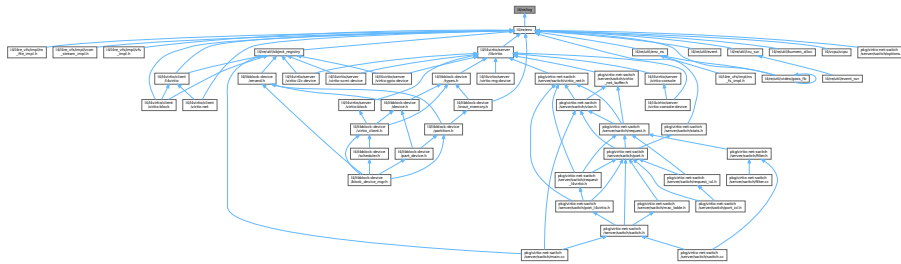
Log interface.

```
#include <linux/sys/vcon>
```

Include dependency graph for log:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class [L4Re::Log](#)  
*Log interface class.*

## Namespaces

- namespace [L4Re](#)  
*L4Re C++ Interfaces.*

## 17.353.1 Detailed Description

Log interface.

Definition in file [log](#).

## 17.354 log

[Go to the documentation of this file.](#)

```
00001 // -*- Mode: C++ -*-
00002 // vim:ft=cpp
00007 /*
00008 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00009 * Alexander Warg <warg@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #pragma once
00015
00016 #include <l4/sys/vcon>
00017
00018 namespace L4Re {
00019
00033 class L4_EXPORT Log : public L4::Kobject_t<Log, L4::Vcon, L4::PROTO_EMPTY>
00034 {
00035 public:
00036
00043 void printn(char const *string, int len) const noexcept;
00044
00050 void print(char const *string) const noexcept;
00051 };
00052 }
```

## 17.355 l4/re/log-sys.h File Reference

Log protocol definition.

### Namespaces

- namespace [L4Re](#)  
*[L4Re](#) C++ Interfaces.*

### Enumerations

- enum [L4Re::Log\\_::Opcodes](#)  
*Logging-service communication-protocol opcodes.*

### 17.355.1 Detailed Description

Log protocol definition.

Definition in file [log-sys.h](#).

## 17.356 log-sys.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00014 namespace L4Re
00015 {
00016 namespace Log_
00017 {
00023 enum Opcodes { Print };
00024 };
00025 };

```

## 17.357 l4/re/mem\_alloc File Reference

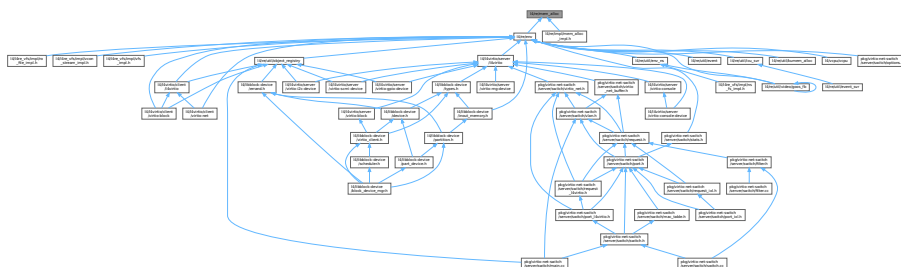
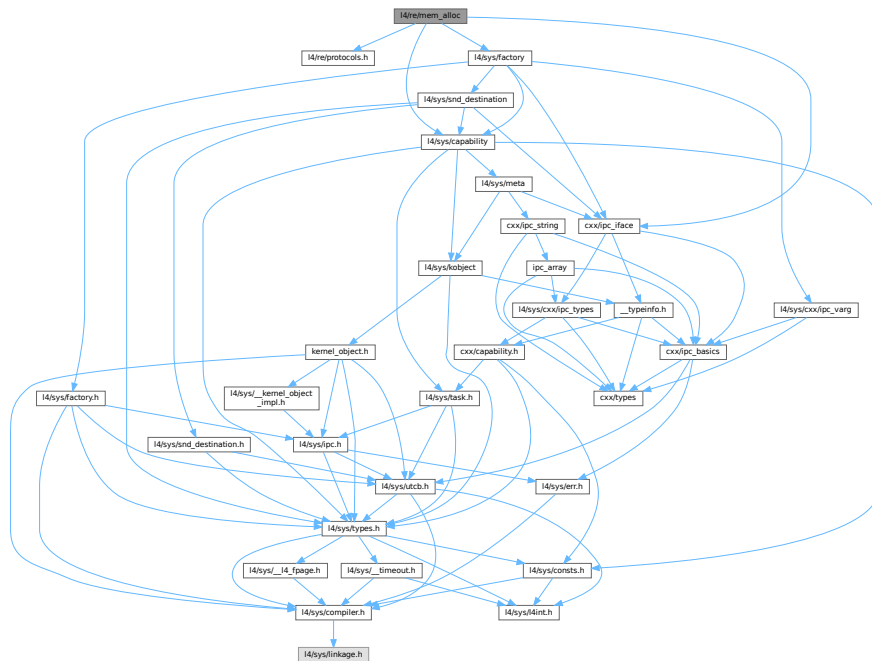
Memory allocator interface.

```

#include <l4/re/protocols.h>
#include <l4/sys/capability>
#include <l4/sys/cxx/ipc_iface>

```

Include dependency graph for mem\_alloc:



1. [Lecture 1: Introduction to the course](#)

## 17.357.1 Detailed Description

Memory allocator interface.

Definition in file [mem\\_alloc](#).

## 17.358 mem\_alloc

[Go to the documentation of this file.](#)

```

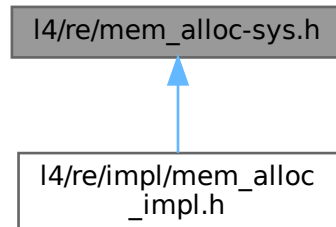
00001 // -*- Mode: C++ -*-
00002 // vim:ft=cpp
00007 /*
00008 * Copyright (C) 2014-2016, 2019, 2021, 2024 Kernkonzept GmbH.
00009 * Author(s): Alexander Warg <alexander.warg@kernkonzept.com>
00010 */
00011 /*
00012 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00013 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00014 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00015 * economic rights: Technische Universität Dresden (Germany)
00016 *
00017 * License: see LICENSE.spdx (in this directory or the directories above)
00018 */
00019 #pragma once
00020
00021 #include <l4/re/protocols.h>
00022 #include <l4/sys/capability>
00023 #include <l4/sys/cxx/ipc_iface>
00024 #include <l4/sys/factory>
00025
00026 namespace L4Re {
00027 class Dataspace;
00028
00029 // MISSING:
00030 // * alignment constraints
00031 // * shall we support superpages in noncont memory?
00032
00052 class L4_EXPORT Mem_alloc :
00053 public L4::Kobject_t<Mem_alloc, L4::Factory, L4RE_PROTO_MEM_ALLOC>
00054 {
00055 public:
00062 enum Mem_alloc_flags
00063 {
00064 Continuous = 0x01,
00065 Pinned = 0x02,
00066 Super_pages = 0x04,
00067 Fixed_paddr = 0x08,
00068 };
00069
00070 struct Stats
00071 {
00083 l4_size_t quota;
00084
00094 l4_size_t quota_used;
00095
00102 l4_size_t mem_limit;
00103
00116 l4_size_t mem_used;
00117
00126 l4_size_t mem_free;
00127 };
00128
00157 long alloc(long size, L4::Cap<Dataspace> mem,
00158 unsigned long flags = 0, unsigned long align = 0,
00159 l4_addr_t paddr = 0) const noexcept;
00160
00169 L4_INLINE_RPC(long, info, (Stats &stats));
00170
00171 typedef L4::Typeid::Rpc<info_t> Rpc;
00172 };
00173
00174 };

```

## 17.359 l4/re/mem\_alloc-sys.h File Reference

Memory allocator protocol definitions.

This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [L4Re](#)  
*[L4Re](#) C++ Interfaces.*

### Enumerations

- enum [L4Re::Mem\\_alloc\\_::Opcodes](#)  
*Memory-allocator communication-protocol opcodes.*

### 17.359.1 Detailed Description

Memory allocator protocol definitions.

Definition in file [mem\\_alloc-sys.h](#).

## 17.360 mem\_alloc-sys.h

[Go to the documentation of this file.](#)

```
00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00014 namespace L4Re
00015 {
00016 namespace Mem_alloc_
00017 {
00023 enum Opcodes { Alloc, Free };
00024 };
00025 };
```





## 17.362 mmio\_space

[Go to the documentation of this file.](#)

```

00001 // -*- Mode: C++ -*-
00002 // vim:ft=cpp
00003 /*
00004 * Copyright (C) 2017-2018, 2022, 2024 Kernkonzept GmbH.
00005 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00013 #pragma once
00014
00015 #include <l4/re/protocols.h>
00016 #include <l4/sys/capability>
00017 #include <l4/sys/cxx/ipc_types>
00018 #include <l4/sys/cxx/ipc_iface>
00019
00020 namespace L4Re
00021 {
00022
00045 struct L4_EXPORT Mmio_space
00046 : public L4::Kobject_t<Mmio_space, L4::Kobject, L4RE_PROTO_MMIO_SPACE>
00047 {
00049 enum Access_width
00050 {
00051 Wd_8bit = 0,
00052 Wd_16bit = 1,
00053 Wd_32bit = 2,
00054 Wd_64bit = 3
00055 };
00056
00058 typedef l4_uint64_t Addr;
00059
00074 L4_INLINE_RPC(long, mmio_read, (Addr addr, char width, l4_uint64_t *value));
00075
00090 L4_INLINE_RPC(long, mmio_write, (Addr addr, char width, l4_uint64_t value));
00091
00092 typedef L4::Typeid::Rpc<mmio_read_t, mmio_write_t> Rpc;
00093 };
00094
00095 }
```

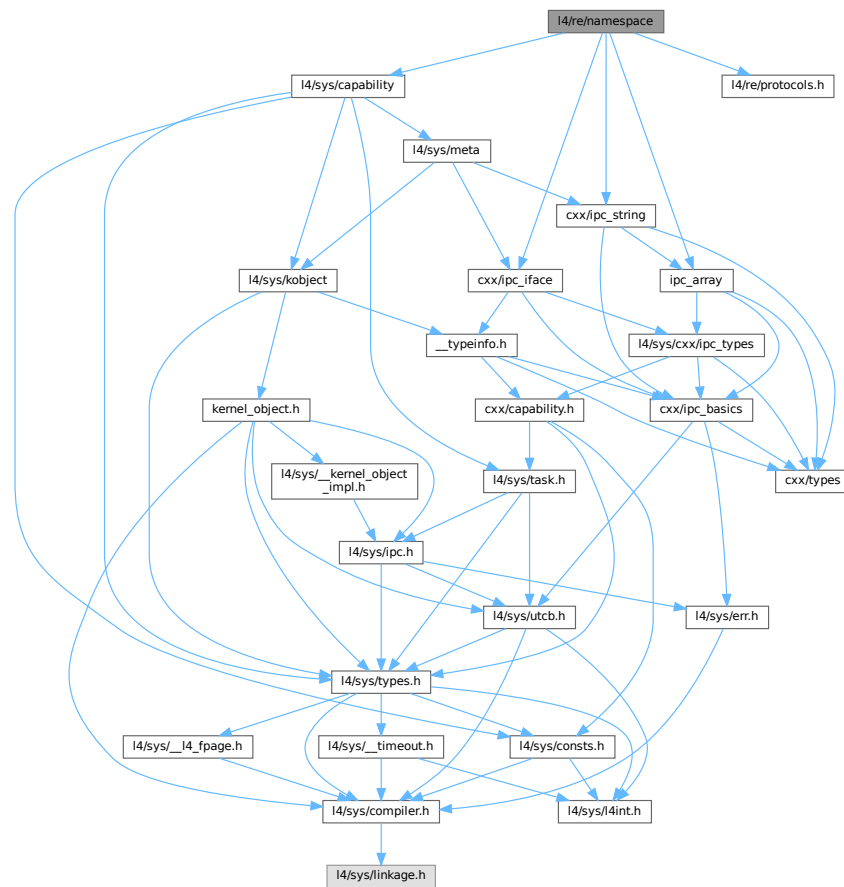
## 17.363 l4/re/namespace File Reference

Namespace interface.

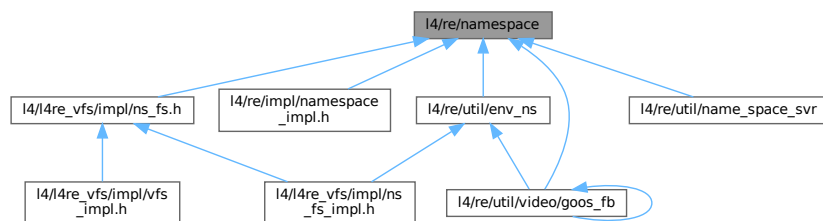
```

#include <l4/sys/capability>
#include <l4/re/protocols.h>
#include <l4/sys/cxx/ipc_iface>
#include <l4/sys/cxx/ipc_array>
#include <l4/sys/cxx/ipc_string>
```

Include dependency graph for namespace:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class [L4Re::Namespace](#)  
*Name-space interface.*

## Namespaces

- namespace [L4Re](#)  
*L4Re C++ Interfaces.*

### 17.363.1 Detailed Description

Namespace interface.

Definition in file [namespace](#).

## 17.364 namespace

[Go to the documentation of this file.](#)

```

00001 // -*- Mode: C++ -*-
00002 // vim:ft=cpp
00003 /*
00004 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00005 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00006 * Björn Döbel <doebel@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #pragma once
00012
00013 #include <l4/sys/capability>
00014 #include <l4/re/protocols.h>
00015 #include <l4/sys/cxx/ipc_iface>
00016 #include <l4/sys/cxx/ipc_array>
00017 #include <l4/sys/cxx/ipc_string>
00018
00019 namespace L4Re {
00020
00021 class L4_EXPORT Namespace :
00022 public L4::Kobject_t<Namespace, L4::Kobject, L4RE_PROTO_NAMESPACE,
00023 L4::Type_info::Demand_t<1> >
00024 {
00025 public:
00026 enum Register_flags
00027 {
00028 Ro = L4_CAP_FPAGE_RO,
00029 Rw = L4_CAP_FPAGE_RW,
00030 Rs = L4_CAP_FPAGE_RS,
00031 Rws = L4_CAP_FPAGE_RWS,
00032 Strong = L4_CAP_FPAGE_S,
00033 Trusted = 0x008,
00034
00035 Cap_flags = Ro | Rw | Strong | Trusted,
00036
00037 Link = 0x100,
00038 Overwrite = 0x200,
00039 };
00040
00041 enum Query_result_flags
00042 {
00043 Partly_resolved = 0x020,
00044 };
00045
00046 enum Query_timeout
00047 {
00048 To_default = 3600000,
00049 To_non_blocking = 0,
00050 };
00051
00052 L4_RPC_NF(
00053 long, query, (L4::Ipc::Array_ref<char const, unsigned long> name,
00054 L4::Ipc::Small_buf cap,
00055 L4::Ipc::Snd_fpage &snd_cap, L4::Ipc::Opt<L4::Opcode &> dummy,
00056 L4::Ipc::Opt<L4::Ipc::Array_ref<char const, unsigned long> &> out_name));
00057
00058 long query(char const *name, L4::Cap<void> const &cap,
00059 int timeout = To_default,
00060 l4_umword_t *local_id = 0, bool iterate = true) const noexcept;
00061
00062 long query(char const *name, unsigned len, L4::Cap<void> const &cap,
00063 int timeout = To_default,
00064 l4_umword_t *local_id = 0, bool iterate = true) const noexcept;
00065
00066 L4_RPC_NF(long, register_obj, (unsigned flags,
00067 L4::Ipc::Array<char const, unsigned long> name,
00068 L4::Ipc::Opt< L4::Ipc::Cap<void> > obj),
00069 L4::Ipc::Call_t<L4_CAP_FPAGE_W>);

```

```

00141
00165 long register_obj(char const *name, L4::Ipc::Cap<void> obj,
00166 unsigned flags = Rw) const noexcept
00167 {
00168 return register_obj_t::call(c(), flags,
00169 L4::Ipc::Array<char const, unsigned long>(
00170 __builtin_strlen(name), name),
00171 obj);
00172 }
00173
00174 L4_RPC_NF_OP(3, // backward compatibility opcode
00175 long, unlink, (L4::Ipc::Array<char const, unsigned long> name),
00176 L4::Ipc::Call_t<L4_CAP_FPAGE_W>);
00177
00192 long unlink(char const* name)
00193 {
00194 return unlink_t::call(c(), L4::Ipc::Array<char const, unsigned long>(
00195 __builtin_strlen(name), name));
00196 }
00197
00198 typedef L4::Typeid::Rpc<query_t, register_obj_t, unlink_t> Rpc<
00199
00200 private:
00201 long _query(char const *name, unsigned len,
00202 L4::Cap<void> const &target, l4_umword_t *local_id,
00203 bool iterate) const noexcept;
00204
00205 };
00206
00207 };

```

## 17.365 l4/re/namespace-sys.h File Reference

Namespace protocol definitions.

### Namespaces

- namespace [L4Re](#)  
*L4Re C++ Interfaces.*

### Enumerations

- enum [L4Re::Namespace\\_::Opcodes](#)  
*Name-space communication-protocol opcodes.*

### 17.365.1 Detailed Description

Namespace protocol definitions.

Definition in file [namespace-sys.h](#).

## 17.366 namespace-sys.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00014 namespace L4Re {
00015 namespace Namespace_
00016 {
00022 enum Opcodes { Query, Register, Link, Unlink };
00023 };
00024 };

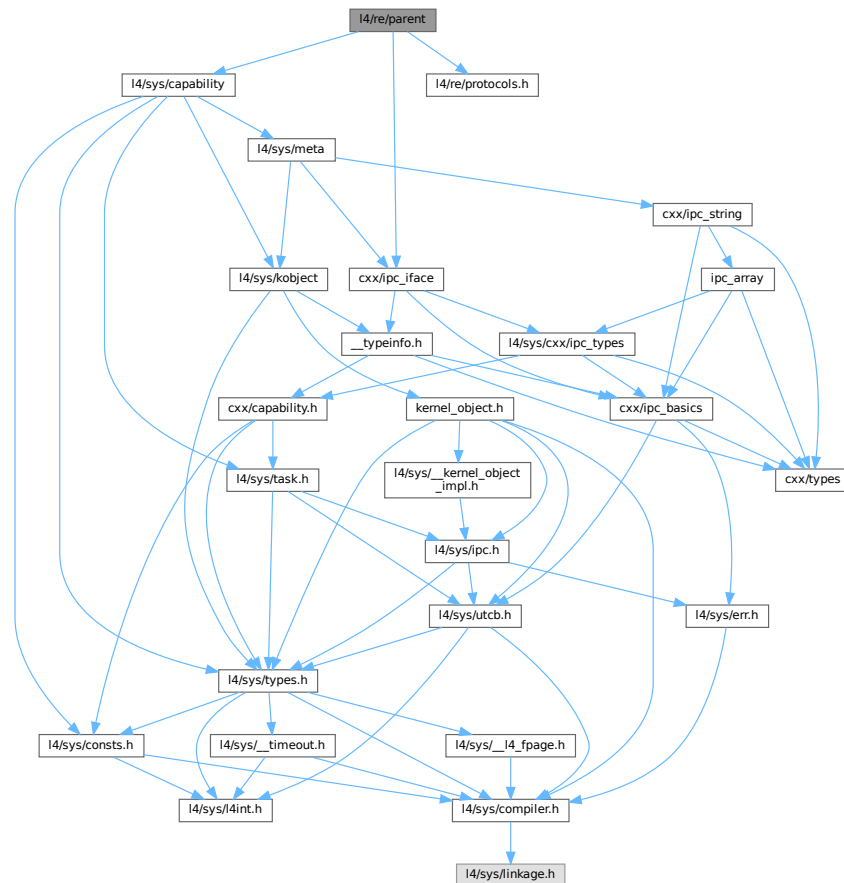
```

## 17.367 I4/re/parent File Reference

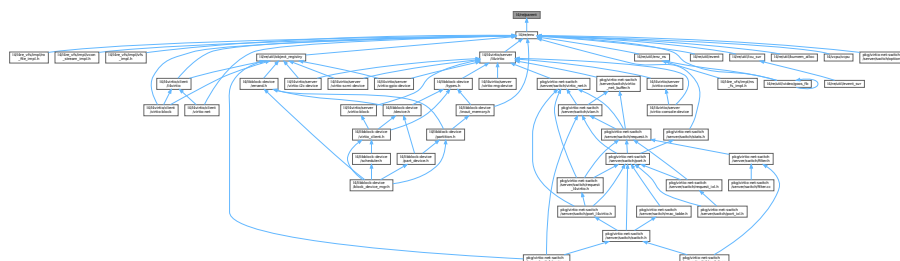
Parent interface.

```
#include <l4/sys/capability>
#include <l4/re/protocols.h>
#include <l4/sys/cxx/ipc_iface>
```

Include dependency graph for parent:



This graph shows which files directly or indirectly include this file:



### Data Structures

- class [L4Re::Parent](#)  
*Parent interface.*

## Namespaces

- namespace [L4Re](#)  
*L4Re C++ Interfaces.*

### 17.367.1 Detailed Description

Parent interface.

Definition in file [parent](#).

## 17.368 parent

[Go to the documentation of this file.](#)

```

00001 // -*- Mode: C++ -*-
00002 // vim:ft=cpp
00007 /*
00008 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00009 * Alexander Warg <warg@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #pragma once
00015
00016 #include <l4/sys/capability>
00017 #include <l4/re/protocols.h>
00018 #include <l4/sys/cxx/ipc_iface>
00019
00020 namespace L4Re {
00021
00034
00042 class L4_EXPORT Parent :
00043 public L4::Kobject_t<Parent, L4::Kobject, L4RE_PROTO_PARENT>
00044 {
00045 public:
00061 L4_INLINE_RPC(long, signal, (unsigned long sig, unsigned long val));
00062 typedef L4::Typeid::Rpc<signal_t> Rpc;
00063 };
00064 };
00065

```

## 17.369 l4/re/parent-sys.h File Reference

Parent protocol definition.

## Namespaces

- namespace [L4Re](#)  
*L4Re C++ Interfaces.*

## Enumerations

- enum [L4Re::Parent\\_::Opcodes](#)  
*Parent communication-protocol opcodes.*

## 17.369.1 Detailed Description

Parent protocol definition.

Definition in file [parent-sys.h](#).

## 17.370 parent-sys.h

[Go to the documentation of this file.](#)

```

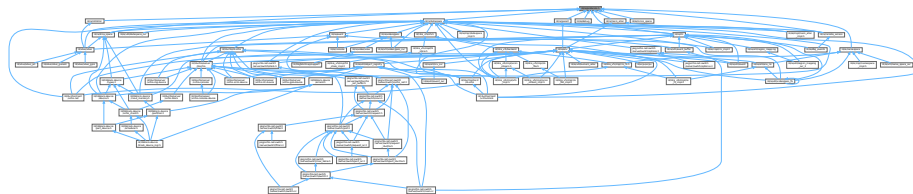
00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00014 namespace L4Re
00015 {
00016 namespace Parent_
00017 {
00023 enum Opcodes { Signal };
00024 };
00025 };

```

## 17.371 I4/re/protocols.h File Reference

[L4Re](#) Protocol Constants (C version).

This graph shows which files directly or indirectly include this file:



### Enumerations

- enum [L4re\\_protocols](#) {  
[L4RE\\_PROTO\\_DATASPACE](#) = 0x4000 , [L4RE\\_PROTO\\_NAMESPACE](#) , [L4RE\\_PROTO\\_PARENT](#) ,  
[L4RE\\_PROTO\\_GOOS](#) ,  
[L4RE\\_PROTO\\_RSVD\\_1](#) , [L4RE\\_PROTO\\_RM](#) , [L4RE\\_PROTO\\_EVENT](#) , [L4RE\\_PROTO\\_INHIBITOR](#) ,  
[L4RE\\_PROTO\\_DMA\\_SPACE](#) , [L4RE\\_PROTO\\_MMIO\\_SPACE](#) , [L4RE\\_PROTO\\_ITAS](#) , [L4RE\\_PROTO\\_MEM\\_ALLOC](#)  
, [L4RE\\_PROTO\\_REMOTE\\_ACCESS](#) , [L4RE\\_PROTO\\_DEBUG](#) = ~0x7fffL }

Common [L4Re](#) Protocol Constants.

## 17.371.1 Detailed Description

[L4Re](#) Protocol Constants (C version).

Definition in file [protocols.h](#).

## 17.372 protocols.h

[Go to the documentation of this file.](#)

```
00001
00005
00006 /*
00007 * (c) 2015 Alexander Warg <alexander.warg@kernkonzept.com>
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011
00012 #pragma once
00013
00019
00024 enum L4re_protocols
00025 {
00026 L4RE_PROTO_DATASPACE = 0x4000,
00027 L4RE_PROTO_NAMESPACE,
00028 L4RE_PROTO_PARENT,
00029 L4RE_PROTO_GOOS,
00030 L4RE_PROTO_RSVD_1,
00031 L4RE_PROTO_RM,
00032 L4RE_PROTO_EVENT,
00033 L4RE_PROTO_INHIBITOR,
00034 L4RE_PROTO_DMA_SPACE,
00035 L4RE_PROTO_MMIO_SPACE,
00036 L4RE_PROTO_ITAS,
00037 L4RE_PROTO_MEM_ALLOC,
00038 L4RE_PROTO_REMOTE_ACCESS,
00039
00040 L4RE_PROTO_DEBUG = ~0x7fffL
00041 };
00042
```

## 17.373 l4/re/random File Reference

Random number generator interface definition.

```
#include <l4/sys/capability>
#include <l4/sys/cxx/ipc_types>
#include <l4/sys/cxx/ipc_iface>
#include <l4/sys/icu>
```



[illegible]

- struct `L4Re::Random`  
*Low-bandwidth interface for random number generators.*

- namespace **L4Re**  
*L4Re C++ Interfaces.*

Definition in file [random](#).

## 17.374 random

[Go to the documentation of this file.](#)

```

00001 // -*- Mode: C++ -*-
00002 /*
00003 * Copyright (C) 2019-2020, 2022, 2024 Kernkonzept GmbH.
00004 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00012 #pragma once
00013
00014 #include <l4/sys/capability>
00015 #include <l4/sys/cxx/ipc_types>
00016 #include <l4/sys/cxx/ipc_iface>
00017 #include <l4/sys/icu>
00018
00019 namespace L4Re
00020 {
00021
00033 struct L4_EXPORT Random
00034 : public L4::Kobject_t<Random, L4::Icu>
00035 {
00060 L4_INLINE_RPC(long, get_random, (l4_size_t size,
00061 L4::Ipc::Array<char, unsigned long> *buffer));
00062
00063 typedef L4::Typeid::Rpcs<get_random_t> Rpcs;
00064 };
00065
00066 } // namespace

```

## 17.375 remote\_access

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * Copyright (C) 2025 Adam Lackorzynski <adam@l4re.org>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #pragma once
00009
00010 #include <l4/re/protocols.h>
00011 #include <l4/re/dataspace>
00012 #include <l4/sys/cxx/ipc_iface>
00013 #include <l4/sys/cxx/ipc_types>
00014 #include <l4/sys/cxx/types>
00015 #include <l4/sys/l4int.h>
00016
00017 namespace L4Re
00018 {
00019
00020 struct L4_EXPORT Remote_access
00021 : public L4::Kobject_t<Remote_access, Dataspace, L4RE_PROTO_REMOTE_ACCESS>
00022 {
00023 enum Access_width
00024 {
00025 Wd_8bit = 0,
00026 Wd_16bit = 1,
00027 Wd_32bit = 2,
00028 Wd_64bit = 3
00029 };
00030
00031 L4_INLINE_RPC(long, read_mem, (l4_addr_t addr, char width, l4_uint64_t *value));
00032 L4_INLINE_RPC(long, write_mem, (l4_addr_t addr, char width, l4_uint64_t value));
00033
00034 //L4_INLINE_RPC(long, get_regs, (unsigned TBD_THREAD_ID, l4_exc_regs_t *regs));
00035 //L4_INLINE_RPC(long, set_regs, (unsigned TBD_THREAD_ID, l4_exc_regs_t regs));
00036
00037 // !!! sizeof(l4_fpu_regs_t) > sizeof(utcb)
00038 //L4_INLINE_RPC(long, get_fpu_regs, (l4_fpu_regs_t *regs));
00039 //L4_INLINE_RPC(long, set_fpu_regs, (l4_fpu_regs_t regs));
00040
00041 L4_INLINE_RPC(long, terminate, (int exit_code), L4::Ipc::Send_only);
00042
00043 typedef L4::Typeid::Rpcs<read_mem_t, write_mem_t, terminate_t> Rpcs;
00044 };
00045
00046 }

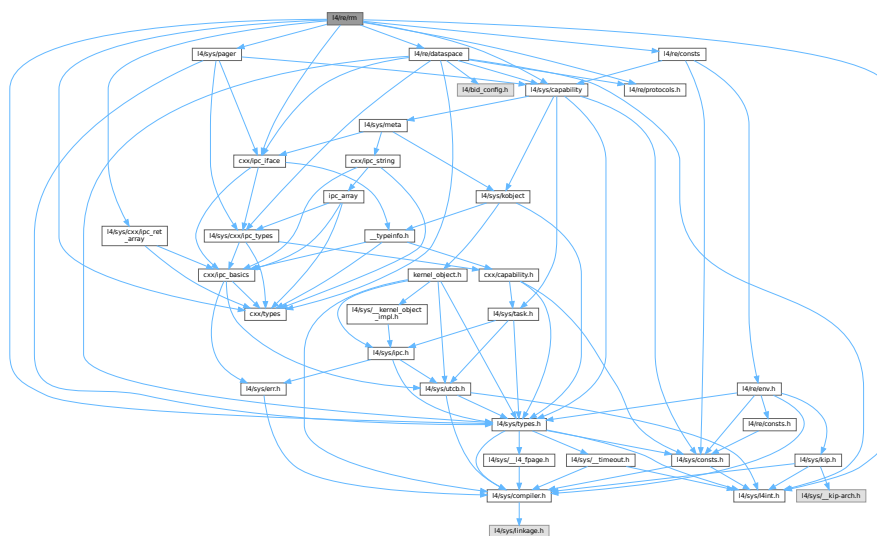
```

### 17.376 I4/re/rm File Reference

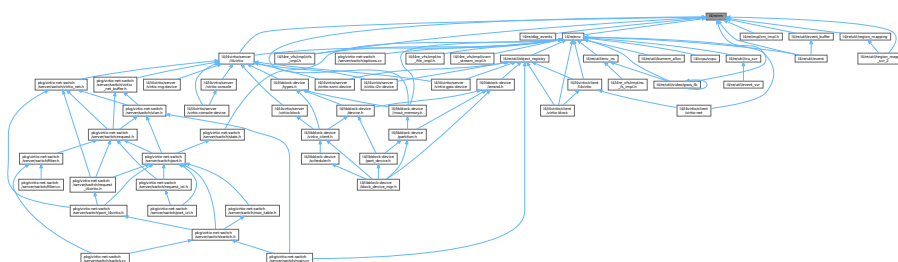
### Region mapper interface.

```
#include <l4/sys/types.h>
#include <l4/sys/l4int.h>
#include <l4/sys/capability>
#include <l4/re/protocols.h>
#include <l4/sys/pager>
#include <l4/sys/cxx/ipc_iface>
#include <l4/sys/cxx/ipc_ret_array>
#include <l4/sys/cxx/types>
#include <l4/re/consts>
#include <l4/re/dataspace>
```

Include dependency graph for rm:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class `L4Re::Rm`  
*Region* map.
- struct `L4Re::Rm::F`  
*Rm* flags definitions.

- class [L4Re::Rm::Unique\\_region< T >](#)  
*Unique region.*
- struct [L4Re::Rm::Region](#)  
*A region is a range of virtual addresses which is backed by content.*
- struct [L4Re::Rm::Area](#)  
*An area is a range of virtual addresses which is reserved, see [L4Re::Rm::reserve\\_area\(\)](#).*

## Namespaces

- namespace [L4Re](#)  
*[L4Re](#) C++ Interfaces.*

## 17.376.1 Detailed Description

Region mapper interface.

Definition in file [rm](#).

## 17.377 rm

[Go to the documentation of this file.](#)

```
00001 // -*- Mode: C++ -*-
00002 // vim:ft=cpp
00007 /*
00008 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00009 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00010 * Björn Döbel <doebel@os.inf.tu-dresden.de>,
00011 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00012 * economic rights: Technische Universität Dresden (Germany)
00013 *
00014 * License: see LICENSE.spdx (in this directory or the directories above)
00015 */
00016 #pragma once
00017
00018 #include <l4/sys/types.h>
00019 #include <l4/sys/l4int.h>
00020 #include <l4/sys/capability>
00021 #include <l4/re/protocols.h>
00022 #include <l4/sys/pager>
00023 #include <l4/sys/cxx/ipc_iface>
00024 #include <l4/sys/cxx/ipc_ret_array>
00025 #include <l4/sys/cxx/types>
00026 #include <l4/re/consts>
00027 #include <l4/re/dataspace>
00028
00029 namespace L4Re {
00030
00073
00081 class L4_EXPORT Rm :
00082 public L4::Kobject_t<Rm, L4::Pager, L4RE_PROTO_RM,
00083 L4::Type_info::Demand_t<1> >
00084 {
00085 public:
00086 typedef L4Re::Dataspace::Offset Offset;
00087
00089 enum Detach_result
00090 {
00091 Detached_ds = 0,
00092 Kept_ds = 1,
00093 Split_ds = 2,
00094 Detach_result_mask = 3,
00095
00096 Detach_again = 4,
00097 };
00098
00099
```

```

00101 enum Region_flag_shifts
00102 {
00104 Caching_shift = Dataspace::F::Caching_shift,
00105 };
00106
00108 struct F
00109 {
00111 enum Attach_flags : l4_uint32_t
00112 {
00114 Search_addr = 0x20000,
00116 In_area = 0x40000,
00118 Eager_map = 0x80000,
00120 No_eager_map = 0x100000,
00122 Attach_mask = 0x1f0000,
00123 };
00124
00125 L4_TYPES_FLAGS_OPS_DEF(Attach_flags);
00126
00128 enum Region_flags : l4_uint16_t
00129 {
00131 Rights_mask = 0x0f,
00133 R = Dataspace::F::R,
00135 W = Dataspace::F::W,
00137 X = Dataspace::F::X,
00139 RW = Dataspace::F::RW,
00141 RX = Dataspace::F::RX,
00143 RWX = Dataspace::F::RWX,
00144
00146 Kernel = 0x100,
00148 Detach_free = 0x200,
00150 Pager = 0x400,
00152 Reserved = 0x800,
00153
00154
00156 Caching_mask = Dataspace::F::Caching_mask,
00159 Cache_normal = Dataspace::F::Normal,
00161 Cache_buffered = Dataspace::F::Bufferable,
00163 Cache_uncached = Dataspace::F::Uncacheable,
00164
00166 Ds_map_mask = 0xff,
00167
00169 Region_flags_mask = 0xffff,
00170 };
00171
00172 L4_TYPES_FLAGS_OPS_DEF(Region_flags);
00173
00174 friend constexpr Dataspace::Flags map_flags(Region_flags rf)
00175 {
00176 return Dataspace::Flags(static_cast<l4_uint16_t>(rf) & Ds_map_mask);
00177 }
00178
00179 struct Flags : L4::Types::Flags_ops_t<Flags>
00180 {
00181 l4_uint32_t raw;
00182 Flags() = default;
00183 explicit constexpr Flags(l4_uint32_t f) : raw(f) {}
00184 constexpr Flags(Attach_flags rf) : raw(static_cast<l4_uint32_t>(rf)) {}
00185 constexpr Flags(Region_flags rf) : raw(static_cast<l4_uint32_t>(rf)) {}
00186
00187 friend constexpr Dataspace::Flags map_flags(Flags f)
00188 {
00189 return Dataspace::Flags(f.raw & Ds_map_mask);
00190 }
00191
00192 constexpr Region_flags region_flags() const
00193 {
00194 return Region_flags(raw & Region_flags_mask);
00195 }
00196
00197 constexpr Attach_flags attach_flags() const
00198 {
00199 return Attach_flags(raw & Attach_mask);
00200 }
00201
00202 constexpr bool r() const { return raw & L4_FPAGE_RO; }
00203 constexpr bool w() const { return raw & L4_FPAGE_W; }
00204 constexpr bool x() const { return raw & L4_FPAGE_X; }
00205 constexpr unsigned cap_rights() const
00206 { return w() ? L4_CAP_FPAGE_RW : L4_CAP_FPAGE_RO; }
00207 };
00208
00209 friend constexpr Flags operator | (Region_flags l, Attach_flags r)
00210 { return Flags(l) | Flags(r); }
00211
00212 friend constexpr Flags operator | (Attach_flags l, Region_flags r)
00213 { return Flags(l) | Flags(r); }
00214 };

```

```

00215
00216 using Attach_flags = F::Attach_flags;
00217 using Region_flags = F::Region_flags;
00218 using Flags = F::Flags;
00219
00221 enum Detach_flags
00222 {
00232 Detach_exact = 1,
00242 Detach_overlap = 2,
00243
00251 Detach_keep = 4,
00252 };
00253
00280 long reserve_area(l4_addr_t *start, unsigned long size,
00281 Flags flags = Flags(0),
00282 unsigned char align = L4_PAGESHIFT) const noexcept
00283 { return reserve_area_t::call(c(), start, size, flags, align); }
00284
00285 L4_RPC_NF(long, reserve_area, (L4::Ipc::In_out<l4_addr_t *> start,
00286 unsigned long size,
00287 Flags flags,
00288 unsigned char align));
00289
00305 template< typename T >
00306 long reserve_area(T **start, unsigned long size,
00307 Flags flags = Flags(0),
00308 unsigned char align = L4_PAGESHIFT) const noexcept
00309 {
00310 return reserve_area_t::call(c(), reinterpret_cast<l4_addr_t*>(start), size,
00311 flags, align);
00312 }
00313
00326 L4_RPC(long, free_area, (l4_addr_t addr));
00327
00328 L4_RPC_NF(long, attach, (L4::Ipc::In_out<l4_addr_t *> start,
00329 unsigned long size, Flags flags,
00330 L4::Ipc::Opt<L4::Ipc::Cap<Dataspace> > mem,
00331 Offset offs, unsigned char align,
00332 L4::Ipc::Opt<l4_cap_idx_t> client_cap,
00333 L4::Ipc::String<> name, Offset backing_offset));
00334
00335 L4_RPC_NF(long, detach, (l4_addr_t addr, unsigned long size, unsigned flags,
00336 l4_addr_t &start, l4_addr_t &rsz,
00337 l4_cap_idx_t &mem_cap));
00338
00397 long attach(l4_addr_t *start, unsigned long size, Flags flags,
00398 L4::Ipc::Cap<Dataspace> mem, Offset offs = 0,
00399 unsigned char align = L4_PAGESHIFT,
00400 L4::Cap<L4::Task> const task
00401 = L4::Cap<L4::Task>::Invalid,
00402 char const *name = nullptr,
00403 Offset backing_offset = 0) const noexcept;
00404
00408 template< typename T >
00409 long attach(T **start, unsigned long size, Flags flags,
00410 L4::Ipc::Cap<Dataspace> mem, Offset offs = 0,
00411 unsigned char align = L4_PAGESHIFT,
00412 L4::Cap<L4::Task> const task
00413 = L4::Cap<L4::Task>::Invalid,
00414 char const *name = nullptr,
00415 Offset backing_offset = 0) const noexcept
00416 {
00417 union X { l4_addr_t a; T* t; };
00418 X *x = reinterpret_cast<X*>(start);
00419 return attach(&x->a, size, flags, mem, offs, align, task,
00420 name, backing_offset);
00421 }
00422
00423 #if __cplusplus >= 201103L
00434 template< typename T >
00435 class Unique_region
00436 {
00437 private:
00438 T _addr;
00439 L4::Cap<Rm> _rm;
00440
00441 public:
00442 Unique_region(Unique_region const &) = delete;
00443 Unique_region &operator = (Unique_region const &) = delete;
00444
00448 Unique_region() noexcept
00449 : _addr(0), _rm(L4::Cap<Rm>::Invalid) {}
00450
00456 explicit Unique_region(T addr) noexcept
00457 : _addr(addr), _rm(L4::Cap<Rm>::Invalid) {}
00458
00465 Unique_region(T addr, L4::Cap<Rm> const &rm) noexcept

```

```

00466 : _addr(addr), _rm(rm) {}
00467
00473 Unique_region(Unique_region &&o) noexcept : _addr(o.get()), _rm(o._rm)
00474 { o.release(); }
00475
00481 Unique_region &operator = (Unique_region &&o) noexcept
00482 {
00483 if (&o != this)
00484 {
00485 if (_rm.is_valid())
00486 _rm->detach(reinterpret_cast<l4_addr_t>(_addr), 0);
00487 _rm = o._rm;
00488 _addr = o.release();
00489 }
00490 return *this;
00491 }
00492
00498 ~Unique_region() noexcept
00499 {
00500 if (_rm.is_valid())
00501 _rm->detach(reinterpret_cast<l4_addr_t>(_addr), 0);
00502 }
00503
00509 T get() const noexcept
00510 { return _addr; }
00511
00517 T release() noexcept
00518 {
00519 _rm = L4::Cap<Rm>::Invalid;
00520 return _addr;
00521 }
00522
00529 void reset(T addr, L4::Cap<Rm> const &rm) noexcept
00530 {
00531 if (_rm.is_valid())
00532 _rm->detach(l4_addr_t(_addr), 0);
00533
00534 _rm = rm;
00535 _addr = addr;
00536 }
00537
00541 void reset() noexcept
00542 { reset(0, L4::Cap<Rm>::Invalid); }
00543
00549 bool is_valid() const noexcept
00550 { return _rm.is_valid(); }
00551
00553 T operator * () const noexcept { return _addr; }
00554
00556 T operator -> () const noexcept { return _addr; }
00557 };
00558
00559 template< typename T >
00560 long attach(Unique_region<T> *start, unsigned long size, Flags flags,
00561 L4::Ipc::Cap<Dataspace> mem, Offset offs = 0,
00562 unsigned char align = L4_PAGESHIFT,
00563 L4::Cap<L4::Task> const task
00564 = L4::Cap<L4::Task>::Invalid,
00565 char const *name = nullptr,
00566 Offset backing_offset = 0) const noexcept
00567 {
00568 l4_addr_t addr = reinterpret_cast<l4_addr_t>(start->get());
00569
00570 long res = attach(&addr, size, flags, mem, offs, align, task,
00571 name, backing_offset);
00572 if (res < 0)
00573 return res;
00574
00575 start->reset(reinterpret_cast<T>(addr), L4::Cap<Rm>(cap()));
00576 return res;
00577 }
00578 #endif
00579
00597 int detach(l4_addr_t addr, L4::Cap<Dataspace> *mem,
00598 L4::Cap<L4::Task> const &task = This_task) const noexcept;
00599
00603 int detach(void *addr, L4::Cap<Dataspace> *mem,
00604 L4::Cap<L4::Task> const &task = This_task) const noexcept;
00605
00625 int detach(l4_addr_t start, unsigned long size, L4::Cap<Dataspace> *mem,
00626 L4::Cap<L4::Task> const &task) const noexcept;
00627
00672 long find(l4_addr_t *addr, unsigned long *size, Offset *offset,
00673 L4Re::Rm::Flags *flags, L4::Cap<Dataspace> *m) noexcept
00674 { return find_t::call(c(), addr, size, flags, offset, m); }
00675
00676 L4_RPC_NF(long, find, (L4::Ipc::In_out<l4_addr_t *> addr,

```

```

00677 L4::Ipc::In_out<unsigned long *> size,
00678 L4Re::Rm::Flags *flags, Offset *offset,
00679 L4::Ipc::As_value<L4::Cap<Dataspace> > *m));
00680
00686 struct Region
00687 {
00688 l4_addr_t start;
00689 l4_addr_t end;
00690 F::Region_flags flags;
00691 };
00692
00699 struct Area
00700 {
00701 l4_addr_t start;
00702 l4_addr_t end;
00703 };
00704
00719 L4_RPC(long, get_regions, (l4_addr_t start, L4::Ipc::Ret_array<Region> regions));
00720
00735 L4_RPC(long, get_areas, (l4_addr_t start, L4::Ipc::Ret_array<Area> areas));
00736
00737 L4_RPC(long, get_info, (l4_addr_t addr, L4::Ipc::String<char> &name,
00752 Offset &backing_offset));
00753
00754 int detach(l4_addr_t start, unsigned long size, L4::Cap<Dataspace> *mem,
00755 L4::Cap<L4::Task> task, unsigned flags) const noexcept;
00756
00757 typedef L4::Typeid::Rpcs<attach_t, detach_t, find_t,
00758 reserve_area_t, free_area_t,
00759 get_regions_t, get_areas_t,
00760 get_info_t> Rpcs;
00761 };
00762
00763 inline int
00764 Rm::detach(l4_addr_t addr, L4::Cap<Dataspace> *mem,
00765 L4::Cap<L4::Task> const &task) const noexcept
00766 { return detach(addr, 1, mem, task, Detach_overlap); }
00767
00768 inline int
00770 Rm::detach(void *addr, L4::Cap<Dataspace> *mem,
00771 L4::Cap<L4::Task> const &task) const noexcept
00772 { return detach(reinterpret_cast<l4_addr_t>(addr), 1, mem, task,
00773 Detach_overlap); }
00774
00775 inline int
00777 Rm::detach(l4_addr_t addr, unsigned long size, L4::Cap<Dataspace> *mem,
00778 L4::Cap<L4::Task> const &task) const noexcept
00779 { return detach(addr, size, mem, task, Detach_exact); }
00780
00781 };
00782

```

## 17.378 l4/re/rm-sys.h File Reference

Region mapper protocol definitions.

### Namespaces

- namespace [L4Re](#)  
*L4Re C++ Interfaces.*

### Enumerations

- enum [L4Re::Rm::Opcodes](#)  
*Region-map communication-protocol opcodes.*



## 17.378.1 Detailed Description

Region mapper protocol definitions.

Definition in file [rm-sys.h](#).

## 17.379 rm-sys.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00014 namespace L4Re
00015 {
00016 namespace Rm_
00017 {
00023 enum Opcodes
00024 {
00025 Attach, Detach, Find, Attach_area, Detach_area, Get_regions, Get_areas,
00026 Get_info
00027 };
00028 };
00029 };

```

## 17.380 I4/re/util/bitmap\_cap\_alloc File Reference

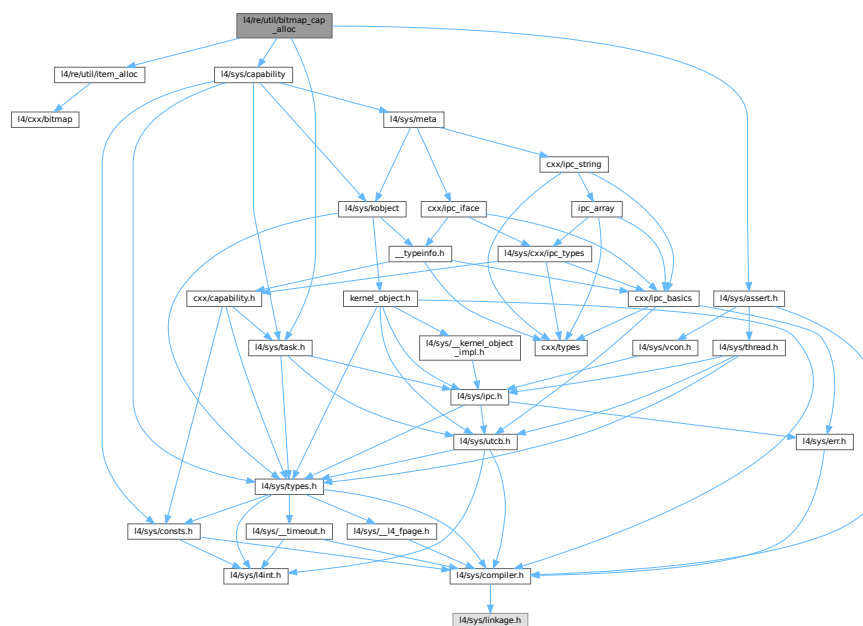
Bitmap capability allocator.

```

#include <l4/re/util/item_alloc>
#include <l4/sys/assert.h>
#include <l4/sys/capability>
#include <l4/sys/task.h>

```

Include dependency graph for bitmap\_cap\_alloc:



## Data Structures

- class [L4Re::Util::Cap\\_alloc\\_base](#)  
*Capability allocator.*

## Namespaces

- namespace [L4Re](#)  
*L4Re C++ Interfaces.*
- namespace [L4Re::Util](#)  
*Documentation of the L4 Runtime Environment utility functionality in C++.*

### 17.380.1 Detailed Description

Bitmap capability allocator.

Definition in file [bitmap\\_cap\\_alloc](#).

## 17.381 bitmap\_cap\_alloc

[Go to the documentation of this file.](#)

```

00001 // -*- Mode: C++ -*-
00002 // vim:ft=cpp
00007 /*
00008 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00009 * Alexander Warg <warg@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014
00015 #pragma once
00016
00017 #include <l4/re/util/item_alloc>
00018 #include <l4/sys/assert.h>
00019 #include <l4/sys/capability>
00020 #include <l4/sys/task.h>
00021
00022 namespace L4Re { namespace Util {
00023
00028 class Cap_alloc_base
00029 {
00030 private:
00031 long _bias;
00032 Item_alloc_base _items;
00033
00034 public:
00035 template <unsigned COUNT>
00036 struct Storage
00037 {
00038 typename Bitmap_base::Word<COUNT>::Type _bits[Bitmap_base::Word<COUNT>::Size];
00039 };
00040
00041 enum State { Free = 0, Allocated, Unknown };
00042 Cap_alloc_base(long max, void *mem, long bias = 0, void * = 0)
00043 noexcept : _bias(bias), _items(max, mem) {}
00044
00045 L4::Cap<void> alloc() noexcept
00046 {
00047 long cap = _items.alloc();
00048 if (cap < 0)
00049 return L4::Cap<void>::Invalid;
00050
00051 return L4::Cap<void>((cap + _bias) « L4_CAP_SHIFT);
00052 }
00053
00054 long hint() const { return _items.hint(); }

```

```

00055
00059 template< typename T >
00060 L4::Cap<T> alloc() noexcept
00061 { return L4::Cap<T>(alloc().cap()); }
00062
00063 State is_allocated(L4::Cap<void> c) const noexcept
00064 {
00065 long idx = (c.cap() » L4_CAP_SHIFT);
00066
00067 if (idx < _bias)
00068 return Unknown;
00069
00070 idx -= _bias;
00071 if (idx >= _items.size())
00072 return Unknown;
00073
00074 return _items.is_allocated(idx) ? Allocated : Free;
00075 }
00076
00080 template< typename T>
00081 void free(L4::Cap<T> const &cap, l4_cap_idx_t task = L4_INVALID_CAP,
00082 l4_umword_t unmap_flags = L4_FP_ALL_SPACES) noexcept
00083 {
00084 long idx = (cap.cap() » L4_CAP_SHIFT);
00085 if (idx < _bias)
00086 return;
00087
00088 idx -= _bias;
00089 if (idx >= _items.size())
00090 return;
00091
00092 l4_assert(_items.is_allocated(idx));
00093
00094 if (l4_is_valid_cap(task))
00095 l4_task_unmap(task, cap.fpage(), unmap_flags | 2);
00096
00097 _items.free(idx);
00098 }
00099
00100 // since we have no counters assume counter always > 0
00101 void take(L4::Cap<void>) noexcept {}
00102 bool release(L4::Cap<void>, l4_cap_idx_t /*task*/ = L4_INVALID_CAP,
00103 unsigned /*unmap_flags*/ = L4_FP_ALL_SPACES) noexcept
00104 { return false; }
00105
00106 long last() noexcept
00107 {
00108 return _items.size() + _bias - 1;
00109 }
00110 };
00111
00112 template< long Size >
00113 class Cap_alloc : public Cap_alloc_base
00114 {
00115 private:
00116 typename Bitmap_base::Word<Size>::Type _bits[Bitmap_base::Word<Size>::Size];
00117
00118 public:
00119 explicit Cap_alloc(long bias = 0) noexcept
00120 : Cap_alloc_base(Size, _bits, bias) {}
00121
00122 };
00123
00124 }
00125 }

```

## 17.382 br\_manager

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2014 Alexander Warg <alexander.warg@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #pragma once
00009
00010 #include <l4/re/util/cap_alloc>
00011 #include <l4/sys/cxx/ipc_server_loop>
00012 #include <l4/cxx/ipc_timeout_queue>
00013 #include <l4/sys/assert.h>
00014
00015 namespace L4Re { namespace Util {

```

```

00016
00025 class Br_manager : public L4::Ipc_svr::Server_iface
00026 {
00027 private:
00028 enum { _mem = 0, _ports = 0 };
00029 enum { Brs_per_timeout = sizeof(l4_kernel_clock_t) / sizeof(l4_umword_t) };
00030
00031 public:
00033 Br_manager() : _caps(0), _cap_flags(L4_RCV_ITEM_LOCAL_ID) {}
00034
00035 Br_manager(Br_manager const &) = delete;
00036 Br_manager &operator = (Br_manager const &) = delete;
00037
00038 Br_manager(Br_manager &&) = delete;
00039 Br_manager &operator = (Br_manager &&) = delete;
00040
00041 ~Br_manager()
00042 {
00043 // Slots for received capabilities are placed at the beginning of the
00044 // (shadowed) buffer registers. Free those.
00045 for (unsigned i = 0; i < _caps; ++i)
00046 cap_alloc.free(L4::Cap<void>(_brs[i] & L4_CAP_MASK));
00047 }
00048
00049 /*
00050 * This implementation dynamically manages assignment of buffer registers for
00051 * the necessary amount of receive buffers allocated by all calls to this
00052 * function.
00053 */
00054 int alloc_buffer_demand(Demand const &d) override
00055 {
00056 using L4::Ipc::Small_buf;
00057
00058 // memory and IO port receive windows currently not supported
00059 if (d.mem || d.ports)
00060 return -L4_EINVAL;
00061
00062 // take extra buffers for a possible timeout and for a zero terminator
00063 if (d.caps + d.mem * 2 + d.ports * 2 + Brs_per_timeout + 1
00064 > L4_UTCB_GENERIC_BUFFERS_SIZE)
00065 return -L4_ERANGE;
00066
00067 if (d.caps > _caps)
00068 {
00069 while (_caps < d.caps)
00070 {
00071 L4::Cap<void> cap = cap_alloc.alloc();
00072 if (!cap)
00073 return -L4_ENOMEM;
00074
00075 reinterpret_cast<Small_buf*>(_brs[_caps])
00076 = Small_buf(cap.cap(), _cap_flags);
00077 ++_caps;
00078 }
00079 _brs[_caps] = 0;
00080 }
00081
00082 return L4_EOK;
00083 }
00084
00085
00086 L4::Cap<void> get_rcv_cap(int i) const override
00087 {
00088 if (i < 0 || i >= _caps)
00089 return L4::Cap<void>::Invalid;
00090
00091 return L4::Cap<void>(_brs[i] & L4_CAP_MASK);
00092 }
00093
00094 int realloc_rcv_cap(int i) override
00095 {
00096 using L4::Ipc::Small_buf;
00097
00098 if (i < 0 || i >= _caps)
00099 return -L4_EINVAL;
00100
00101 L4::Cap<void> cap = cap_alloc.alloc();
00102 if (!cap)
00103 return -L4_ENOMEM;
00104
00105 reinterpret_cast<Small_buf*>(_brs[i])
00106 = Small_buf(cap.cap(), _cap_flags);
00107
00108 return L4_EOK;
00109 }
00110
00118 void set_rcv_cap_flags(unsigned long flags)

```

```

00119 {
00120 l4_assert(_caps == 0);
00121
00122 _cap_flags = flags;
00123 }
00124
00126 int add_timeout(L4::Ipc_svr::Timeout *, l4_kernel_clock_t) override
00127 { return -L4_ENOSYS; }
00128
00130 int remove_timeout(L4::Ipc_svr::Timeout *) override
00131 { return -L4_ENOSYS; }
00132
00134 void setup_wait(l4_utcb_t *utcb, L4::Ipc_svr::Reply_mode)
00135 {
00136 l4_buf_regs_t *br = l4_utcb_br_u(utcb);
00137 br->bdr = 0;
00138 for (unsigned i = 0; i <= _caps; ++i)
00139 br->br[i] = _brs[i];
00140 }
00141
00142 protected:
00144 unsigned first_free_br() const
00145 {
00146 // The last BR (64-bit) or the last two BRs (32-bit); this is constant.
00147 return L4_UTCB_GENERIC_BUFFERS_SIZE - Brs_per_timeout;
00148 // We could also do the following dynamic approach:
00149 // return _caps + _mem + _ports + 1
00150 }
00151
00152 private:
00153 unsigned short _caps;
00154 unsigned long _cap_flags;
00155
00156 l4_umword_t _brs[L4_UTCB_GENERIC_BUFFERS_SIZE];
00157 };
00158
00165 struct Br_manager_hooks
00166 : L4::Ipc_svr::Ignore_errors,
00167 L4::Ipc_svr::Default_timeout,
00168 L4::Ipc_svr::Compound_reply,
00169 Br_manager
00170 {};
00171
00179 struct Br_manager_timeout_hooks :
00180 public L4::Ipc_svr::Timeout_queue_hooks<Br_manager_timeout_hooks, Br_manager>,
00181 public L4::Ipc_svr::Ignore_errors
00182 {
00183 public:
00184 static l4_kernel_clock_t now()
00185 { return l4_kip_clock(l4re_kip()); }
00186 };
00187
00188 }}
00189

```

## 17.383 l4/re/util/cap File Reference

Capability utility functions.



## 17.384 cap

[Go to the documentation of this file.](#)

```

00001 // -*- Mode: C++ -*-
00002 // vim:ft=cpp
00007 /*
00008 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013
00014 #pragma once
00015
00016 #include <l4/sys/task>
00017
00018 namespace L4Re { namespace Util {
00019
00027 L4_CV static inline l4_msgtag_t cap_release(L4::Cap<void> cap)
00028 {
00029 return l4_task_unmap(L4_BASE_TASK_CAP,
00030 l4_obj_fpage(cap.cap(), 0, L4_FPAGE_RWX),
00031 L4_FP_ALL_SPACES);
00032 }
00033
00034 }}

```

## 17.385 l4/re/cap\_alloc File Reference

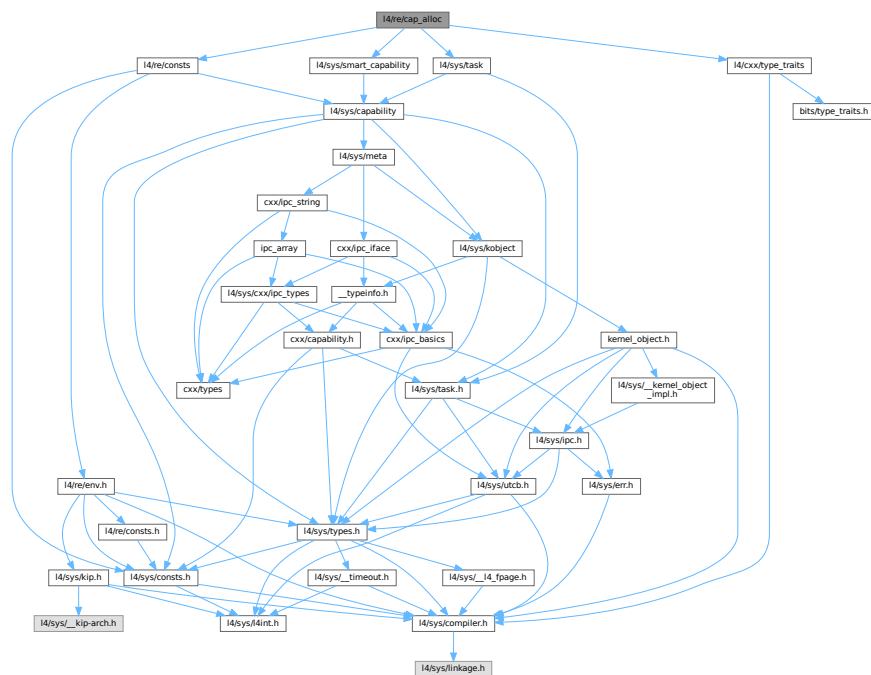
Abstract capability-allocator interface.

```

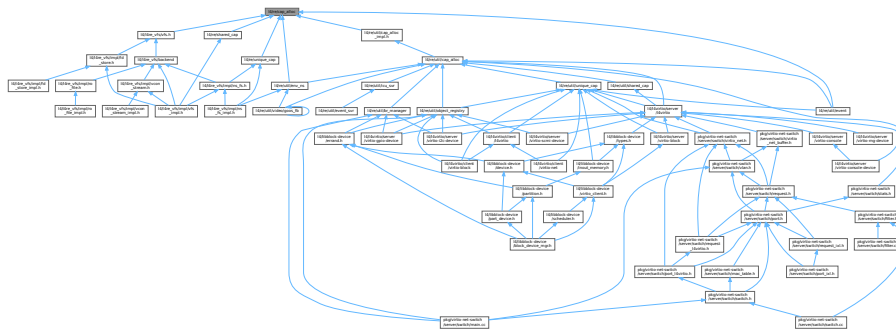
#include <l4/sys/task>
#include <l4/sys/smart_capability>
#include <l4/re/consts>
#include <l4/cxx/type_traits>

```

Include dependency graph for cap\_alloc:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class [L4Re::Cap\\_alloc](#)  
*Capability allocator interface.*
- class [L4Re::Smart\\_cap\\_auto< Unmap\\_flags >](#)  
*Helper for [Unique\\_cap](#) and [Unique\\_del\\_cap](#).*
- class [L4Re::Smart\\_count\\_cap< Unmap\\_flags >](#)  
*Helper for [Ref\\_cap](#) and [Ref\\_del\\_cap](#).*

## Namespaces

- namespace [L4Re](#)  
*[L4Re](#) C++ Interfaces.*

## 17.385.1 Detailed Description

Abstract capability-allocator interface.

Definition in file [cap\\_alloc](#).

## 17.386 cap\_alloc

[Go to the documentation of this file.](#)

```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 #pragma once
00010 #include <l4/sys/task>
00011 #include <l4/sys/smart_capability>
00012 #include <l4/re/consts>
00013 #include <l4/cxx/type_traits>
00014 namespace L4Re {
00015
```



```

00030 class Cap_alloc
00031 {
00032 private:
00033 void operator = (Cap_alloc const &);
00034
00035 protected:
00036 Cap_alloc(Cap_alloc const &) {}
00037 Cap_alloc() {}
00038
00039 public:
00040
00041 virtual L4::Cap<void> alloc() noexcept = 0;
00042 virtual void take(L4::Cap<void> cap) noexcept = 0;
00043
00044 template< typename T >
00045 L4::Cap<T> alloc() noexcept
00046 { return L4::cap_cast<T>(alloc()); }
00047
00048 virtual void free(L4::Cap<void> cap, l4_cap_idx_t task = L4_INVALID_CAP,
00049 unsigned unmap_flags = L4_FP_ALL_SPACES) noexcept = 0;
00050 virtual bool release(L4::Cap<void> cap, l4_cap_idx_t task = L4_INVALID_CAP,
00051 unsigned unmap_flags = L4_FP_ALL_SPACES) noexcept = 0;
00052
00053 virtual ~Cap_alloc() = 0;
00054 };
00055
00056 template<typename ALLOC>
00057 struct Cap_alloc_t : ALLOC, L4Re::Cap_alloc
00058 {
00059 template<typename ...ARGS>
00060 Cap_alloc_t(ARGS &&...args) : ALLOC(cxx::forward<ARGS>(args)...) {}
00061
00062 L4::Cap<void> alloc() noexcept override { return ALLOC::alloc(); }
00063 void take(L4::Cap<void> cap) noexcept override { ALLOC::take(cap); }
00064
00065 template <typename T>
00066 L4::Cap<T> alloc() noexcept
00067 {
00068 return L4::cap_cast<T>(alloc());
00069 }
00070
00071 void free(L4::Cap<void> cap, l4_cap_idx_t task = L4_INVALID_CAP,
00072 unsigned unmap_flags = L4_FP_ALL_SPACES) noexcept override
00073 { ALLOC::free(cap, task, unmap_flags); }
00074
00075 bool release(L4::Cap<void> cap, l4_cap_idx_t task,
00076 unsigned unmap_flags) noexcept override
00077 { return ALLOC::release(cap, task, unmap_flags); }
00078
00079 void operator delete(void *) {}
00080 };
00081
00082 inline
00083 Cap_alloc::~~Cap_alloc()
00084 {}
00085
00086 extern Cap_alloc *virt_cap_alloc;
00087
00088 template< unsigned long Unmap_flags = L4_FP_ALL_SPACES >
00089 class Smart_cap_auto
00090 {
00091 private:
00092 Cap_alloc *_ca;
00093
00094 public:
00095 Smart_cap_auto() : _ca(0) {}
00096 Smart_cap_auto(Cap_alloc *ca) : _ca(ca) {}
00097
00098 void free(L4::Cap_base &c)
00099 {
00100 if (c.is_valid() && _ca)
00101 _ca->free(L4::Cap<void>(c.cap()), This_task, Unmap_flags);
00102 invalidate(c);
00103 }
00104
00105 static void invalidate(L4::Cap_base &c)
00106 {
00107 if (c.is_valid())
00108 c.invalidate();
00109 }
00110 };
00111
00112 template< unsigned long Unmap_flags = L4_FP_ALL_SPACES >
00113 class Smart_count_cap
00114 {

```





## 17.387.1 Detailed Description

Capability allocator.

Definition in file [cap\\_alloc](#).

## 17.388 cap\_alloc

[Go to the documentation of this file.](#)

```

00001 // -*- Mode: C++ -*-
00002 // vim:ft=cpp
00003 /*
00004 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00005 * Alexander Warg <warg@os.inf.tu-dresden.de>
00006 * economic rights: Technische Universität Dresden (Germany)
00007 *
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010
00011 #pragma once
00012
00013 #include <l4/re/util/cap_alloc_impl.h>
00014 #include <l4/sys/smart_capability>
00015 #include <l4/sys/task>
00016 #include <l4/re/consts>
00017
00018 namespace L4Re { namespace Util {
00019
00020 extern _Cap_alloc &cap_alloc;
00021
00022 template< unsigned long Unmap_flags = L4_FP_ALL_SPACES >
00023 class Smart_cap_auto
00024 {
00025 public:
00026 static void free(L4::Cap_base &c)
00027 {
00028 if (c.is_valid())
00029 {
00030 cap_alloc.free(L4::Cap<void>(c.cap()), This_task, Unmap_flags);
00031 c.invalidate();
00032 }
00033 }
00034
00035 static void invalidate(L4::Cap_base &c)
00036 {
00037 if (c.is_valid())
00038 c.invalidate();
00039 }
00040 };
00041
00042 template< unsigned long Unmap_flags = L4_FP_ALL_SPACES >
00043 class Smart_count_cap
00044 {
00045 public:
00046 static void free(L4::Cap_base &c) noexcept
00047 {
00048 if (c.is_valid())
00049 {
00050 if (cap_alloc.release(L4::Cap<void>(c.cap()), This_task, Unmap_flags))
00051 c.invalidate();
00052 }
00053 }
00054
00055 static void invalidate(L4::Cap_base &c) noexcept
00056 {
00057 if (c.is_valid())
00058 c.invalidate();
00059 }
00060
00061 static L4::Cap_base copy(L4::Cap_base const &src)
00062 {
00063 cap_alloc.take(L4::Cap<void>(src.cap()));
00064 return src;
00065 }
00066 };
00067
00068 };
00069
00070
00071
00072
00073
00074
00075
00076
00077
00078
00079
00080
00081
00082
00083
00084
00085
00086
00087
00088
00089
00090
00091
00092
00093
00094
00095
00096
00097
00098
00099
00100
00101
00102
00103
00104
00105
00106
00107
00108
00109
00110

```

```

00111
00141 template< typename T >
00142 struct Ref_cap
00143 {
00144 typedef L4::Smart_cap<T, Smart_count_cap<L4_FP_ALL_SPACES> > Cap;
00145 };
00146
00182 template< typename T >
00183 struct Ref_del_cap
00184 {
00185 typedef L4::Smart_cap<T, Smart_count_cap<L4_FP_DELETE_OBJ> > Cap;
00186 };
00187
00193 template< typename T >
00194 typename Ref_cap<T>::Cap
00195 make_ref_cap() { return typename Ref_cap<T>::Cap(cap_alloc.alloc<T>()); }
00196
00202 template< typename T >
00203 typename Ref_del_cap<T>::Cap
00204 make_ref_del_cap()
00205 { return typename Ref_del_cap<T>::Cap(cap_alloc.alloc<T>()); }
00206
00208
00209 }}
00210

```

## 17.389 l4/re/util/cap\_alloc\_impl.h File Reference

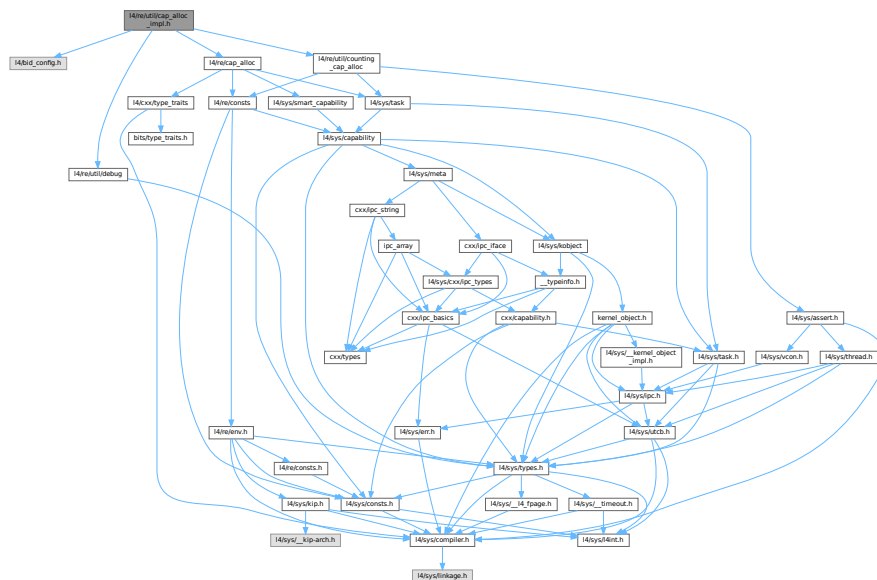
Capability allocator implementation.

```

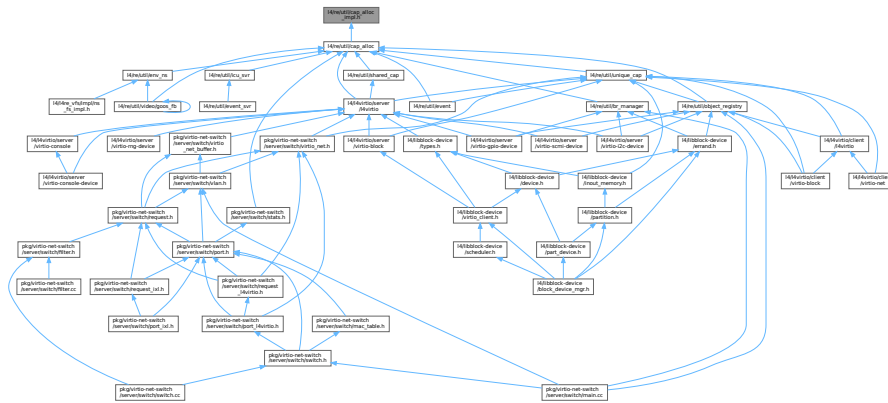
#include <l4/bid_config.h>
#include <l4/re/cap_alloc>
#include <l4/re/util/counting_cap_alloc>
#include <l4/re/util/debug>

```

Include dependency graph for cap\_alloc\_impl.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class [L4Re::Util::Cap\\_alloc](#)

Adapter to expose the cap allocator implementation as [L4Re::Cap\\_alloc](#) compatible class.

## Namespaces

- namespace [L4Re](#)  
[L4Re C++ Interfaces.](#)
- namespace [L4Re::Util](#)

Documentation of the [L4 Runtime Environment](#) utility functionality in C++.

## 17.389.1 Detailed Description

Capability allocator implementation.

Definition in file [cap\\_alloc\\_impl.h](#).

## 17.390 cap\_alloc\_impl.h

[Go to the documentation of this file.](#)

```
00001 // -*- Mode: C++ -*-
00002 // vim:ft=cpp
00003 /*
00004 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009
00010 #pragma once
00011
00012 #include <l4/bid_config.h>
00013 #include <l4/re/cap_alloc>
00014
00015 #if defined(CONFIG_L4RE_BITMAP_CAP_ALLOC)
00016 #include <l4/re/util/bitmap_cap_alloc>
00017 #endif
```

```

00023 namespace L4Re { namespace Util {
00024
00025 using _Cap_alloc_impl = Cap_alloc_base;
00026
00027 }}
00028
00029 #elif defined(CONFIG_L4RE_COUNTING_CAP_ALLOC)
00030
00031 #include <l4/re/util/counting_cap_alloc>
00032 #include <l4/re/util/debug>
00033
00034 namespace L4Re { namespace Util {
00035
00036 // RISC-V does not natively support subword atomics, such as __atomic_load_1.
00037 // The RISC-V gcc developers have decided to emulate these via libatomic, which
00038 // is automatically linked against.
00039 #if defined(__GCC_HAVE_SYNC_COMPARE_AND_SWAP_1) || defined(ARCH_arm) || defined(ARCH_riscv)
00040 using _Cap_alloc_impl
00041 = Counting_cap_alloc<L4Re::Util::Counter_atomic<unsigned char>,
00042 L4Re::Util::Dbg>;
00043 #elif defined(ARCH_sparc)
00044 using _Cap_alloc_impl
00045 = Counting_cap_alloc<L4Re::Util::Counter<unsigned char>,
00046 L4Re::Util::Dbg>;
00047 #warning "Thread-safe capability allocator not available!"
00048 #else
00049 #error "Unsupported platform"
00050 #endif
00051
00052 }}
00053
00054 #else
00055 #error "No supported capability allocator selected"
00056 #endif
00057
00058 namespace L4Re { namespace Util {
00059
00060 class _Cap_alloc final : public L4Re::Cap_alloc, private _Cap_alloc_impl
00061 {
00062 public:
00063 template <unsigned COUNT>
00064 using Storage = _Cap_alloc_impl::Storage<COUNT>;
00065
00066 using _Cap_alloc_impl::_Cap_alloc_impl; // Expose underlying constructor
00067 void operator delete(void *) {} // Prevent global operator delete reference
00068
00069 L4::Cap<void> alloc() noexcept override
00070 { return _Cap_alloc_impl::alloc(); }
00071
00072 template< typename T >
00073 L4::Cap<T> alloc() noexcept
00074 { return L4::cap_cast<T>(alloc()); }
00075
00076 void take(L4::Cap<void> cap) noexcept override
00077 { _Cap_alloc_impl::take(cap); }
00078
00079 void free(L4::Cap<void> cap, l4_cap_idx_t task = L4_INVALID_CAP,
00080 unsigned unmap_flags = L4_FP_ALL_SPACES) noexcept override
00081 { _Cap_alloc_impl::free(cap, task, unmap_flags); }
00082
00083 bool release(L4::Cap<void> cap, l4_cap_idx_t task,
00084 unsigned unmap_flags) noexcept override
00085 { return _Cap_alloc_impl::release(cap, task, unmap_flags); }
00086
00087 using _Cap_alloc_impl::last;
00088 };
00089
00090
00091
00092
00093
00094
00095
00096

```

## 17.391 l4/re/util/counting\_cap\_alloc File Reference

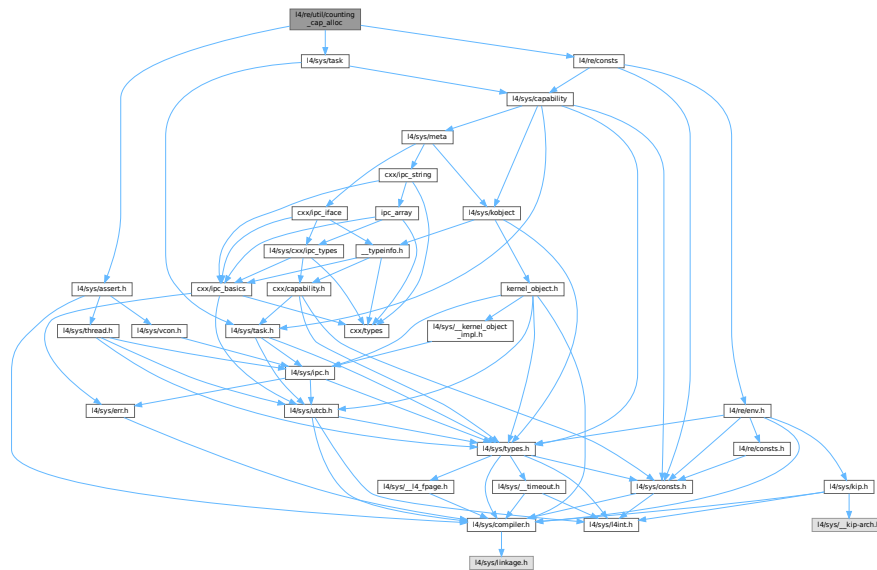
Reference-counting capability allocator.

```

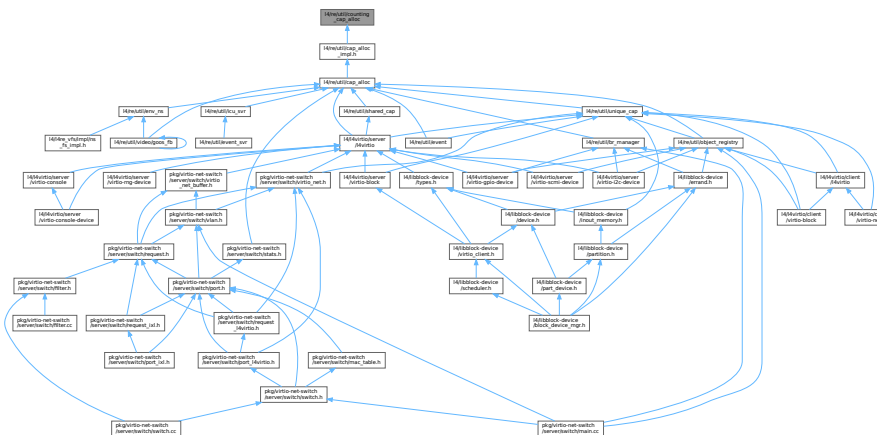
#include <l4/sys/task>
#include <l4/sys/assert.h>

```

```
#include <linux/re/consts>
Include dependency graph for counting_cap_alloc:
```



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct `L4Re::Util::Counter< COUNTER >`  
*Counter for `Counting_cap_alloc` with variable data width.*
- struct `L4Re::Util::Counter_atomic< COUNTER >`  
*Thread safe version of counter for `Counting_cap_alloc`.*
- class `L4Re::Util::Counting_cap_alloc< COUNTERTYPE, Dbg >`  
*Internal reference-counting cap allocator.*



## Namespaces

- namespace [L4Re](#)  
*[L4Re](#) C++ Interfaces.*
- namespace [L4Re::Util](#)  
*Documentation of the [L4](#) Runtime Environment utility functionality in C++.*

### 17.391.1 Detailed Description

Reference-counting capability allocator.

Definition in file [counting\\_cap\\_alloc](#).

## 17.392 counting\_cap\_alloc

[Go to the documentation of this file.](#)

```

00001 // vim:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2008-2010 Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012
00013 #pragma once
00014
00015 #include <l4/sys/task>
00016 #include <l4/sys/assert.h>
00017 #include <l4/re/consts>
00018
00019 namespace L4Re { namespace Util {
00020
00026 template< typename COUNTER = unsigned char >
00027 struct Counter
00028 {
00029 typedef COUNTER Type;
00030 Type _cnt;
00031
00032 static Type nil() { return 0; }
00033 static Type unused() { return 0; }
00034
00035 void free() { _cnt = 0; }
00036 bool is_free() const { return _cnt == 0; }
00037 bool is_saturated() const { return static_cast<Type>(_cnt + 1) == 0; }
00038
00050 bool inc()
00051 {
00052 if (is_saturated())
00053 return true; // no change and no warning
00054 ++_cnt;
00055 if (is_saturated())
00056 return false; // warn caller that counter is now saturated
00057 else
00058 return true; // success
00059 }
00060
00067 Type dec()
00068 {
00069 if (is_saturated())
00070 return _cnt; // no change
00071 else
00072 return --_cnt; // success
00073 }
00074
00075 bool try_alloc()
00076 {
00077 if (_cnt == 0)
00078 {
00079 _cnt = 1;
00080 return true;
00081 }
00082 return false;

```

```

00083 }
00084 };
00085
00097 template< typename COUNTER = unsigned char >
00098 struct Counter_atomic
00099 {
00100 typedef COUNTER Type;
00101 Type _cnt;
00102
00103 static Type nil() { return 0; }
00104 static Type unused() { return 1; }
00105
00106 bool is_free() const { return __atomic_load_n(&_cnt, __ATOMIC_RELAXED) == 0; }
00107 static bool is_saturated(Type cnt) { return static_cast<Type>(cnt + 1) == 0; }
00108
00109 bool try_alloc()
00110 {
00111 Type expected = nil();
00112 // Use "acquire" memory ordering. Any operations tied to the capability slot
00113 // must only be observable after the slot has been occupied.
00114 return __atomic_compare_exchange_n(&_cnt, &expected, 2, false,
00115 __ATOMIC_ACQUIRE, __ATOMIC_RELAXED);
00116 }
00117
00121 bool inc()
00122 {
00123 Type old_cnt = __atomic_load_n(&_cnt, __ATOMIC_RELAXED);
00124 Type new_cnt;
00125 do
00126 {
00127 if (is_saturated(old_cnt))
00128 return true; // no change and no warning
00129 new_cnt = old_cnt + 1;
00130 }
00131 while (!__atomic_compare_exchange_n(&_cnt, &old_cnt, new_cnt, false,
00132 __ATOMIC_RELAXED, __ATOMIC_RELAXED));
00133 if (is_saturated(new_cnt))
00134 return false; // warn caller that counter is now saturated
00135 else
00136 return true; // success
00137 }
00138
00142 Type dec()
00143 {
00144 Type old_cnt = __atomic_load_n(&_cnt, __ATOMIC_RELAXED);
00145 Type new_cnt;
00146 do
00147 {
00148 if (is_saturated(old_cnt))
00149 return old_cnt; // no change
00150 new_cnt = old_cnt - 1;
00151 }
00152 while (!__atomic_compare_exchange_n(&_cnt, &old_cnt, new_cnt, false,
00153 __ATOMIC_RELAXED, __ATOMIC_RELAXED));
00154 return new_cnt; // success
00155 }
00156
00157 void free()
00158 {
00159 // Use "release" memory ordering to make sure that any operations tied to
00160 // the capability slot are observable by other threads before the slot can
00161 // be reused.
00162 __atomic_store_n(&_cnt, 0, __ATOMIC_RELEASE);
00163 }
00164 };
00165
00190 template <typename COUNTERTYPE, typename Dbg>
00191 class Counting_cap_alloc
00192 {
00193 private:
00194 void operator = (Counting_cap_alloc const &) { }
00195 typedef COUNTERTYPE Counter;
00196
00197 COUNTERTYPE *_items;
00198 long _free_hint;
00199 long _bias;
00200 long _capacity;
00201 Dbg *_dbg;
00202
00203 public:
00204
00205 template <unsigned COUNT>
00206 struct Storage
00207 {
00208 COUNTERTYPE _buf[COUNT];
00209 typedef COUNTERTYPE Buf_type[COUNT];
00210 enum { Size = COUNT };

```

```

00211 };
00212
00213 Counting_cap_alloc(long capacity, void *m, long bias, Dbg *dbg) noexcept
00214 : _items((Counter*)m), _free_hint(0), _bias(bias), _capacity(capacity),
00215 _dbg(dbg)
00216 {}
00217
00218 protected:
00224 Counting_cap_alloc() noexcept
00225 : _items(0), _free_hint(0), _bias(0), _capacity(0)
00226 {}
00227
00242 void setup(void *m, long capacity, long bias, Dbg *dbg) noexcept
00243 {
00244 _items = static_cast<Counter*>(m);
00245 _capacity = capacity;
00246 _bias = bias;
00247 _dbg = dbg;
00248 }
00249
00250 public:
00257 L4::Cap<void> alloc() noexcept
00258 {
00259 long free_hint = __atomic_load_n(&_free_hint, __ATOMIC_RELAXED);
00260
00261 for (long i = free_hint; i < _capacity; ++i)
00262 if (_items[i].try_alloc())
00263 {
00264 _free_hint = i + 1;
00265 return L4::Cap<void>((i + _bias) « L4_CAP_SHIFT);
00266 }
00267
00268 // _free_hint is not necessarily correct in case of multi-threading! Make
00269 // sure we don't miss any potentially free slots.
00270 for (long i = 0; i < free_hint && i < _capacity; ++i)
00271 if (_items[i].try_alloc())
00272 {
00273 _free_hint = i + 1;
00274 return L4::Cap<void>((i + _bias) « L4_CAP_SHIFT);
00275 }
00276
00277 return L4::Cap<void>::Invalid;
00278 }
00279
00281 template <typename T>
00282 L4::Cap<T> alloc() noexcept
00283 {
00284 return L4::cap_cast<T>(alloc());
00285 }
00286
00287
00296 void take(L4::Cap<void> cap) noexcept
00297 {
00298 long c;
00299 if (!range_check_and_get_idx(cap, &c))
00300 return;
00301
00302 if (!L4_UNLIKELY(_items[c].inc()))
00303 _dbg->printf("Warning: Reference counter of cap 0x%lx now saturated!\n",
00304 cap.cap() » L4_CAP_SHIFT);
00305 }
00306
00307
00321 bool free(L4::Cap<void> cap, l4_cap_idx_t task = L4_INVALID_CAP,
00322 unsigned unmap_flags = L4_FP_ALL_SPACES) noexcept
00323 {
00324 long c;
00325 if (!range_check_and_get_idx(cap, &c))
00326 return false;
00327
00328 l4_assert(!_items[c].is_free());
00329
00330 if (l4_is_valid_cap(task))
00331 l4_task_unmap(task, cap.fpage(), unmap_flags);
00332
00333 if (c < _free_hint)
00334 _free_hint = c;
00335
00336 _items[c].free();
00337
00338 return true;
00339 }
00340
00359 bool release(L4::Cap<void> cap, l4_cap_idx_t task = L4_INVALID_CAP,
00360 unsigned unmap_flags = L4_FP_ALL_SPACES) noexcept
00361 {
00362 long c;

```

```

00363 if (!range_check_and_get_idx(cap, &c))
00364 return false;
00365
00366 l4_assert(!_items[c].is_free());
00367
00368 if (_items[c].dec() == Counter::unused())
00369 {
00370 if (task != L4_INVALID_CAP)
00371 l4_task_unmap(task, cap.fpage(), unmap_flags);
00372
00373 if (c < _free_hint)
00374 _free_hint = c;
00375
00376 // Let others allocate this slot only after the l4_task_unmap() has
00377 // finished.
00378 _items[c].free();
00379
00380 return true;
00381 }
00382 return false;
00383 }
00384
00388 long last() noexcept
00389 {
00390 return _capacity + _bias - 1;
00391 }
00392
00393 private:
00394 bool range_check_and_get_idx(L4::Cap<void> cap, long *c)
00395 {
00396 *c = cap.cap() » L4_CAP_SHIFT;
00397 if (*c < _bias)
00398 return false;
00399
00400 *c -= _bias;
00401
00402 return *c < _capacity;
00403 }
00404 };
00405
00406 }}

```

## 17.393 dataspace\_svr

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00005 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00006 * economic rights: Technische Universität Dresden (Germany)
00007 *
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010 #pragma once
00011
00012 #include <string.h>
00013 #include <stddef.h>
00014 #include <l4/bid_config.h>
00015 #include <l4/sys/types.h>
00016 #include <l4/cxx/minmax>
00017 #include <l4/re/dataspace>
00018 #include <l4/re/dataspace-sys.h>
00019 #include <l4/sys/cxx/ipc_legacy>
00020
00021 namespace L4Re { namespace Util {
00022
00029 class Dataspace_svr
00030 {
00031 public:
00032 L4_RPC_LEGACY_DISPATCH(L4Re::Dataspace);
00033
00034 typedef L4::IpC::Snd_fpage::Map_type Map_type;
00035 typedef L4::IpC::Snd_fpage::Cacheopt Cache_type;
00036
00037 Dataspace_svr() noexcept
00038 : _ds_start(0), _ds_size(0), _map_flags(L4::IpC::Snd_fpage::Map),
00039 _cache_flags(L4::IpC::Snd_fpage::Cached)
00040 {}
00041
00042 virtual ~Dataspace_svr() noexcept {}
00043
00057 int map(Dataspace::Offset offset,
00058 Dataspace::Map_addr local_addr,

```

```

00059 Dataspace::Flags flags,
00060 Dataspace::Map_addr min_addr,
00061 Dataspace::Map_addr max_addr,
00062 L4::Ipc::Snd_fpage &memory)
00063 {
00064 memory = L4::Ipc::Snd_fpage();
00065
00066 offset = l4_trunc_page(offset);
00067 local_addr = l4_trunc_page(local_addr);
00068
00069 if (!check_limit(offset))
00070 {
00071 #if 0
00072 printf("limit failed: off=%lx sz=%lx\n", offset, size());
00073 #endif
00074 return -L4_ERANGE;
00075 }
00076
00077 min_addr = l4_trunc_page(min_addr);
00078 max_addr = l4_round_page(max_addr);
00079
00080 l4_addr_t addr = _ds_start + offset;
00081 unsigned char order = L4_PAGESHIFT;
00082
00083 while (order < 30 /* limit to 1GB flexpage */)
00084 {
00085 l4_addr_t map_base = l4_trunc_size(addr, order + 1);
00086 if (map_base < _ds_start)
00087 break;
00088
00089 if (map_base + (1UL << (order + 1)) - 1 > (_ds_start + round_size() - 1))
00090 break;
00091
00092 map_base = l4_trunc_size(local_addr, order + 1);
00093 if (map_base < min_addr)
00094 break;
00095
00096 if (map_base + (1UL << (order + 1)) - 1 > max_addr - 1)
00097 break;
00098
00099 l4_addr_t mask = ~(~0UL << (order + 1));
00100 if (local_addr == ~0UL || ((addr ^ local_addr) & mask))
00101 break;
00102
00103 ++order;
00104 }
00105
00106 l4_addr_t map_base = l4_trunc_size(addr, order);
00107
00108 Dataspace::Map_addr b = map_base;
00109 unsigned send_order = order;
00110 int err = map_hook(offset /*map_base - _ds_start*/, order, flags,
00111 &b, &send_order);
00112 if (err < 0)
00113 return err;
00114
00115 l4_fpage_t fpage = l4_fpage(b, send_order, flags.fpage_rights());
00116
00117 memory = L4::Ipc::Snd_fpage(fpage, local_addr, _map_flags, _cache_flags);
00118
00119 return L4_EOK;
00120 }
00121
00136 virtual int map_hook([[maybe_unused]] Dataspace::Offset offs,
00137 [[maybe_unused]] unsigned order,
00138 [[maybe_unused]] Dataspace::Flags flags,
00139 [[maybe_unused]] Dataspace::Map_addr *base,
00140 [[maybe_unused]] unsigned *send_order)
00141 {
00142 return 0;
00143 }
00144
00150 virtual void take() noexcept
00151 {}
00152
00160 virtual unsigned long release() noexcept
00161 { return 0; }
00162
00175 virtual long copy([[maybe_unused]] l4_addr_t dst_offs,
00176 [[maybe_unused]] l4_umword_t src_id,
00177 [[maybe_unused]] l4_addr_t src_offs,
00178 [[maybe_unused]] unsigned long size) noexcept
00179 {
00180 return -L4_ENODEV;
00181 }
00182
00192 virtual long clear(unsigned long offs, unsigned long size) const noexcept

```

```

00193 {
00194 if (!check_limit(offsets))
00195 return -L4_ERANGE;
00196
00197 unsigned long sz = size = cxx::min(size, round_size() - offsets);
00198
00199 while (sz)
00200 {
00201 unsigned long b_addr = _ds_start + offsets;
00202 unsigned long b_sz = cxx::min(size - offsets, sz);
00203
00204 memset(reinterpret_cast<void *>(b_addr), 0, b_sz);
00205
00206 offsets += b_sz;
00207 sz -= b_sz;
00208 }
00209
00210 return 0;
00211 }
00212
00224 virtual long allocate([[maybe_unused]] l4_addr_t offset,
00225 [[maybe_unused]] l4_size_t size,
00226 [[maybe_unused]] unsigned access) noexcept
00227 {
00228 return -L4_ENODEV;
00229 }
00230
00236 virtual unsigned long page_shift() const noexcept
00237 { return L4_LOG2_PAGESIZE; }
00238
00244 virtual bool is_static() const noexcept
00245 { return true; }
00246
00259 virtual long map_info([[maybe_unused]] l4_addr_t &start_addr,
00260 [[maybe_unused]] l4_addr_t &end_addr) noexcept
00261 { return -L4_EPERM; }
00262
00263
00264 long op_map(L4Re::Dataspace::Rights rights,
00265 L4Re::Dataspace::Offset offset,
00266 L4Re::Dataspace::Map_addr spot,
00267 L4Re::Dataspace::Flags flags,
00268 L4::Ipc::Snd_fpage &fp)
00269 {
00270 auto rf = map_flags(rights);
00271
00272 if (!rf.w() && flags.w())
00273 return -L4_EPERM;
00274
00275 return map(offset, spot, flags & rf, 0, ~0, fp);
00276 }
00277
00278 long op_allocate(L4Re::Dataspace::Rights rights,
00279 L4Re::Dataspace::Offset offset,
00280 L4Re::Dataspace::Size size)
00281 { return allocate(offset, size, rights & 3); }
00282
00283 long op_copy_in(L4Re::Dataspace::Rights rights,
00284 L4Re::Dataspace::Offset dst_offs,
00285 L4::Ipc::Snd_fpage const &src_cap,
00286 L4Re::Dataspace::Offset src_offs,
00287 L4Re::Dataspace::Size sz)
00288 {
00289 if (!src_cap.id_received())
00290 return -L4_EINVAL;
00291
00292 if (!(rights & L4_CAP_FPAGE_W))
00293 return -L4_EACCESS;
00294
00295 if (sz == 0)
00296 return L4_EOK;
00297
00298 return copy(dst_offs, src_cap.data(), src_offs, sz);
00299 }
00300
00301 long op_info(L4Re::Dataspace::Rights rights, L4Re::Dataspace::Stats &s)
00302 {
00303 s.size = size();
00304 // only return writable if really writable
00305 s.flags = Dataspace::Flags(0);
00306 if (map_flags(rights).w())
00307 s.flags |= Dataspace::F::W;
00308 return L4_EOK;
00309 }
00310
00311 long op_clear(L4Re::Dataspace::Rights rights,
00312 L4Re::Dataspace::Offset offset,

```

```

00313 L4Re::Dataspace::Size size)
00314 {
00315 if (!map_flags(rights).w())
00316 return -L4_EACCESS;
00317
00318 return clear(offset, size);
00319 }
00320
00321 long op_map_info(L4Re::Dataspace::Rights,
00322 [[maybe_unused]] l4_addr_t &start_addr,
00323 [[maybe_unused]] l4_addr_t &end_addr)
00324 {
00325 #ifdef CONFIG_MMU
00326 return 0;
00327 #else
00328 return map_info(start_addr, end_addr);
00329 #endif
00330 }
00331
00332 protected:
00333 unsigned long size() const noexcept
00334 { return _ds_size; }
00335 unsigned long map_flags() const noexcept
00336 { return _map_flags; }
00337 unsigned long page_size() const noexcept
00338 { return 1UL < page_shift(); }
00339 unsigned long round_size() const noexcept
00340 { return l4_round_size(size(), page_shift()); }
00341 bool check_limit(l4_addr_t offset) const noexcept
00342 { return offset < round_size(); }
00343
00344 L4Re::Dataspace::Flags
00345 map_flags(L4Re::Dataspace::Rights rights = L4_CAP_FPAGE_W) const noexcept
00346 {
00347 auto f = (_rw_flags & L4Re::Dataspace::Flags(0x0f)) | L4Re::Dataspace::F::Caching_mask;
00348 if (!(rights & L4_CAP_FPAGE_W))
00349 f &= ~L4Re::Dataspace::F::W;
00350
00351 return f;
00352 }
00353
00354 protected:
00355 void size(unsigned long size) noexcept { _ds_size = size; }
00356
00357 l4_addr_t _ds_start;
00358 l4_size_t _ds_size;
00359 Map_type _map_flags;
00360 Cache_type _cache_flags;
00361 L4Re::Dataspace::Flags _rw_flags;
00362 };
00363
00364 }}

```

## 17.394 l4/re/debug File Reference

Debug interface.

```

#include <l4/sys/capability>
#include <l4/re/protocols.h>
#include <l4/sys/cxx/ipc_iface>

```





## 17.395 debug

[Go to the documentation of this file.](#)

```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #pragma once
00014
00015 #include <l4/sys/capability>
00016 #include <l4/re/protocols.h>
00017 #include <l4/sys/cxx/ipc_iface>
00018
00019 namespace L4Re {
00033
00040 class L4_EXPORT Debug_obj :
00041 public L4::Kobject_t<Debug_obj, L4::Kobject, L4RE_PROTO_DEBUG>
00042 {
00043 public:
00044
00056 L4_INLINE_RPC(long, debug, (unsigned long function));
00057 typedef L4::Typeid::Rpc_nocode<debug_t> Rpcs;
00058 };
00059
00060 template<typename BASE>
00061 class Debug_obj_t :
00062 public L4::Kobject_2t<Debug_obj_t<BASE>, BASE, Debug_obj, L4::PROTO_EMPTY>
00063 {
00064 typedef L4::Typeid::Rpcs<> Rpcs;
00065 };
00066 }
```

## 17.396 debug

```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00007 /*
00008 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00009 * Alexander Warg <warg@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #pragma once
00015
00016 #include <l4/sys/types.h>
00017
00018 namespace L4Re { namespace Util {
00019 class Err
00020 {
00021 public:
00022 enum Level
00023 {
00024 Normal = 0,
00025 Fatal,
00026 };
00027
00028 static char const *const levels[];
00029
00030 void tag() const
00031 { printf("%s: %s", _component, levels[_l]); }
00032
00033 int printf(char const *fmt, ...) const
00034 __attribute__((format(printf,2,3)));
00035
00036 int cprintf(char const *fmt, ...) const
00037 __attribute__((format(printf,2,3)));
00038
00039 constexpr Err(Level l, char const *component) : _l(l), _component(component)
00040 {}
00041
00042 private:
00043 Level _l;
00044 char const *_component;
00045 };
00046
00047 }
```

```

00048 class Dbg
00049 {
00050 private:
00051 void tag() const;
00052
00053 #ifndef NDEBUG
00054
00055 unsigned long _m;
00056 char const *const _component;
00057 char const *const _subsys;
00058
00059 # ifnfe __clang__
00060
00061 int printf_impl(char const *fmt, ...) const
00062 __attribute__((format(printf, 2, 3)));
00063
00064 int cprintf_impl(char const *fmt, ...) const
00065 __attribute__((format(printf, 2, 3)));
00066
00067 # endif
00068
00069 public:
00070 static unsigned long level;
00071
00072 static void set_level(unsigned long l) { level = l; }
00073
00074 bool is_active() const { return _m & level; }
00075
00076 # ifdef __clang__
00077
00078 int printf(char const *fmt, ...) const
00079 __attribute__((format(printf, 2, 3)));
00080
00081 int cprintf(char const *fmt, ...) const
00082 __attribute__((format(printf, 2, 3)));
00083
00084 # else
00085
00086 int __attribute__((always_inline, format(printf, 2, 3)))
00087 printf(char const *fmt, ...) const
00088 {
00089 if (!(level & _m))
00090 return 0;
00091
00092 return printf_impl(fmt, __builtin_va_arg_pack());
00093 }
00094
00095 int __attribute__((always_inline, format(printf, 2, 3)))
00096 cprintf(char const *fmt, ...) const
00097 {
00098 if (!(level & _m))
00099 return 0;
00100
00101 return cprintf_impl(fmt, __builtin_va_arg_pack());
00102 }
00103
00104 # endif
00105
00106 explicit constexpr
00107 Dbg() : _m(1), _component(0), _subsys(0) {}
00108
00109 explicit constexpr
00110 Dbg(unsigned long mask, char const *comp, char const *subs)
00111 : _m(mask), _component(comp), _subsys(subs)
00112 {}
00113
00114 #else
00115
00116 public:
00117 static void set_level(unsigned long) {}
00118 bool is_active() const { return false; }
00119
00120 int printf(char const * /*fmt*/, ...) const
00121 __attribute__((format(printf, 2, 3)))
00122 { return 0; }
00123
00124 int cprintf(char const * /*fmt*/, ...) const
00125 __attribute__((format(printf, 2, 3)))
00126 { return 0; }
00127
00128 explicit constexpr
00129 Dbg() {}
00130
00131 explicit constexpr
00132 Dbg(unsigned long, char const *, char const *) {}
00133
00134 #endif

```

```

00135
00136 };
00137
00138 }}
00139

```

## 17.397 env\_ns

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002
00003 #pragma once
00004
00005 #include <l4/re/cap_alloc>
00006 #include <l4/re/util/cap_alloc>
00007 #include <l4/re/namespace>
00008 #include <l4/re/env>
00009 #include <string.h>
00010
00011 namespace L4Re { namespace Util {
00012
00013 class Env_ns
00014 {
00015 private:
00016 L4Re::Cap_alloc *_ca;
00017 Env const *_env;
00018
00019 public:
00020 explicit Env_ns(Env const *env = Env::env(),
00021 L4Re::Cap_alloc *ca = &L4Re::Util::cap_alloc)
00022 : _ca(ca), _env(env) {}
00023
00024 L4::Cap<void>
00025 query(char const *name, unsigned len, int timeout = Namespace::To_default,
00026 l4_umword_t *local_id = 0, bool iterate = true) const noexcept
00027 {
00028 typedef Env::Cap_entry Cap_entry;
00029
00030 // Skip possible first slash
00031 if (len && name[0] == '/')
00032 {
00033 ++name;
00034 --len;
00035 }
00036
00037 char const *n = name;
00038 for (; len && *n != '/'; ++n, --len) // Count first path element
00039 ;
00040
00041 Cap_entry const *e = _env->get(name, n - name);
00042 if (!e)
00043 return L4::Cap<void>(-L4_ENOENT);
00044
00045 if (len > 0 && *n == '/')
00046 {
00047 L4::Cap<L4Re::Namespace> ns(e->cap);
00048 L4::Cap<void> cap = _ca->alloc<void>();
00049
00050 if (!cap.is_valid())
00051 return L4::Cap<void>(-L4_ENOMEM);
00052
00053 long r = ns->query(n + 1, len - 1, cap, timeout, local_id, iterate);
00054 if (r >= 0)
00055 return cap;
00056
00057 _ca->free(cap);
00058
00059 return L4::Cap<void>(r);
00060 }
00061
00062 return L4::Cap<void>(e->cap);
00063 }
00064
00065 L4::Cap<void>
00066 query(char const *name, int timeout = Namespace::To_default,
00067 l4_umword_t *local_id = 0, bool iterate = true) const noexcept
00068 { return query(name, __builtin_strlen(name), timeout, local_id, iterate); }
00069
00070 template<typename T >
00071 L4::Cap<T>
00072 query(char const *name, int timeout = Namespace::To_default,
00073 l4_umword_t *local_id = 0, bool iterate = true) const noexcept
00074 {
00075 return L4::cap_cast<T>(query(name, __builtin_strlen(name),

```

```

00076 timeout, local_id, iterate));
00077 }
00078 };
00079
00080 }}

```

## 17.398 event

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009
00010 #pragma once
00011
00012 #include <l4/sys/capability>
00013 #include <l4/sys/irq>
00014 #include <l4/sys/cxx/ipc_iface>
00015 #include <l4/sys/cxx/ipc_array>
00016 #include <l4/re/dataspace>
00017 #include <l4/re/event.h>
00018
00019 namespace L4Re {
00020
00021
00022
00023
00024 typedef l4re_event_stream_id_t Event_stream_id;
00025 typedef l4re_event_absinfo_t Event_absinfo;
00026
00027 class L4_EXPORT Event_stream_bitmap_h
00028 {
00029 protected:
00030 static unsigned __get_idx(unsigned idx)
00031 { return idx / (sizeof(unsigned long)*8); }
00032
00033 static unsigned long __get_mask(unsigned idx)
00034 { return 1ul << (idx % (sizeof(unsigned long)*8)); }
00035
00036 static bool __get_bit(unsigned long const *bm, unsigned max, unsigned idx)
00037 {
00038 if (idx <= max)
00039 return bm[__get_idx(idx)] & __get_mask(idx);
00040 return false;
00041 }
00042
00043 static void __set_bit(unsigned long *bm, unsigned max, unsigned idx, bool v)
00044 {
00045 if (idx > max)
00046 return;
00047
00048 if (v)
00049 bm[__get_idx(idx)] |= __get_mask(idx);
00050 else
00051 bm[__get_idx(idx)] &= ~__get_mask(idx);
00052 }
00053 };
00054
00055 class L4_EXPORT Event_stream_info
00056 : public l4re_event_stream_info_t,
00057 private Event_stream_bitmap_h
00058 {
00059 public:
00060 bool get_propbit(unsigned idx) const
00061 { return __get_bit(propbits, L4RE_EVENT_PROP_MAX, idx); }
00062
00063 void set_propbit(unsigned idx, bool v)
00064 { __set_bit(propbits, L4RE_EVENT_PROP_MAX, idx, v); }
00065
00066 bool get_evbit(unsigned idx) const
00067 { return __get_bit(evbits, L4RE_EVENT_EV_MAX, idx); }
00068
00069 void set_evbit(unsigned idx, bool v)
00070 { __set_bit(evbits, L4RE_EVENT_EV_MAX, idx, v); }
00071
00072 bool get_keybit(unsigned idx) const
00073 { return __get_bit(keybits, L4RE_EVENT_KEY_MAX, idx); }
00074
00075 void set_keybit(unsigned idx, bool v)
00076 { __set_bit(keybits, L4RE_EVENT_KEY_MAX, idx, v); }
00077
00078
00079
00080
00081
00082
00083
00084
00085
00086
00087
00088
00089
00090
00091
00092
00093

```

```

00094 bool get_relbit(unsigned idx) const
00095 { return __get_bit(relbits, L4RE_EVENT_REL_MAX, idx); }
00096
00097 void set_relbit(unsigned idx, bool v)
00098 { __set_bit(relbits, L4RE_EVENT_REL_MAX, idx, v); }
00099
00100 bool get_absbit(unsigned idx) const
00101 { return __get_bit(absbits, L4RE_EVENT_ABS_MAX, idx); }
00102
00103 void set_absbit(unsigned idx, bool v)
00104 { __set_bit(absbits, L4RE_EVENT_ABS_MAX, idx, v); }
00105
00106 bool get_swbit(unsigned idx) const
00107 { return __get_bit(swbits, L4RE_EVENT_SW_MAX, idx); }
00108
00109 void set_swbit(unsigned idx, bool v)
00110 { __set_bit(swbits, L4RE_EVENT_SW_MAX, idx, v); }
00111 };
00112
00113 class L4_EXPORT Event_stream_state
00114 : public l4re_event_stream_state_t,
00115 private Event_stream_bitmap_h
00116 {
00117 public:
00118 bool get_keybit(unsigned idx) const
00119 { return __get_bit(keybits, L4RE_EVENT_KEY_MAX, idx); }
00120
00121 void set_keybit(unsigned idx, bool v)
00122 { __set_bit(keybits, L4RE_EVENT_KEY_MAX, idx, v); }
00123
00124 bool get_swbit(unsigned idx) const
00125 { return __get_bit(swbits, L4RE_EVENT_SW_MAX, idx); }
00126
00127 void set_swbit(unsigned idx, bool v)
00128 { __set_bit(swbits, L4RE_EVENT_SW_MAX, idx, v); }
00129 };
00130
00131 class L4_EXPORT Event :
00132 public L4::Kobject_t<Event, L4::Icu, L4RE_PROTO_EVENT>
00133 {
00134 public:
00135 L4_RPC(long, get_buffer, (L4::Ipc::Out<L4::Cap<Dataspace> > ds));
00136
00137 L4_RPC(long, get_num_streams, ());
00138
00139 L4_RPC(long, get_stream_info, (int idx, Event_stream_info *info));
00140
00141 L4_RPC(long, get_stream_info_for_id, (l4_umword_t stream_id, Event_stream_info *info));
00142
00143 L4_RPC_NF(long, get_axis_info, (l4_umword_t stream_id,
00144 L4::Ipc::Array<unsigned const, unsigned long> axes,
00145 L4::Ipc::Array<Event_absinfo, unsigned long> &info));
00146
00147 long get_axis_info(l4_umword_t stream_id, unsigned naxes,
00148 unsigned const *axis, Event_absinfo *info) const noexcept
00149 {
00150 L4::Ipc::Array<Event_absinfo, unsigned long> i(naxes, info);
00151 return get_axis_info_t::call(c(), stream_id,
00152 L4::Ipc::Array<unsigned const, unsigned long>(naxes, axis), i);
00153 }
00154
00155 L4_RPC(long, get_stream_state_for_id, (l4_umword_t stream_id,
00156 Event_stream_state *state));
00157
00158 typedef L4::TypeId::Rpc<
00159 get_buffer_t,
00160 get_num_streams_t,
00161 get_stream_info_t,
00162 get_stream_info_for_id_t,
00163 get_axis_info_t,
00164 get_stream_state_for_id_t
00165 > Rpc<
00166 >;
00167
00168 struct L4_EXPORT Default_event_payload
00169 {
00170 unsigned short type;
00171 unsigned short code;
00172 int value;
00173 l4_umword_t stream_id;
00174 };
00175
00176 template< typename PAYLOAD = Default_event_payload >
00177 class L4_EXPORT Event_buffer_t
00178 {
00179 public:

```

```

00249
00253 struct Event
00254 {
00255 long long time;
00256 PAYLOAD payload;
00257
00261 void free() noexcept { l4_mb(); time = 0; }
00262 };
00263
00264 private:
00265 Event *_current;
00266 Event *_begin;
00267 Event const *_end;
00268
00269 void inc() noexcept
00270 {
00271 ++_current;
00272 if (_current == _end)
00273 _current = _begin;
00274 }
00275
00276 public:
00277 Event_buffer_t() : _current(0), _begin(0), _end(0) {}
00278
00280 void reset()
00281 {
00282 for (Event *i = _begin; i != _end; ++i)
00283 i->time = 0;
00284 _current = _begin;
00285 }
00286
00293 Event_buffer_t(void *buffer, l4_addr_t size)
00294 : _current(static_cast<Event*>(buffer)), _begin(_current),
00295 _end(_begin + size / sizeof(Event))
00296 { reset(); }
00297
00303 Event *next() noexcept
00304 {
00305 Event *c = _current;
00306 if (c->time)
00307 {
00308 inc();
00309 return c;
00310 }
00311 return 0;
00312 }
00313
00320 bool put(Event const &ev) noexcept
00321 {
00322 Event *c = _current;
00323 if (c->time)
00324 return false;
00325
00326 inc();
00327 c->payload = ev.payload;
00328 l4_wmb();
00329 c->time = ev.time;
00330 return true;
00331 }
00332 };
00333
00334 typedef Event_buffer_t<Default_event_payload> Event_buffer;
00335
00336 }

```

## 17.399 l4/re/util/event File Reference

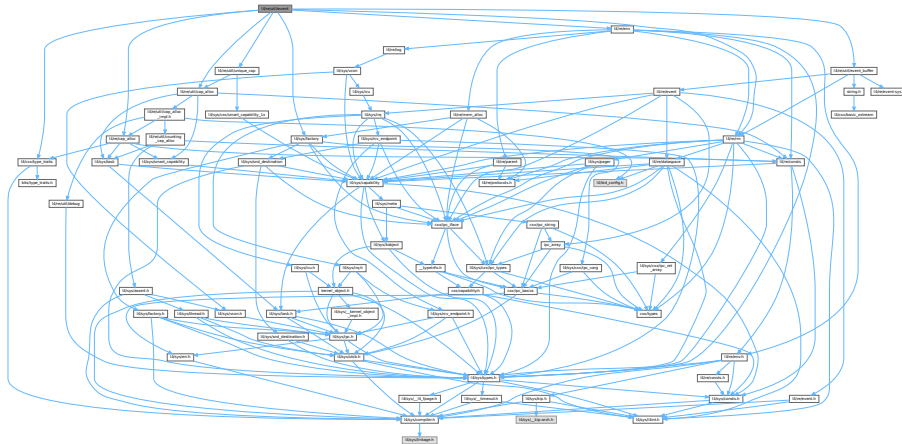
```

#include <l4/re/cap_alloc>
#include <l4/re/util/cap_alloc>
#include <l4/re/util/unique_cap>
#include <l4/re/env>
#include <l4/re/rm>
#include <l4/re/util/event_buffer>
#include <l4/sys/factory>

```

```
#include <l4/cxx/type_traits>
```

Include dependency graph for event:



## Data Structures

- class `L4Re::Util::Event_t< PAYLOAD >`  
*Convenience wrapper for getting access to an event object.*

## Namespaces

- namespace [L4Re](#)  
*[L4Re C++ Interfaces](#).*
- namespace [L4Re::Util](#)  
*Documentation of the [L4 Runtime Environment](#) utility functionality in C++.*

**17.400 event**

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00014 #include <l4/re/cap_alloc>
00015 #include <l4/re/util/cap_alloc>
00016 #include <l4/re/util/unique_cap>
00017 #include <l4/re/env>
00018 #include <l4/re/rm>
00019 #include <l4/re/util/event_buffer>
00020 #include <l4/sys/factory>
00021 #include <l4/cxx/type_traits>
00022
00023 namespace L4Re { namespace Util {
00024
00031 template< typename PAYLOAD >
00032 class Event_t
00033 {
00034 public:
00038 enum Mode

```

```

00039 {
00040 Mode_irq,
00041 Mode_polling,
00042 };
00043
00058 template<typename IRQ_TYPE>
00059 int init(L4::Cap<L4Re::Event> event,
00060 L4Re::Env const *env = L4Re::Env::env(),
00061 L4Re::Cap_alloc *ca = &L4Re::Util::cap_alloc)
00062 {
00063 Unique_cap<L4Re::Dataspace> ev_ds(ca->alloc<L4Re::Dataspace>());
00064 if (!ev_ds.is_valid())
00065 return -L4_ENOMEM;
00066
00067 int r;
00068
00069 Unique_del_cap<IRQ_TYPE> ev_irq(ca->alloc<IRQ_TYPE>());
00070 if (!ev_irq.is_valid())
00071 return -L4_ENOMEM;
00072
00073 if ((r = l4_error(env->factory()->create(ev_irq.get()))))
00074 return r;
00075
00076 if ((r = l4_error(event->bind(0, ev_irq.get()))))
00077 return r;
00078
00079 if ((r = event->get_buffer(ev_ds.get())))
00080 return r;
00081
00082 long sz = ev_ds->size();
00083 if (sz < 0)
00084 return sz;
00085
00086 Rm::Unique_region<void*> buf;
00087
00088 if ((r = env->rm()->attach(&buf, sz,
00089 L4Re::Rm::F::Search_addr | L4Re::Rm::F::RW,
00090 L4::Ipc::make_cap_rw(ev_ds.get()))))
00091 return r;
00092
00093 _ev_buffer = L4Re::Event_buffer_t<PAYLOAD>(buf.get(), sz);
00094 _ev_ds = cxx::move(ev_ds);
00095 _ev_irq = cxx::move(ev_irq);
00096 _buf = cxx::move(buf);
00097
00098 return 0;
00099 }
00100
00112 int init_poll(L4::Cap<L4Re::Event> event,
00113 L4Re::Env const *env = L4Re::Env::env(),
00114 L4Re::Cap_alloc *ca = &L4Re::Util::cap_alloc)
00115 {
00116 Unique_cap<L4Re::Dataspace> ev_ds(ca->alloc<L4Re::Dataspace>());
00117 if (!ev_ds.is_valid())
00118 return -L4_ENOMEM;
00119
00120 int r;
00121
00122 if ((r = event->get_buffer(ev_ds.get())))
00123 return r;
00124
00125 long sz = ev_ds->size();
00126 if (sz < 0)
00127 return sz;
00128
00129 Rm::Unique_region<void*> buf;
00130
00131 if ((r = env->rm()->attach(&buf, sz,
00132 L4Re::Rm::F::Search_addr | L4Re::Rm::F::RW,
00133 L4::Ipc::make_cap_rw(ev_ds.get()))))
00134 return r;
00135
00136 _ev_buffer = L4Re::Event_buffer_t<PAYLOAD>(buf.get(), sz);
00137 _ev_ds = cxx::move(ev_ds);
00138 _buf = cxx::move(buf);
00139
00140 return 0;
00141 }
00142
00148 L4Re::Event_buffer_t<PAYLOAD> &buffer() { return _ev_buffer; }
00149
00155 L4::Cap<L4::Triggerable> irq() const { return _ev_irq.get(); }
00156
00157 private:
00158 Unique_cap<L4Re::Dataspace> _ev_ds;
00159 Unique_del_cap<L4::Triggerable> _ev_irq;
00160 L4Re::Event_buffer_t<PAYLOAD> _ev_buffer;

```



```

00161 Rm::Unique_region<void*> _buf;
00162 };
00163
00164 typedef Event_t<Default_event_payload> Event;
00165
00166 }}

```

## 17.401 event\_buffer

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009
00010 #pragma once
00011
00012 #include <l4/re/event>
00013 #include <l4/re/event-sys.h>
00014 #include <l4/re/rm>
00015
00016 #include <string.h>
00017
00018 namespace L4Re { namespace Util {
00019
00024 template< typename PAYLOAD >
00025 class Event_buffer_t : public L4Re::Event_buffer_t<PAYLOAD>
00026 {
00027 private:
00028 void *_buf;
00029 public:
00035 void *_buf() const noexcept { return _buf; }
00036
00045 long attach(L4::Cap<L4Re::Dataspace> ds, L4::Cap<L4Re::Rm> rm) noexcept
00046 {
00047 l4_addr_t sz = ds->size();
00048 _buf = 0;
00049
00050 long r = rm->attach(&_buf, sz,
00051 L4Re::Rm::F::Search_addr | L4Re::Rm::F::RW,
00052 L4::Ipc::make_cap_rw(ds));
00053 if (r < 0)
00054 return r;
00055
00056 *static_cast<L4Re::Event_buffer_t<PAYLOAD>*>(this)
00057 = L4Re::Event_buffer_t<PAYLOAD>(_buf, sz);
00058 return 0;
00059 }
00060
00068 long detach(L4::Cap<L4Re::Rm> rm) noexcept
00069 {
00070 L4::Cap<L4Re::Dataspace> ds;
00071 if (_buf)
00072 return rm->detach(_buf, &ds);
00073 return 0;
00074 }
00075 };
00076
00077 template< typename PAYLOAD >
00083 class Event_buffer_consumer_t : public Event_buffer_t<PAYLOAD>
00084 {
00085 public:
00086
00093 template< typename CB, typename D >
00094 void foreach_available_event(CB const &cb, D data = D())
00095 {
00096 typename Event_buffer_t<PAYLOAD>::Event *e;
00097 while ((e = Event_buffer_t<PAYLOAD>::next()))
00098 {
00099 cb(e, data);
00100 e->free();
00101 }
00102 }
00103
00114 template< typename CB, typename D >
00115 void process(L4::Cap<L4::Irq> irq,
00116 L4::Cap<L4::Thread> thread,
00117 CB const &cb, D data = D())
00118 {

```

```

00119
00120 if (l4_error(irq->bind_thread(thread, 0)))
00121 return;
00122
00123 while (1)
00124 {
00125 long r;
00126 r = l4_ipc_error(l4_irq_receive(irq.cap(), L4_IPC_NEVER),
00127 l4_utcb());
00128 if (r)
00129 continue;
00130
00131 foreach_available_event(cb, data);
00132 }
00133 }
00134 };
00135
00136 typedef Event_buffer_t<Default_event_payload> Event_buffer;
00137 typedef Event_buffer_consumer_t<Default_event_payload> Event_buffer_consumer;
00138
00139 }}

```

## 17.402 event\_svr

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009
00010 #pragma once
00011
00012 #include <l4/re/event_enums.h>
00013 #include <l4/re/event>
00014 #include <l4/re/event-sys.h>
00015 #include <l4/re/util/icu_svr>
00016 #include <l4/cxx/minmax>
00017
00018 #include <l4/sys/cxx/ipc_legacy>
00019
00020 namespace L4Re { namespace Util {
00021
00022 template< typename SVR >
00023 class Event_svr : public Icu_cap_array_svr<SVR>
00024 {
00025 private:
00026 typedef Icu_cap_array_svr<SVR> Icu_svr;
00027
00028 protected:
00029 L4::Cap<L4Re::Dataspace> _ds;
00030 typename Icu_svr::Irq _irq;
00031
00032 public:
00033 Event_svr() : Icu_svr(1, &_irq) {}
00034
00035 L4_RPC_LEGACY_DISPATCH(L4Re::Event);
00036 L4_RPC_LEGACY_USING(Icu_svr);
00037
00038 long op_get_buffer(L4Re::Event::Rights, L4::Ipc::Cap<L4Re::Dataspace> &ds)
00039 {
00040 static_cast<SVR*>(this)->reset_event_buffer();
00041 ds = L4::Ipc::Cap<L4Re::Dataspace>(_ds, L4_CAP_FPAGE_RW);
00042 return 0;
00043 }
00044
00045 long op_get_num_streams(L4Re::Event::Rights)
00046 { return static_cast<SVR*>(this)->get_num_streams(); }
00047
00048 long op_get_stream_info(L4Re::Event::Rights, int idx, Event_stream_info &info)
00049 { return static_cast<SVR*>(this)->get_stream_info(idx, &info); }
00050
00051 long op_get_stream_info_for_id(L4Re::Event::Rights, l4_umword_t id,
00052 Event_stream_info &info)
00053 { return static_cast<SVR*>(this)->get_stream_info_for_id(id, &info); }
00054
00055 long op_get_axis_info(L4Re::Event::Rights, l4_umword_t id,
00056 L4::Ipc::Array_in_buf<unsigned, unsigned long> const &axes,
00057 L4::Ipc::Array_ref<Event_absinfo, unsigned long> &info)
00058 {
00059 unsigned naxes = cxx::min<unsigned>(L4RE_ABS_MAX, axes.length);

```

```

00066
00067 info.length = 0;
00068
00069 Event_absinfo _info[L4RE_ABS_MAX];
00070 int r = static_cast<SVR*>(this)->get_axis_info(id, naxes, axes.data, _info);
00071 if (r < 0)
00072 return r;
00073
00074 for (unsigned i = 0; i < naxes; ++i)
00075 info.data[i] = _info[i];
00076
00077 info.length = naxes;
00078 return r;
00079 }
00080
00081 long op_get_stream_state_for_id(L4Re::Event::Rights, l4_umword_t stream_id,
00082 Event_stream_state &state)
00083 { return static_cast<SVR*>(this)->get_stream_state_for_id(stream_id, &state); }
00084
00085 int get_num_streams() const { return 0; }
00086 int get_stream_info(int, L4Re::Event_stream_info *)
00087 { return -L4_EINVAL; }
00088 int get_stream_info_for_id(l4_umword_t, L4Re::Event_stream_info *)
00089 { return -L4_EINVAL; }
00090 int get_axis_info(l4_umword_t, unsigned /*naxes*/, unsigned const * /*axes*/,
00091 L4Re::Event_absinfo *)
00092 { return -L4_EINVAL; }
00093 int get_stream_state_for_id(l4_umword_t, L4Re::Event_stream_state *)
00094 { return -L4_EINVAL; }
00095 };
00096
00097 }

```

## 17.403 icu\_svr

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2009-2014 Alexander Warg <alexander.warg@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #pragma once
00009
00010
00011 #include <l4/sys/types.h>
00012
00013 #include <l4/sys/icu>
00014 #include <l4/sys/task>
00015 #include <l4/re/env>
00016 #include <l4/re/util/cap_alloc>
00017 #include <l4/sys/cxx/ipc_legacy>
00018
00019 namespace L4Re { namespace Util {
00020
00021 template<typename ICU>
00022 class Icu_svr
00023 {
00024 private:
00025 ICU const *this_icu() const { return static_cast<ICU const *>(this); }
00026 ICU *this_icu() { return static_cast<ICU*>(this); }
00027
00028 public:
00029 L4_RPC_LEGACY_DISPATCH(L4::Icu);
00030
00031 int op_bind(L4::Icu::Rights, l4_umword_t irqnum,
00032 L4::Ipc::Snd_fpage irq_fp);
00033 int op_unbind(L4::Icu::Rights, l4_umword_t irqnum,
00034 L4::Ipc::Snd_fpage irq_fp);
00035 int op_info(L4::Icu::Rights, L4::Icu::_Info &info);
00036 int op_msi_info(L4::Icu::Rights, l4_umword_t irqnum,
00037 l4_uint64_t source, l4_icu_msi_info_t &info);
00038 int op_mask(L4::Icu::Rights, l4_umword_t irqnum);
00039 int op_unmask(L4::Icu::Rights, l4_umword_t irqnum);
00040 int op_set_mode(L4::Icu::Rights, l4_umword_t, l4_umword_t)
00041 { return 0; }
00042 };
00043
00044 template<typename ICU> inline
00045 int
00046 Icu_svr<ICU>::op_bind(L4::Icu::Rights, l4_umword_t irqnum,
00047 L4::Ipc::Snd_fpage irq_fp)
00048 {

```

```

00049 typename ICU::Irq *irq = this_icu()->icu_get_irq(irqnum);
00050 if (!irq)
00051 return -L4_EINVAL;
00052
00053 return irq->bind(this_icu(), irq_fp);
00054 }
00055
00056 template<typename ICU> inline
00057 int
00058 Icu_svr<ICU>::op_unbind(L4::Icu::Rights, l4_umword_t irqnum,
00059 L4::Ipc::Snd_fpage irq_fp)
00060 {
00061 typename ICU::Irq *irq = this_icu()->icu_get_irq(irqnum);
00062 if (!irq)
00063 return -L4_EINVAL;
00064
00065 return irq->unbind(this_icu(), irq_fp);
00066 }
00067
00068 template<typename ICU> inline
00069 int
00070 Icu_svr<ICU>::op_info(L4::Icu::Rights, L4::Icu::_Info &info)
00071 {
00072 l4_icu_info_t i;
00073 this_icu()->icu_get_info(&i);
00074 info.features = i.features;
00075 info.nr_irqs = i.nr_irqs;
00076 info.nr_msis = i.nr_msis;
00077 return 0;
00078 }
00079
00080 template<typename ICU> inline
00081 int
00082 Icu_svr<ICU>::op_msi_info(L4::Icu::Rights, l4_umword_t irqnum,
00083 l4_uint64_t source, l4_icu_msi_info_t &info)
00084 {
00085 typename ICU::Irq *irq = this_icu()->icu_get_irq(irqnum);
00086 if (!irq)
00087 return -L4_EINVAL;
00088 return irq->msi_info(source, &info);
00089 }
00090
00091 template<typename ICU> inline
00092 int
00093 Icu_svr<ICU>::op_mask(L4::Icu::Rights, l4_umword_t irqnum)
00094 {
00095 typename ICU::Irq *irq = this_icu()->icu_get_irq(irqnum);
00096 if (irq)
00097 irq->mask(true);
00098 return -L4_ENOREPLY;
00099 }
00100
00101 template<typename ICU> inline
00102 int
00103 Icu_svr<ICU>::op_unmask(L4::Icu::Rights, l4_umword_t irqnum)
00104 {
00105 typename ICU::Irq *irq = this_icu()->icu_get_irq(irqnum);
00106 if (irq)
00107 irq->mask(false);
00108 return -L4_ENOREPLY;
00109 }
00110
00111
00112 template< typename ICU >
00113 class Icu_cap_array_svr : public Icu_svr<ICU>
00114 {
00115 protected:
00116 static void free_irq_cap(L4::Cap<L4::Irq> &cap)
00117 {
00118 if (cap)
00119 {
00120 L4Re::Util::cap_alloc.free(cap);
00121 cap.invalidate();
00122 }
00123 }
00124
00125 public:
00126 class Irq
00127 {
00128 public:
00129 Irq() {}
00130 ~Irq() { ICU::free_irq_cap(_cap); }
00131
00132 void trigger() const
00133 {
00134 if (_cap)
00135 _cap->trigger();

```

```

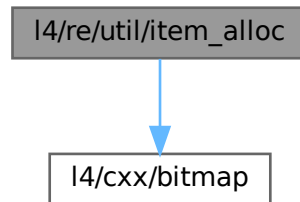
00136 }
00137
00138 int bind(ICU *, L4::Ipc::Snd_fpage const &irq_fp);
00139 int unbind(ICU *, L4::Ipc::Snd_fpage const &irq_fp);
00140 void mask(bool /*mask*/) const
00141 { }
00142
00143 int msi_info(l4_uint64_t, l4_icu_msi_info_t *) const
00144 { return -L4_EINVAL; }
00145
00146 L4::Cap<L4::Irq> cap() const { return _cap; }
00147
00148 private:
00149 L4::Cap<L4::Irq> _cap;
00150 };
00151
00152 private:
00153 Irq *_irqs;
00154 unsigned _nr_irqs;
00155
00156 public:
00157
00158 Icu_cap_array_svr(unsigned nr_irqs, Irq *irqs)
00159 : _irqs(irqs), _nr_irqs(nr_irqs)
00160 {}
00161
00162 Irq *icu_get_irq(l4_umword_t irqnum)
00163 {
00164 if (irqnum >= _nr_irqs)
00165 return 0;
00166
00167 return _irqs + irqnum;
00168 }
00169
00170 void icu_get_info(l4_icu_info_t *inf)
00171 {
00172 inf->features = 0;
00173 inf->nr_irqs = _nr_irqs;
00174 inf->nr_msis = 0;
00175 }
00176 };
00177
00178 template< typename ICU >
00179 int
00180 Icu_cap_array_svr<ICU>::Irq::bind(ICU *cfb, L4::Ipc::Snd_fpage const &irq_fp)
00181 {
00182 if (!irq_fp.cap_received())
00183 return -L4_EINVAL;
00184
00185 L4::Cap<L4::Irq> irq = cfb->server_iface()->template_rcv_cap<L4::Irq>(0);
00186 if (!irq)
00187 return -L4_EINVAL;
00188
00189 int r = cfb->server_iface()->realloc_rcv_cap(0);
00190 if (r < 0)
00191 return r;
00192
00193 ICU::free_irq_cap(_cap);
00194 _cap = irq;
00195 return 0;
00196 }
00197
00198 template< typename ICU >
00199 int
00200 Icu_cap_array_svr<ICU>::Irq::unbind(ICU *, L4::Ipc::Snd_fpage const &/*irq_fp*/)
00201 {
00202 ICU::free_irq_cap(_cap);
00203 _cap = L4::Cap<L4::Irq>::Invalid;
00204 return 0;
00205 }
00206
00207
00208 }

```

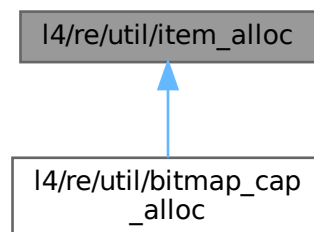
## 17.404 I4/re/util/item\_alloc File Reference

Item allocator.

```
#include <l4/cxx/bitmap>
Include dependency graph for item_alloc:
```



This graph shows which files directly or indirectly include this file:



## Data Structures

- class [L4Re::Util::Item\\_alloc\\_base](#)  
*Item allocator.*
- class [L4Re::Util::Bitmap\\_base](#)  
*Basic bitmap abstraction.*
- class [L4Re::Util::Bitmap< BITS >](#)  
*A static bitmap.*

## Namespaces

- namespace [L4Re](#)  
*L4Re C++ Interfaces.*
- namespace [L4Re::Util](#)  
*Documentation of the L4 Runtime Environment utility functionality in C++.*

## 17.404.1 Detailed Description

Item allocator.

Definition in file [item\\_alloc](#).

## 17.405 item\_alloc

[Go to the documentation of this file.](#)

```

00001 // -*- Mode: C++ -*-
00002 // vim:ft=cpp
00003 /*
00004 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00005 * Alexander Warg <warg@os.inf.tu-dresden.de>
00006 * economic rights: Technische Universität Dresden (Germany)
00007 *
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010
00011 #pragma once
00012
00013 #include <l4/cxx/bitmap>
00014
00015 namespace L4Re { namespace Util {
00016
00017 using cxx::Bitmap_base;
00018 using cxx::Bitmap;
00019
00020 class Item_alloc_base
00021 {
00022 private:
00023 long _capacity;
00024 long _free_hint;
00025 Bitmap_base _bits;
00026
00027 void hint(long hint)
00028 { __atomic_store_n(&_free_hint, hint, __ATOMIC_RELAXED); }
00029
00030 public:
00031 bool is_allocated(long item) const noexcept
00032 { return _bits[item]; }
00033
00034 long hint() const { return __atomic_load_n(&_free_hint, __ATOMIC_RELAXED); }
00035
00036 bool alloc(long item) noexcept
00037 {
00038 return !_bits.atomic_get_and_set(item);
00039 }
00040
00041 void free(long item) noexcept
00042 {
00043 if (item < hint())
00044 hint(item);
00045
00046 _bits.atomic_clear_bit(item);
00047 }
00048
00049 Item_alloc_base(long size, void *mem) noexcept
00050 : _capacity(size), _free_hint(0), _bits(mem)
00051 {}
00052
00053 long alloc() noexcept
00054 {
00055 long free_hint = hint();
00056
00057 for (long i = free_hint; i < _capacity; ++i)
00058 if (alloc(i))
00059 hint(i + 1);
00060 return i;
00061 }
00062
00063 // _free_hint is not necessarily correct in case of multi-threading! Make
00064 // sure we don't miss any potentially free slots.
00065 for (long i = 0; i < free_hint && i < _capacity; ++i)
00066 if (alloc(i))
00067 hint(i + 1);
00067 }
00068 }

```

```

00077 return i;
00078 }
00079
00080 return -1;
00081 }
00082
00083 long size() const noexcept
00084 {
00085 return _capacity;
00086 }
00087 };
00088
00089 template< long Bits >
00090 class Item_alloc : public Item_alloc_base
00091 {
00092 private:
00093 typename Bitmap_base::Word<Bits>::Type _bits[Bitmap_base::Word<Bits>::Size];
00094
00095 public:
00096 Item_alloc() noexcept : Item_alloc_base(Bits, _bits) {}
00097 };
00098
00099 }

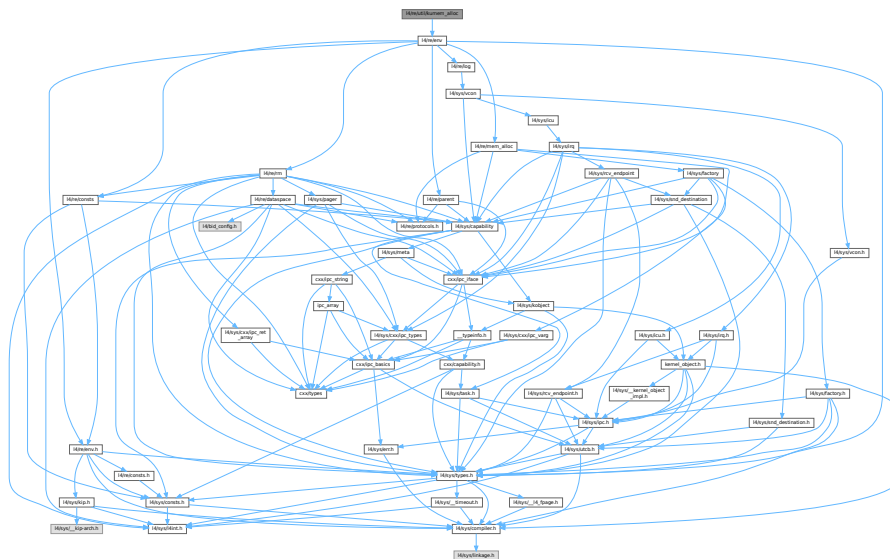
```

## 17.406 l4/re/util/kumem\_alloc File Reference

Kumem allocator helper.

```
#include <l4/re/env>
```

Include dependency graph for kumem\_alloc:



### Namespaces

- namespace [L4Re](#)  
[L4Re](#) C++ Interfaces.
- namespace [L4Re::Util](#)  
Documentation of the [L4](#) Runtime Environment utility functionality in C++.



## Functions

- `int L4Re::Util::kumem_alloc(l4_addr_t *mem, unsigned pages_order, L4::Cap< L4::Task > task=L4Re::Env::env() ->task(), L4::Cap< L4Re::Rm > rm=L4Re::Env::env() ->rm()) noexcept`

*Allocate state area.*

### 17.406.1 Detailed Description

Kumem allocator helper.

Definition in file [kumem\\_alloc](#).

## 17.407 kumem\_alloc

[Go to the documentation of this file.](#)

```
00001 // -*- Mode: C++ -*-
00002 // vim:ft=cpp
00003 /*
00004 * (c) 2010 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00005 * Alexander Warg <warg@os.inf.tu-dresden.de>
00006 * economic rights: Technische Universität Dresden (Germany)
00007 *
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010 #pragma once
00011 #include <l4/re/env>
00012 namespace L4Re { namespace Util {
00013 int
00014 kumem_alloc(l4_addr_t *mem, unsigned pages_order,
00015 L4::Cap<L4::Task> task = L4Re::Env::env()->task(),
00016 L4::Cap<L4Re::Rm> rm = L4Re::Env::env()->rm()) noexcept;
00017 }}
00018 }
```

## 17.408 name\_space\_svr

```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 #pragma once
00010 #include <l4/cxx/avl_tree>
00011 #include <l4/cxx/std_ops>
00012 #include <l4/sys/cxx/ipc_epiface>
00013 #include <l4/cxx/string>
00014 #include <l4/re/util/debug>
00015 #include <l4/sys/capability>
00016 #include <l4/re/namespace>
00017 #include <stddef.h>
00018 #include <string.h>
00019 namespace L4Re { namespace Util { namespace Names {
00020 class Name : public cxx::String
00021 {
00022 public:
```

```

00031
00032 Name(const char *name = "") : String(name, __builtin_strlen(name)) {}
00033 Name(const char *name, unsigned long len) : String(name, len) {}
00034 Name(cxx::String const &n) : String(n) {}
00035 char const *name() const { return start(); }
00036 bool operator < (Name const &r) const
00037 {
00038 unsigned long l = cxx::min(len(), r.len());
00039 int v = memcmp(start(), r.start(), l);
00040 return v < 0 || (v == 0 && len() < r.len());
00041 }
00042 };
00043
00044
00045 class Obj
00046 {
00047 protected:
00048 unsigned _f;
00049 union
00050 {
00051 l4_cap_idx_t _cap;
00052 L4::Epiface *_obj;
00053 };
00054
00055 public:
00056 enum Flags
00057 {
00058 F_rw = L4Re::Namespace::Rw,
00059 F_strong = L4Re::Namespace::Strong,
00060
00061 F_trusted = L4Re::Namespace::Trusted,
00062
00063 F_rights_mask = F_rw | F_strong | F_trusted,
00064
00065 F_cap = 0x100,
00066 F_local = 0x200,
00067 F_replacable = 0x400,
00068 F_base_mask = 0xf00,
00069 };
00070
00071 unsigned flags() const { return _f; }
00072 void restrict_flags(unsigned max_rights)
00073 { _f &= (~F_rights_mask | (max_rights & F_rights_mask)); }
00074
00075 bool is_rw() const { return (_f & F_rw) == F_rw; }
00076 bool is_strong() const { return _f & F_strong; }
00077
00078 bool is_valid() const { return _f & F_cap; }
00079 bool is_complete() const { return is_valid(); }
00080 bool is_local() const { return _f & F_local; }
00081 bool is_replacable() const { return _f & F_replacable; }
00082 bool is_trusted() const { return _f & F_trusted; }
00083
00084 L4::Epiface *obj() const { if (is_local()) return _obj; return 0; }
00085 L4::Cap<void> cap() const
00086 {
00087 if (!is_local())
00088 return L4::Cap<void>(_cap);
00089 if (!_obj)
00090 return L4::Cap<void>::Invalid;
00091 return _obj->obj_cap();
00092 }
00093
00094 void set(Obj const &o, unsigned flags)
00095 {
00096 *this = o;
00097 restrict_flags(flags);
00098 }
00099
00100 explicit Obj(unsigned flags = 0)
00101 : _f(flags), _cap(L4_INVALID_CAP)
00102 {}
00103
00104 Obj(unsigned f, L4::Cap<void> const &cap)
00105 : _f((f & ~F_base_mask) | F_cap), _cap(cap.cap())
00106 {}
00107
00108 Obj(unsigned f, L4::Epiface *o)
00109 : _f((f & ~F_base_mask) | F_cap | F_local), _obj(o)
00110 {}
00111
00112 void reset(unsigned flags)
00113 {
00114 _f = (_f & F_replacable) | (flags & ~(F_cap | F_local));
00115 }

```

```

00121 _cap = L4_INVALID_CAP;
00122 }
00123
00124
00125 };
00126
00127
00131 class Entry : public cxx::Avl_tree_node
00132 {
00133 private:
00134 friend class Name_space;
00135 Name _n;
00136 Obj _o;
00137
00138 bool _dynamic;
00139
00140 public:
00141 Entry(Name const &n, Obj const &o, bool dynamic = false)
00142 : _n(n), _o(o), _dynamic(dynamic) {}
00143
00144 Name const &name() const { return _n; }
00145 Obj const *obj() const { return &_amp;o; }
00146 Obj *obj() { return &_amp;o; }
00147 void obj(Obj const &o) { _o = o; }
00148
00149 bool is_placeholder() const
00150 { return !obj()->is_complete(); }
00151
00152 bool is_dynamic() const { return _dynamic; }
00153
00154 void set(Obj const &o)
00155 {
00156 obj()->set(o, obj()->flags());
00157 }
00158
00159 private:
00160 void * operator new (size_t s);
00161 void operator delete(void *b);
00162
00163 };
00164
00165 struct Names_get_key
00166 {
00167 typedef Name Key_type;
00168 static Key_type const &key_of(Entry const *e)
00169 { return e->name(); }
00170 };
00171
00172
00180 class Name_space
00181 {
00182 friend class Entry;
00183
00184 private:
00185 typedef cxx::Avl_tree<Entry, Names_get_key> Tree;
00186 Tree _tree;
00187
00188 protected:
00189 L4Re::Util::Dbg const &dbg;
00190 L4Re::Util::Err const &err;
00191
00192 public:
00193
00194 typedef Tree::Const_iterator Const_iterator;
00195
00196 Const_iterator begin() const { return _tree.begin(); }
00197 Const_iterator end() const { return _tree.end(); }
00198
00199 Name_space(L4Re::Util::Dbg const &dbg, L4Re::Util::Err const &err)
00200 : _dbg(dbg), _err(err)
00201 {}
00202
00206 virtual ~Name_space() {}
00207
00208 Entry *find(Name const &name) const { return _tree.find_node(name); }
00209 Entry *remove(Name const &name) { return _tree.remove(name); }
00210 Entry *find_iter(Name const &pname) const
00211 {
00212 Name name = pname;
00213 _dbg.printf("resolve '%.*s': ", name.len(), name.start());
00214 Name_space const *ns = this;
00215 while (ns)
00216 {
00217 cxx::String::Index sep = name.find("/");
00218 cxx::String part;
00219 if (!name.eof(sep))
00220 part = name.head(sep);

```

```

00221 else
00222 part = name;
00223
00224 _dbg.cprintf(" '%.*s'", part.len(), part.start());
00225 Entry *o = ns->find(Name(part.start(), part.len()));
00226
00227 if (!o)
00228 {
00229 _dbg.cprintf(": resolution failed: '%.*s' remaining\n",
00230 name.len(), name.start());
00231 return 0;
00232 }
00233
00234 auto const *obj = o->obj()->obj();
00235 ns = dynamic_cast<Name_space const *>(obj);
00236 if (ns)
00237 {
00238 if (!name.eof(sep))
00239 {
00240 name = name.substr(sep + 1);
00241 continue;
00242 }
00243 }
00244
00245 _dbg.cprintf(": found object: %p (%s)\n",
00246 obj, obj ? typeid(*obj).name() : "");
00247
00248 return o;
00249 }
00250
00251 return 0;
00252 }
00253
00254 bool insert(Entry *e) { return _tree.insert(e).second; }
00255
00256 void dump(bool rec = false, int indent = 0) const;
00257
00258 protected:
00259 // server support -----
00272 virtual Entry *alloc_dynamic_entry(Name const &n, unsigned flags) = 0;
00273
00279 virtual void free_dynamic_entry(Entry *e) = 0;
00280
00303 virtual int get_epiface(l4_umword_t data, bool is_local, L4::Epiface **lo) = 0;
00304
00317 virtual int copy_receive_cap(L4::Cap<void> *cap) = 0;
00318
00327 virtual void free_capability(L4::Cap<void> cap) = 0;
00328
00337 virtual void free_epiface(L4::Epiface *epiface) = 0;
00338
00339 int insert_entry(Name const &name, unsigned flags, Entry **e)
00340 {
00341 Entry *n = find(name);
00342 if (n && n->obj()->is_valid())
00343 {
00344 if (!(flags & L4Re::Namespace::Overwrite)
00345 && n->obj()->cap().validate(L4_BASE_TASK_CAP).label() > 0)
00346 return -L4_EEXIST;
00347
00348 if (n->obj()->is_local())
00349 free_epiface(n->obj()->obj());
00350 else
00351 free_capability(n->obj()->cap());
00352
00353 if (n->is_dynamic())
00354 {
00355 remove(n->name());
00356 free_dynamic_entry(n);
00357 n = 0;
00358 }
00359 else
00360 {
00361 if (!n->obj()->is_replacable())
00362 return -L4_EEXIST;
00363 n->obj()->reset(Obj::F_rw);
00364 }
00365 }
00366
00367 flags &= L4Re::Namespace::Cap_flags;
00368 if (!n)
00369 {
00370 if (!(n = alloc_dynamic_entry(name, flags)))
00371 return -L4_ENOMEM;
00372 else
00373 {
00374 if (!insert(n))

```

```

00375 {
00376 free_dynamic_entry(n);
00377 return -L4_ENOENT;
00378 }
00379 }
00380 }
00381
00382 *e = n;
00383 return 0;
00384 }
00385
00386 public:
00387 // server interface -----
00388 int op_query(L4Re::Namespace::Rights,
00389 L4::Ipc::Array_in_buf<char, unsigned long> const &name,
00390 L4::Ipc::Snd_fpage &snd_cap, L4::Ipc::Opt<L4::Opcode> &dummy,
00391 L4::Ipc::Opt<L4::Ipc::Array_ref<char, unsigned long> > &out_name)
00392 {
00393 #if 1
00394 _dbg.printf("query: [%ld] '%s'\n", name.length,
00395 static_cast<int>(name.length), name.data);
00396 #endif
00397
00398 char const *sep
00399 = static_cast<char const*>(memchr(name.data, '/', name.length));
00400 unsigned long part;
00401 if (sep)
00402 part = sep - name.data;
00403 else
00404 part = name.length;
00405
00406 Entry *n = find(Name(name.data, part));
00407 if (!n)
00408 return -L4_ENOENT;
00409 else if (!n->obj()->is_valid())
00410 return -L4_EAGAIN;
00411 else
00412 {
00413 if (n->obj()->cap().validate(L4_BASE_TASK_CAP).label() <= 0)
00414 {
00415 if (n->obj()->is_local())
00416 free_epiface(n->obj()->obj());
00417 else
00418 free_capability(n->obj()->cap());
00419
00420 if (n->is_dynamic())
00421 {
00422 remove(n->name());
00423 free_dynamic_entry(n);
00424 }
00425 return -L4_ENOENT;
00426 }
00427
00428 // make picky clients happy
00429 dummy.set_valid();
00430 // prevent warning about writing uninitialized data in IPC framework
00431 dummy = 0;
00432
00433 l4_umword_t result = 0;
00434
00435 out_name.set_valid();
00436 if (part < name.length)
00437 {
00438 result |= L4Re::Namespace::Partly_resolved;
00439 memcpy(out_name->data, name.data + part + 1, name.length - part - 1);
00440 out_name->length = name.length - part - 1;
00441 }
00442 else
00443 out_name->length = 0;
00444
00445 unsigned flags = L4_CAP_FPAGE_R;
00446 if (n->obj()->is_rw()) flags |= L4_CAP_FPAGE_W;
00447 if (n->obj()->is_strong()) flags |= L4_CAP_FPAGE_S;
00448
00449 snd_cap = L4::Ipc::Snd_fpage(n->obj()->cap(), flags);
00450 _dbg.printf(" result = %lx flgs=%x strg=%d\n",
00451 result, flags, static_cast<int>(n->obj()->is_strong()));
00452 return result;
00453 }
00454 }
00455
00456 int op_register_obj(L4Re::Namespace::Rights, unsigned flags,
00457 L4::Ipc::Array_in_buf<char, unsigned long> const &name,
00458 L4::Ipc::Snd_fpage &cap)
00459 {
00460 if (name.length == 0 || memchr(name.data, '/', name.length))
00461 return -L4_EINVAL;

```

```

00462
00463 L4::Cap<void> reg_cap(L4_INVALID_CAP);
00464 L4::Epiface *src_o = 0;
00465
00466 // Did we receive something we have handed out ourselves? If yes,
00467 // register the object under the given name but do not allocate
00468 // anything more.
00469 if (cap.id_received() || cap.local_id_received())
00470 {
00471 if (int ret = get_epiface(cap.data(), cap.local_id_received(), &src_o))
00472 return ret;
00473
00474 // Make sure rights are restricted to the mapped rights.
00475 flags &= (cap.data() & 0x3UL) | ~0x3UL;
00476 }
00477 else if (cap.cap_received())
00478 {
00479 if (int ret = copy_receive_cap(®_cap))
00480 return ret;
00481 }
00482 else if (!cap.is_valid())
00483 {
00484 reg_cap = L4::Cap<void>::Invalid;
00485 }
00486 else
00487 return -L4_EINVAL;
00488
00489 // got a valid entry to register
00490 _dbg.printf("register: '%.*s' flags=%x\n", static_cast<int>(name.length),
00491 name.data, flags);
00492
00493 Name _name(name.data, name.length);
00494
00495 Entry *n;
00496 if (int r = insert_entry(_name, flags, &n))
00497 {
00498 if (cap.cap_received())
00499 free_capability(reg_cap);
00500 if (src_o)
00501 free_epiface(src_o);
00502 return r;
00503 }
00504
00505 if (src_o)
00506 n->set(Names::Obj(flags & L4Re::Namespace::Cap_flags, src_o));
00507 else if (reg_cap.is_valid())
00508 n->set(Names::Obj(flags & L4Re::Namespace::Cap_flags, reg_cap));
00509
00510 return 0;
00511 }
00512
00513 int op_unlink(L4Re::Namespace::Rights,
00514 L4::Ipc::Array_in_buf<char, unsigned long> const &name)
00515 {
00516 #if 1
00517 _dbg.printf("unlink: [%ld] '%.*s'\n", name.length,
00518 static_cast<int>(name.length), name.data);
00519 #endif
00520
00521 char const *sep
00522 = static_cast<char const*>(memchr(name.data, '/', name.length));
00523 unsigned long part;
00524 if (sep)
00525 part = sep - name.data;
00526 else
00527 part = name.length;
00528
00529 Entry *n = find(Name(name.data, part));
00530 if (!n || !n->obj()->is_valid())
00531 return -L4_ENOENT;
00532
00533 if (n->obj()->is_local())
00534 free_epiface(n->obj()->obj());
00535 else
00536 free_capability(n->obj()->cap());
00537
00538 if (n->is_dynamic())
00539 {
00540 remove(n->name());
00541 free_dynamic_entry(n);
00542 }
00543 else
00544 return -L4_EACCESS;
00545
00546 return 0;

```

```

00549 }
00550 };
00551
00552 }}}
00553

```

## 17.409 object\_registry

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009
00010 #pragma once
00011
00012 #include <l4/re/util/cap_alloc>
00013 #include <l4/re/util/unique_cap>
00014 #include <l4/re/consts>
00015 #include <l4/re/env>
00016
00017 #include <l4/sys/cxx/ipc_server_loop>
00018 #include <l4/sys/factory>
00019 #include <l4/sys/task>
00020 #include <l4/sys/thread>
00021 #include <l4/sys/ipc_gate>
00022
00023 #include <l4/cxx/exceptions>
00024
00025 namespace L4Re { namespace Util {
00026
00041 class Object_registry :
00042 public L4::Basic_registry,
00043 public L4::Registry_iface
00044 {
00049 struct Null_handler : L4::Epiface_t<Null_handler, L4::Kobject>
00050 {};
00051
00052 protected:
00053 L4::Cap<L4::Thread> _server;
00054 L4::Cap<L4::Factory> _factory;
00055 L4::Ipc_svr::Server_iface *_sif;
00056
00057 private:
00058 Null_handler _null_handler;
00059
00060 public:
00066 explicit
00067 Object_registry(L4::Ipc_svr::Server_iface *sif)
00068 : _server(L4Re::Env::env()->main_thread()),
00069 _factory(L4Re::Env::env()->factory()),
00070 _sif(sif)
00071 {}
00072
00081 Object_registry(L4::Ipc_svr::Server_iface *sif,
00082 L4::Cap<L4::Thread> server,
00083 L4::Cap<L4::Factory> factory)
00084 : _server(server), _factory(factory), _sif(sif)
00085 {}
00086
00087 private:
00088 typedef L4::Ipc_svr::Server_iface Server_iface;
00089 typedef Server_iface::Demand Demand;
00090
00091 L4::Cap<L4::Rcv_endpoint>
00092 _register_ep(L4::Epiface *o, L4::Cap<L4::Rcv_endpoint> ep,
00093 Demand const &demand)
00094 {
00095 int err = _sif->alloc_buffer_demand(demand);
00096 if (err < 0)
00097 return L4::Cap<L4::Rcv_endpoint>(err | L4_INVALID_CAP_BIT);
00098
00099 err = o->set_server(_sif, ep);
00100 if (err < 0)
00101 return L4::Cap<L4::Rcv_endpoint>(err | L4_INVALID_CAP_BIT);
00102
00103 l4_umword_t id = l4_umword_t(o);
00104 err = l4_error(ep->bind_thread(_server, id));
00105 if (err < 0)
00106 return L4::Cap<L4::Rcv_endpoint>(err | L4_INVALID_CAP_BIT);

```

```

00107
00108 return ep;
00109 }
00110
00111 L4::Cap<void> _register_ep(L4::Epiface *o, char const *service,
00112 Demand const &demand)
00113 {
00114 L4::Cap<L4::Rcv_endpoint> cap = L4Re::Env::env()->get_cap<L4::Rcv_endpoint>(service);
00115 if (!cap.is_valid())
00116 return cap;
00117
00118 return _register_ep(o, cap, demand);
00119 }
00120
00121 L4::Cap<void> _register_gate(L4::Epiface *o, Demand const &demand)
00122 {
00123 int err = _sif->alloc_buffer_demand(demand);
00124 if (err < 0)
00125 return L4::Cap<void>(err | L4_INVALID_CAP_BIT);
00126
00127 auto cap = L4Re::Util::make_unique_cap<L4::Kobject>();
00128
00129 if (!cap.is_valid())
00130 return cap.get();
00131
00132 l4_umword_t id = l4_umword_t(o);
00133 err = l4_error(_factory->create_gate(cap.get(), _server, id));
00134 if (err < 0)
00135 return L4::Cap<void>(err | L4_INVALID_CAP_BIT);
00136
00137 err = o->set_server(_sif, cap.get(), true);
00138 if (err < 0)
00139 return L4::Cap<void>(err | L4_INVALID_CAP_BIT);
00140
00141 return cap.release();
00142 }
00143
00144 L4::Cap<L4::Irq> _register_irq(L4::Epiface *o,
00145 Demand const &demand)
00146 {
00147 int err = _sif->alloc_buffer_demand(demand);
00148 if (err < 0)
00149 return L4::Cap<L4::Irq>(err | L4_INVALID_CAP_BIT);
00150
00151 auto cap = L4Re::Util::make_unique_cap<L4::Irq>();
00152
00153 if (!cap.is_valid())
00154 return cap.get();
00155
00156 l4_umword_t id = l4_umword_t(o);
00157 err = l4_error(_factory->create(cap.get()));
00158 if (err < 0)
00159 return L4::Cap<L4::Irq>(err | L4_INVALID_CAP_BIT);
00160
00161 err = o->set_server(_sif, cap.get(), true);
00162 if (err < 0)
00163 return L4::Cap<L4::Irq>(err | L4_INVALID_CAP_BIT);
00164
00165 err = l4_error(cap->bind_thread(_server, id));
00166 if (err < 0)
00167 return L4::Cap<L4::Irq>(err | L4_INVALID_CAP_BIT);
00168
00169 return cap.release();
00170 }
00171
00172 static Demand _get_buffer_demand(L4::Epiface *o)
00173 { return o->get_buffer_demand(); }
00174
00175 template<typename T>
00176 static Demand _get_buffer_demand(T *,
00177 typename L4::Kobject_typeid<typename T::Interface>::Demand
00178 d = typename L4::Kobject_typeid<typename T::Interface>::Demand())
00179 { return d; }
00180
00181 public:
00194 L4::Cap<void> register_obj(L4::Epiface *o, char const *service) override
00195 {
00196 return _register_ep(o, service, _get_buffer_demand(o));
00197 }
00198
00211 L4::Cap<void> register_obj(L4::Epiface *o) override
00212 {
00213 return _register_gate(o, _get_buffer_demand(o));
00214 }
00215
00227 L4::Cap<L4::Irq> register_irq_obj(L4::Epiface *o) override
00228 {

```



```

00229 return _register_irq(o, _get_buffer_demand(o));
00230 }
00231
00245 L4::Cap<L4::Rcv_endpoint>
00246 register_obj(L4::Epiface *o, L4::Cap<L4::Rcv_endpoint> ep) override
00247 {
00248 return _register_ep(o, ep, _get_buffer_demand(o));
00249 }
00250
00251
00262 void unregister_obj(L4::Epiface *o, bool unmap = true) override
00263 {
00264 L4::Epiface::Stored_cap c;
00265
00266 if (!o || !o->obj_cap().is_valid())
00267 return;
00268
00269 c = o->obj_cap();
00270
00271 if (unmap)
00272 L4::Cap<L4::Task> (L4Re::This_task)->unmap(c.fpage(), L4_FP_ALL_SPACES);
00273
00274 // make sure unhandled ipc ends up with the null handler
00275 L4::Thread::Modify_senders todo;
00276 todo.add(~3UL, reinterpret_cast<l4_umword_t>(o),
00277 ~0UL, reinterpret_cast<l4_umword_t>
00278 (static_cast<L4::Epiface *>(&_null_handler)));
00279 _server->modify_senders(todo);
00280
00281 // we use bit 4 to indicated an internally allocated cap
00282 if (c.managed())
00283 cap_alloc.free(c);
00284
00285 o->set_server(0, L4::Cap<void>::Invalid);
00286 }
00287 };
00288
00292 template< typename LOOP_HOOKS = L4::Ipc_svr::Default_loop_hooks >
00293 class Registry_server : public L4::Server<LOOP_HOOKS>
00294 {
00295 private:
00296 typedef L4::Server<LOOP_HOOKS> Base;
00297 Object_registry _registry;
00298
00299 public:
00305 Registry_server() : _registry(this)
00306 {}
00307
00317 Registry_server(l4_utcb_t *, L4::Cap<L4::Thread> server,
00318 L4::Cap<L4::Factory> factory) L4_DEPRECATED("Omit UTCB pointer argument")
00319 : _registry(this, server, factory)
00320 {}
00321
00328 Registry_server(L4::Cap<L4::Thread> server,
00329 L4::Cap<L4::Factory> factory)
00330 : _registry(this, server, factory)
00331 {}
00332
00334 Object_registry const *registry() const { return &_registry; }
00336 Object_registry *registry() { return &_registry; }
00337
00344 void L4_NORETURN loop(l4_utcb_t *utcb = l4_utcb())
00345 { Base::template loop<L4::Runtime_error, Object_registry &>(_registry, utcb); }
00346
00355 template <typename Printer>
00356 void L4_NORETURN loop_dbg(Printer printer, l4_utcb_t *utcb = l4_utcb())
00357 {
00358 Base::template loop_dbg<L4::Runtime_error, Object_registry &, Printer>
00359 (_registry, printer, utcb);
00360 }
00361 };
00362 }

```

## 17.410 poll\_timeout\_kipclock

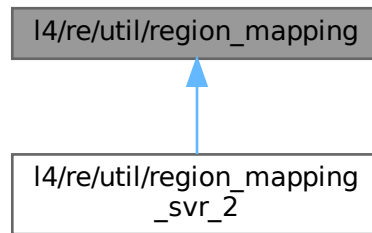
```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2012 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * economic rights: Technische Universität Dresden (Germany)
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008

```



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [L4Re](#)  
*[L4Re](#) C++ Interfaces.*
- namespace [L4Re::Util](#)  
*Documentation of the [L4](#) Runtime Environment utility functionality in C++.*

### 17.411.1 Detailed Description

Region handling.

Definition in file [region\\_mapping](#).

## 17.412 region\_mapping

[Go to the documentation of this file.](#)

```

00001 // -*- Mode: C++ -*-
00002 // vim:ft=cpp
00007 /*
00008 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00009 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00010 * Björn Döbel <doebel@os.inf.tu-dresden.de>
00011 * economic rights: Technische Universität Dresden (Germany)
00012 *
00013 * License: see LICENSE.spdx (in this directory or the directories above)
00014 */
00015
00016 #pragma once
00017
00018 #include <l4/cxx/avl_map>
00019 #include <l4/sys/types.h>
00020 #include <l4/re/rm>
00021
00022 namespace L4Re { namespace Util {
00023 class Region
00024 {
00025 private:
00026 l4_addr_t _start, _end;
00027 #ifdef CONFIG_L4RE_REGION_INFO
00028 char _dbg_name[40]; // Not a 0-terminating string
00029 unsigned char _dbg_name_len = 0;
00030 static_assert(sizeof(_dbg_name) < 256);
00031

```

```

00032 Rm::Offset _dbg_backing_offset = 0;
00033 #endif
00034
00035 public:
00036 Region() noexcept : _start(~0UL), _end(~0UL) {}
00037 Region(l4_addr_t addr) noexcept : _start(addr), _end(addr) {}
00038 Region(l4_addr_t start, l4_addr_t end) noexcept
00039 : _start(start), _end(end) {}
00040 Region(l4_addr_t start, l4_addr_t end,
00041 char const *name, unsigned name_len,
00042 Rm::Offset backing_offset) noexcept
00043 : _start(start), _end(end)
00044 {
00045 #ifdef CONFIG_L4RE_REGION_INFO
00046 _dbg_name_len = name_len > sizeof(_dbg_name)
00047 ? sizeof(_dbg_name) : name_len;
00048 for (unsigned i = 0; i < _dbg_name_len; ++i)
00049 _dbg_name[i] = name[i];
00050 _dbg_backing_offset = backing_offset;
00051 #else
00052 (void)name;
00053 (void)name_len;
00054 (void)backing_offset;
00055 #endif
00056 }
00057
00058 l4_addr_t start() const noexcept { return _start; }
00059 l4_addr_t end() const noexcept { return _end; }
00060 unsigned long size() const noexcept { return end() - start() + 1; }
00061 bool invalid() const noexcept { return _start == ~0UL && _end == ~0UL; }
00062 bool operator < (Region const &o) const noexcept
00063 { return end() < o.start(); }
00064 bool contains(Region const &o) const noexcept
00065 { return o.start() >= start() && o.end() <= end(); }
00066 bool operator == (Region const &o) const noexcept
00067 { return o.start() == start() && o.end() == end(); }
00068 ~Region() noexcept {}
00069
00070 #ifdef CONFIG_L4RE_REGION_INFO
00071 char const *name() const { return _dbg_name; }
00072 unsigned char name_len() const { return _dbg_name_len; }
00073 Rm::Offset backing_offset() const { return _dbg_backing_offset; }
00074 #else
00075 char const *name() const { return "N/A"; }
00076 unsigned char name_len() const { return 3; }
00077 Rm::Offset backing_offset() const { return 0; }
00078 #endif
00079 };
00080
00081 template< typename DS, typename OPS >
00082 class Region_handler
00083 {
00084 private:
00085 L4Re::Rm::Offset _offs;
00086 DS _mem;
00087 l4_cap_idx_t _client_cap = L4_INVALID_CAP;
00088 L4Re::Rm::Region_flags _flags;
00089
00090 public:
00091 typedef DS Dataspace;
00092 typedef OPS Ops;
00093 typedef typename OPS::Map_result Map_result;
00094
00095 Region_handler() noexcept : _offs(0), _mem(), _flags() {}
00096 Region_handler(Dataspace const &mem, l4_cap_idx_t client_cap,
00097 L4Re::Rm::Offset offset = 0,
00098 L4Re::Rm::Region_flags flags = L4Re::Rm::Region_flags(0)) noexcept
00099 : _offs(offset), _mem(mem), _client_cap(client_cap), _flags(flags)
00100 {}
00101
00102 Dataspace const &memory() const noexcept
00103 {
00104 return _mem;
00105 }
00106
00107 l4_cap_idx_t client_cap_idx() const noexcept
00108 {
00109 return _client_cap;
00110 }
00111
00112 L4Re::Rm::Offset offset() const noexcept
00113 {
00114 return _offs;
00115 }
00116
00117 constexpr bool is_ro() const noexcept
00118 {

```

```

00119 return !(_flags & L4Re::Rm::F::W);
00120 }
00121
00122 L4Re::Rm::Region_flags caching() const noexcept
00123 {
00124 return _flags & L4Re::Rm::F::Caching_mask;
00125 }
00126
00127 L4Re::Rm::Region_flags flags() const noexcept
00128 {
00129 return _flags;
00130 }
00131
00132 Region_handler operator + (l4_int64_t offset) const noexcept
00133 {
00134 Region_handler n = *this; n._offs += offset; return n;
00135 }
00136
00137 void free(l4_addr_t start, unsigned long size) const noexcept
00138 {
00139 Ops::free(this, start, size);
00140 }
00141
00142 int map(l4_addr_t addr, Region const &r, bool writable,
00143 Map_result *result) const
00144 {
00145 return Ops::map(this, addr, r, writable, result);
00146 }
00147
00148 int map_info(l4_addr_t *start_addr, l4_addr_t *end_addr) const
00149 {
00150 return Ops::map_info(this, start_addr, end_addr);
00151 }
00152
00153 };
00154
00155
00156 template< typename Hdlr, template<typename T> class Alloc >
00157 class Region_map
00158 {
00159 protected:
00160 typedef cxx::Avl_map< Region, Hdlr, cxx::Lt_functor, Alloc > Tree;
00161 Tree _rm;
00162 Tree _am;
00163
00164 private:
00165 l4_addr_t _start;
00166 l4_addr_t _end;
00167
00168 protected:
00169 void set_limits(l4_addr_t start, l4_addr_t end) noexcept
00170 {
00171 _start = start;
00172 _end = end;
00173 }
00174
00175 public:
00176 typedef typename Tree::Item_type Item;
00177 typedef typename Tree::Node Node;
00178 typedef typename Tree::Key_type Key_type;
00179 typedef Hdlr Region_handler;
00180
00181 typedef typename Tree::Iterator Iterator;
00182 typedef typename Tree::Const_iterator Const_iterator;
00183 typedef typename Tree::Rev_iterator Rev_iterator;
00184 typedef typename Tree::Const_rev_iterator Const_rev_iterator;
00185
00186 Iterator begin() noexcept { return _rm.begin(); }
00187 Const_iterator begin() const noexcept { return _rm.begin(); }
00188 Iterator end() noexcept { return _rm.end(); }
00189 Const_iterator end() const noexcept { return _rm.end(); }
00190
00191 Iterator area_begin() noexcept { return _am.begin(); }
00192 Const_iterator area_begin() const noexcept { return _am.begin(); }
00193 Iterator area_end() noexcept { return _am.end(); }
00194 Const_iterator area_end() const noexcept { return _am.end(); }
00195 Node area_find(Key_type const &c) const noexcept { return _am.find_node(c); }
00196
00197 l4_addr_t min_addr() const noexcept { return _start; }
00198 l4_addr_t max_addr() const noexcept { return _end; }
00199
00200
00201 Region_map(l4_addr_t start, l4_addr_t end) noexcept : _start(start), _end(end) {}
00202
00203 Node find(Key_type const &key) const noexcept
00204 {
00205 Node n = _rm.find_node(key);

```

```

00206 if (!n)
00207 return Node();
00208
00209 // 'find' should find any region overlapping with the searched one, the
00210 // caller should check for further requirements
00211 if (0)
00212 if (!n->first.contains(key))
00213 return Node();
00214
00215 return n;
00216 }
00217
00218 Node lower_bound(Key_type const &key) const noexcept
00219 {
00220 Node n = _rm.lower_bound_node(key);
00221 return n;
00222 }
00223
00224 Node lower_bound_area(Key_type const &key) const noexcept
00225 {
00226 Node n = _am.lower_bound_node(key);
00227 return n;
00228 }
00229
00230 l4_addr_t attach_area(l4_addr_t addr, unsigned long size,
00231 L4Re::Rm::Flags flags = L4Re::Rm::Flags(0),
00232 unsigned char align = L4_PAGESHIFT) noexcept
00233 {
00234 if (size < 2)
00235 return L4_INVALID_ADDR;
00236
00237 Region c;
00238
00239 if (!(flags & L4Re::Rm::F::Search_addr))
00240 {
00241 c = Region(addr, addr + size - 1);
00242 Node r = _am.find_node(c);
00243 if (r)
00244 return L4_INVALID_ADDR;
00245 }
00246
00247 while (flags & L4Re::Rm::F::Search_addr)
00248 {
00249 if (addr < min_addr() || (addr + size - 1) > max_addr())
00250 addr = min_addr();
00251 addr = find_free(addr, max_addr(), size, align, flags);
00252 if (addr == L4_INVALID_ADDR)
00253 return L4_INVALID_ADDR;
00254
00255 c = Region(addr, addr + size - 1);
00256 Node r = _am.find_node(c);
00257 if (!r)
00258 break;
00259
00260 if (r->first.end() >= max_addr())
00261 return L4_INVALID_ADDR;
00262
00263 addr = r->first.end() + 1;
00264 }
00265
00266 if (_am.insert(c, Hdlr(typename Hdlr::Dataspace(), 0, 0, flags.region_flags())).second == 0)
00267 return addr;
00268
00269 return L4_INVALID_ADDR;
00270 }
00271
00272 bool detach_area(l4_addr_t addr) noexcept
00273 {
00274 if (_am.remove(addr))
00275 return false;
00276
00277 return true;
00278 }
00279
00280 void *attach(void *addr, unsigned long size, Hdlr const &hdlr,
00281 L4Re::Rm::Flags attach_flags = L4Re::Rm::Flags(0),
00282 unsigned char align = L4_PAGESHIFT,
00283 char const *name = nullptr, unsigned name_len = 0,
00284 L4Re::Rm::Offset backing_offset = 0) noexcept
00285 {
00286 if (size < 2)
00287 return L4_INVALID_PTR;
00288
00289 l4_addr_t beg, end;
00290 int err = hdlr.map_info(&beg, &end);
00291 if (err > 0)

```

```

00293 {
00294 // Mapping address determined by underlying dataspace. Make sure we
00295 // prevent any additional alignment. We already know the place!
00296 beg += hdlr.offset();
00297 end = beg + size - 1U;
00298 align = L4_PAGESHIFT;
00299
00300 // In case of exact mappings, the supplied address must match because
00301 // we cannot remap.
00302 if (!(attach_flags & L4Re::Rm::F::Search_addr)
00303 && reinterpret_cast<l4_addr_t>(addr) != beg)
00304 return L4_INVALID_PTR;
00305
00306 // When searching for a suitable address, the start must cover the
00307 // dataspace beginning to "find" the right spot.
00308 if ((attach_flags & L4Re::Rm::F::Search_addr)
00309 && reinterpret_cast<l4_addr_t>(addr) > beg)
00310 return L4_INVALID_PTR;
00311 }
00312 else if (err == 0)
00313 {
00314 beg = reinterpret_cast<l4_addr_t>(addr);
00315 end = max_addr();
00316 }
00317 else if (err < 0)
00318 return L4_INVALID_PTR;
00319
00320 if (attach_flags & L4Re::Rm::F::In_area)
00321 {
00322 Node r = _am.find_node(Region(beg, beg + size - 1));
00323 if (!r || (r->second.flags() & L4Re::Rm::F::Reserved))
00324 return L4_INVALID_PTR;
00325
00326 end = r->first.end();
00327 }
00328
00329 if (attach_flags & L4Re::Rm::F::Search_addr)
00330 {
00331 beg = find_free(beg, end, size, align, attach_flags);
00332 if (beg == L4_INVALID_ADDR)
00333 return L4_INVALID_PTR;
00334 }
00335
00336 if (!(attach_flags & (L4Re::Rm::F::Search_addr | L4Re::Rm::F::In_area))
00337 && _am.find_node(Region(beg, beg + size - 1)))
00338 return L4_INVALID_PTR;
00339
00340 if (beg < min_addr() || beg + size - 1 > end)
00341 return L4_INVALID_PTR;
00342
00343 if (_rm.insert(Region(beg, beg + size - 1,
00344 name, name_len, backing_offset), hdlr).second
00345 == 0)
00346 return reinterpret_cast<void*>(beg);
00347
00348 return L4_INVALID_PTR;
00349 }
00350
00351 int detach(void *addr, unsigned long sz, unsigned flags,
00352 Region *reg, Hdlr *hdlr) noexcept
00353 {
00354 l4_addr_t a = reinterpret_cast<l4_addr_t>(addr);
00355 Region dr(a, a + sz - 1);
00356 Region res(~0UL, 0);
00357
00358 Node r = find(dr);
00359 if (!r)
00360 return -L4_ENOENT;
00361
00362 Region g = r->first;
00363 Hdlr const &h = r->second;
00364
00365 if (flags & L4Re::Rm::Detach_overlap || dr.contains(g))
00366 {
00367 // successful removal of the AVL tree item also frees the node
00368 Hdlr h_copy = h;
00369
00370 if (_rm.remove(g))
00371 return -L4_ENOENT;
00372
00373 if (!(flags & L4Re::Rm::Detach_keep) && (h_copy.flags() & L4Re::Rm::F::Detach_free))
00374 h_copy.free(0, g.size());
00375
00376 if (hdlr)
00377 *hdlr = h_copy;
00378 if (reg)
00379 *reg = g;

```

```

00380
00381 if (find(dr))
00382 return Rm::Detached_ds | Rm::Detach_again;
00383 else
00384 return Rm::Detached_ds;
00385 }
00386 else if (dr.start() <= g.start())
00387 {
00388 // move the start of a region
00389
00390 if (!(flags & L4Re::Rm::Detach_keep) && (h.flags() & L4Re::Rm::F::Detach_free))
00391 h.free(0, dr.end() + 1 - g.start());
00392
00393 unsigned long sz = dr.end() + 1 - g.start();
00394 Item &cn = const_cast<Item &>(*r);
00395 cn.first = Region(dr.end() + 1, g.end());
00396 cn.second = cn.second + sz;
00397 if (hdlr)
00398 *hdlr = Hdlr();
00399 if (reg)
00400 *reg = Region(g.start(), dr.end());
00401 if (find(dr))
00402 return Rm::Kept_ds | Rm::Detach_again;
00403 else
00404 return Rm::Kept_ds;
00405 }
00406 else if (dr.end() >= g.end())
00407 {
00408 // move the end of a region
00409
00410 if (!(flags & L4Re::Rm::Detach_keep)
00411 && (h.flags() & L4Re::Rm::F::Detach_free))
00412 h.free(dr.start() - g.start(), g.end() + 1 - dr.start());
00413
00414 Item &cn = const_cast<Item &>(*r);
00415 cn.first = Region(g.start(), dr.start() - 1);
00416 if (hdlr)
00417 *hdlr = Hdlr();
00418 if (reg)
00419 *reg = Region(dr.start(), g.end());
00420
00421 if (find(dr))
00422 return Rm::Kept_ds | Rm::Detach_again;
00423 else
00424 return Rm::Kept_ds;
00425 }
00426 else if (g.contains(dr))
00427 {
00428 // split a single region that contains the new region
00429
00430 if (!(flags & L4Re::Rm::Detach_keep) && (h.flags() & L4Re::Rm::F::Detach_free))
00431 h.free(dr.start() - g.start(), dr.size());
00432
00433 // first move the end off the existing region before the new one
00434 Item &cn = const_cast<Item &>(*r);
00435 cn.first = Region(g.start(), dr.start()-1);
00436
00437 int err;
00438
00439 // insert a second region for the remaining tail of
00440 // the old existing region
00441 err = _rm.insert(Region(dr.end() + 1, g.end()),
00442 h + (dr.end() + 1 - g.start())).second;
00443
00444 if (err)
00445 return err;
00446
00447 if (hdlr)
00448 *hdlr = h;
00449 if (reg)
00450 *reg = dr;
00451 return Rm::Split_ds;
00452 }
00453 return -L4_ENOENT;
00454 }
00455
00456 l4_addr_t find_free(l4_addr_t start, l4_addr_t end, l4_addr_t size,
00457 unsigned char align, L4Re::Rm::Flags attach_flags) const noexcept;
00458
00459 };
00460
00461
00462 template< typename Hdlr, template<typename T> class Alloc >
00463 l4_addr_t
00464 Region_map<Hdlr, Alloc>::find_free(l4_addr_t start, l4_addr_t end,
00465 unsigned long size, unsigned char align, L4Re::Rm::Flags attach_flags) const noexcept
00466 {

```



```

00467 l4_addr_t addr = start;
00468
00469 if (addr == ~0UL || addr < min_addr() || addr >= end)
00470 addr = min_addr();
00471
00472 addr = l4_round_size(addr, align);
00473 Node r;
00474
00475 for(;;)
00476 {
00477 if (addr > 0 && addr - 1 > end - size)
00478 return L4_INVALID_ADDR;
00479
00480 Region c(addr, addr + size - 1);
00481 r = _rm.find_node(c);
00482
00483 if (!r)
00484 {
00485 if (!(attach_flags & L4Re::Rm::F::In_area) && (r = _am.find_node(c)))
00486 {
00487 if (r->first.end() > end - size)
00488 return L4_INVALID_ADDR;
00489
00490 addr = l4_round_size(r->first.end() + 1, align);
00491 continue;
00492 }
00493 break;
00494 }
00495 else if (r->first.end() > end - size)
00496 return L4_INVALID_ADDR;
00497
00498 addr = l4_round_size(r->first.end() + 1, align);
00499 }
00500
00501 if (!r)
00502 return addr;
00503
00504 return L4_INVALID_ADDR;
00505 }
00506
00507 }}

```

## 17.413 region\_mapping\_svr\_2

```

00001 // vi:set ft=c++: -- Mode: C++ --
00002 /*
00003 * (c) 2014 Alexander Warg <alexander.warg@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/sys/types.h>
00010 #include <l4/re/rm>
00011 #include <l4/re/util/region_mapping>
00012
00013 namespace L4Re { namespace Util {
00014
00015 template<typename DERIVED, typename Dbg>
00016 struct Rm_server
00017 {
00018 private:
00019 DERIVED *rm() { return static_cast<DERIVED*>(this); }
00020 DERIVED const *rm() const { return static_cast<DERIVED const*>(this); }
00021 public:
00022
00023 long op_attach(L4Re::Rm::Rights, l4_addr_t &start,
00024 unsigned long size, Rm::Flags flags,
00025 L4::Ipc::Snd_fpage ds_cap, L4Re::Rm::Offset offs,
00026 unsigned char align, l4_cap_idx_t client_cap_idx,
00027 L4::Ipc::String<> name, L4Re::Rm::Offset backing_offset)
00028 {
00029 typename DERIVED::Dataspace ds;
00030
00031 if (!(flags & (Rm::F::Reserved | Rm::F::Kernel)))
00032 {
00033 if (long r = rm()->validate_ds
00034 (static_cast<DERIVED*>(this)->server_iface(), ds_cap,
00035 flags.region_flags(), &ds))
00036 return r;
00037 }
00038 }
00039 };
00040 }
00041
00042 }
00043
00044 }
00045
00046 }
00047

```

```

00048 size = l4_round_page(size);
00049 l4_addr_t start = l4_trunc_page(_start);
00050
00051 if (size < L4_PAGESIZE)
00052 return -L4_EINVAL;
00053
00054 Rm::Region_flags r_flags = flags.region_flags();
00055 Rm::Attach_flags a_flags = flags.attach_flags();
00056
00057 typename DERIVED::Region_handler handler(ds, client_cap_idx, offs, r_flags);
00058 start = l4_addr_t(rm()->attach(reinterpret_cast<void*>(start), size,
00059 handler, a_flags, align,
00060 name.data, name.length, backing_offset));
00061
00062 if (start == L4_INVALID_ADDR)
00063 return -L4_EADDRNOTAVAIL;
00064
00065 _start = start;
00066 return L4_EOK;
00067 }
00068
00072 long op_free_area(L4Re::Rm::Rights, l4_addr_t start)
00073 {
00074 if (!rm()->detach_area(start))
00075 return -L4_ENOENT;
00076
00077 return L4_EOK;
00078 }
00079
00083 long op_find(L4Re::Rm::Rights, l4_addr_t &addr, unsigned long &size,
00084 L4Re::Rm::Flags &flags, L4Re::Rm::Offset &offset,
00085 L4::Cap<L4Re::Dataspace> &m)
00086 {
00087 if (!DERIVED::Have_find)
00088 return -L4_EPERM;
00089
00090 Rm::Flags flag_area { 0 };
00091
00092 typename DERIVED::Node r = rm()->find(Region(addr, addr + size - 1));
00093 if (!r)
00094 {
00095 r = rm()->area_find(Region(addr, addr + size - 1));
00096 if (!r)
00097 return -L4_ENOENT;
00098 flag_area = Rm::F::In_area;
00099 }
00100
00101 addr = r->first.start();
00102 size = r->first.end() + 1 - addr;
00103
00104 flags = r->second.flags() | flag_area;
00105 offset = r->second.offset();
00106 m = L4::Cap<L4Re::Dataspace>(DERIVED::find_res(r->second.memory()));
00107 return L4_EOK;
00108 }
00109
00113 long op_detach(L4Re::Rm::Rights, l4_addr_t addr,
00114 unsigned long size, unsigned flags,
00115 l4_addr_t &start, l4_addr_t &rsz,
00116 l4_cap_idx_t &mem_cap)
00117 {
00118 Region r;
00119 typename DERIVED::Region_handler h;
00120 int err = rm()->detach(reinterpret_cast<void*>(addr), size, flags, &r, &h);
00121 if (err < 0)
00122 {
00123 start = rsz = 0;
00124 mem_cap = L4_INVALID_CAP;
00125 return err;
00126 }
00127
00128 if (r.invalid())
00129 {
00130 start = rsz = 0;
00131 mem_cap = L4_INVALID_CAP;
00132 return -L4_ENOENT;
00133 }
00134
00135 start = r.start();
00136 rsz = r.size();
00137 mem_cap = h.client_cap_idx();
00138 return err;
00139 }
00140
00144 long op_reserve_area(L4Re::Rm::Rights, l4_addr_t &start, unsigned long size,
00145 L4Re::Rm::Flags flags, unsigned char align)
00146 {

```

```

00147 start = rm()->attach_area(start, size, flags, align);
00148 if (start == L4_INVALID_ADDR)
00149 return -L4_EADDRNOTAVAIL;
00150 return L4_EOK;
00151 }
00152
00153 long op_get_regions(L4Re::Rm::Rights, l4_addr_t addr,
00154 L4::Ipc::Ret_array<L4Re::Rm::Region> regions)
00155 {
00156 typename DERIVED::Node r;
00157 unsigned num = 0;
00158 while ((r = rm()->lower_bound(Region(addr))))
00159 {
00160 Rm::Region &x = regions.value[num];
00161 x.start = r->first.start();
00162 x.end = r->first.end();
00163 x.flags = r->second.flags();
00164
00165 if (++num >= regions.max)
00166 break;
00167
00168 if (x.end >= rm()->max_addr())
00169 break;
00170 addr = x.end + 1;
00171 }
00172 return num;
00173 }
00174
00175 long op_get_areas(L4Re::Rm::Rights, l4_addr_t addr,
00176 L4::Ipc::Ret_array<L4Re::Rm::Area> areas)
00177 {
00178 typename DERIVED::Node r;
00179 unsigned num = 0;
00180 while ((r = rm()->lower_bound_area(Region(addr))))
00181 {
00182 Rm::Area &x = areas.value[num];
00183 x.start = r->first.start();
00184 x.end = r->first.end();
00185
00186 if (++num >= areas.max)
00187 break;
00188
00189 if (x.end >= rm()->max_addr())
00190 break;
00191 addr = x.end + 1;
00192 }
00193 return num;
00194 }
00195
00196 private:
00197 static void pager_set_result(L4::Ipc::Opt<L4::Ipc::Snd_fpage> *fp,
00198 L4::Ipc::Snd_fpage const &f)
00199 { *fp = f; }
00200
00201 static void pager_set_result(L4::Ipc::Opt<L4::Ipc::Snd_fpage> *, ...)
00202 {}
00203
00204 public:
00205 long op_io_page_fault(L4::Io_pager::Rights, l4_fpage_t, l4_umword_t,
00206 L4::Ipc::Opt<L4::Ipc::Snd_fpage> &)
00207 {
00208 // generate exception
00209 return -L4_ENOMEM;
00210 }
00211
00212 long op_page_fault(L4::Pager::Rights, l4_umword_t addr, l4_umword_t pc,
00213 L4::Ipc::Opt<L4::Ipc::Snd_fpage> &fp)
00214 {
00215 Dbg(Dbg::Server).printf("page fault: %lx pc=%lx\n", addr, pc);
00216
00217 bool need_w = addr & 2;
00218 bool need_x = addr & 4;
00219
00220 typename DERIVED::Node n = rm()->find(addr);
00221
00222 if (!n || !n->second.memory())
00223 {
00224 Dbg(Dbg::Warn, "rm").printf("unhandled %s page fault at 0x%lx pc=0x%lx\n",
00225 need_w ? "write" :
00226 need_x ? "instruction" : "read", addr, pc);
00227
00228 // generate exception
00229 return -L4_ENOMEM;
00230 }
00231
00232 if (!n->second.flags() & L4Re::Rm::F::W) && need_w)
00233 {

```

```

00243 Dbg(Dbg::Warn, "rm").printf("write page fault in readonly region at 0x%lx pc=0x%lx\n",
00244 addr, pc);
00245 // generate exception
00246 return -L4_EACCESS;
00247 }
00248
00249 if (!(n->second.flags() & L4Re::Rm::F::X) && need_x)
00250 {
00251 Dbg(Dbg::Warn, "rm").printf("instruction page fault in non-exec region at 0x%lx pc=0x%lx\n",
00252 addr, pc);
00253 // generate exception
00254 return -L4_EACCESS;
00255 }
00256
00257 // This check is optional but avoids doing a map operation that will not
00258 // work. We shall never get here from a page-fault but only from
00259 // artificial page handling.
00260 if (n->second.flags() & (L4Re::Rm::F::Kernel | L4Re::Rm::F::Reserved))
00261 {
00262 Dbg(Dbg::Warn, "rm").printf("page fault handling in kernel-memory provided region or reserved
00263 region at 0x%lx pc=0x%lx\n",
00264 addr, pc);
00265 // generate exception
00266 return -L4_ENODEV;
00267 }
00268
00269 typename DERIVED::Region_handler::Ops::Map_result map_res;
00270 if (int err = n->second.map(addr, n->first, need_w, &map_res))
00271 {
00272 Dbg(Dbg::Warn, "rm").printf("mapping for page fault failed with error %d at 0x%lx pc=0x%lx\n",
00273 err, addr, pc);
00274 // generate exception
00275 return -L4_ENOMEM;
00276 }
00277
00278 pager_set_result(&fp, map_res);
00279 return L4_EOK;
00280 }
00281
00282 long op_get_info(L4Re::Rm::Rights, l4_addr_t addr,
00283 L4::Ipc::String<char> &name, L4Re::Rm::Offset &backing_offset)
00284 {
00285 #ifdef CONFIG_L4RE_REGION_INFO
00286 typename DERIVED::Node r = rm()->find(Region(addr));
00287 if (!r)
00288 return -L4_ENOENT;
00289 backing_offset = r->first.backing_offset();
00290 unsigned long i;
00291 char const *src = r->first.name();
00292 unsigned src_len = r->first.name_len();
00293 for (i = 0; i < src_len && i < name.length - 1; ++i)
00294 name.data[i] = src[i];
00295 name.length = i + 1;
00296 name.data[i] = '\0';
00297 return L4_EOK;
00298 #else
00299 (void)addr;
00300 (void)name;
00301 (void)backing_offset;
00302 return -L4_ENOSYS;
00303 #endif
00304 }
00305
00306 }

```

## 17.414 l4/re/shared\_cap File Reference

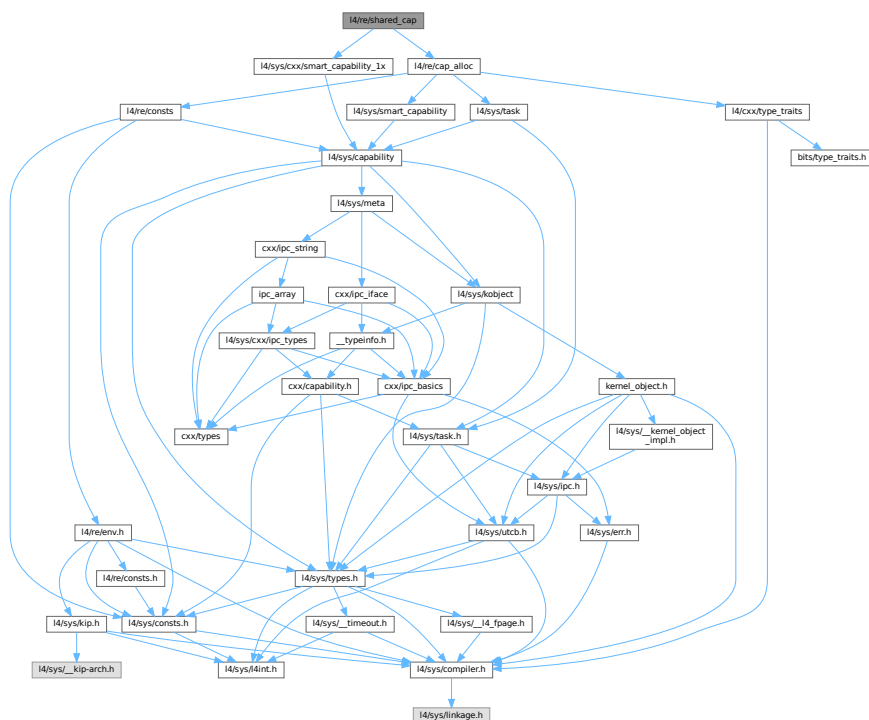
Shared\_cap / Shared\_del\_cap.

```

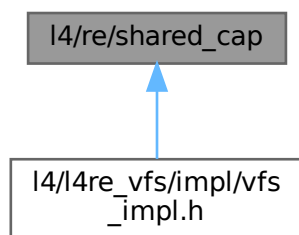
#include <l4/re/cap_alloc>
#include <l4/sys/cxx/smart_capability_1x>

```

Include dependency graph for shared\_cap:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace **L4Re**  
*L4Re C++ Interfaces.*

## Typedefs

- `template<typename T>`  
`using L4Re::Shared_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_ALL_SPACES>>`

*Shared capability that implements automatic free and unmap of the capability selector.*

- `template<typename T>`  
`using L4Re::shared_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_ALL_SPACES>>`  
*Shared capability that implements automatic free and unmap of the capability selector.*
- `template<typename T>`  
`using L4Re::Shared_del_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_DELETE_OBJ>>`  
*Shared capability that implements automatic free and unmap+delete of the capability selector.*
- `template<typename T>`  
`using L4Re::shared_del_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_DELETE_OBJ>>`  
*Shared capability that implements automatic free and unmap+delete of the capability selector.*

## Functions

- `template<typename T>`  
`Shared_cap< T > L4Re::make_shared_cap (L4Re::Cap_alloc *ca)`  
*Allocate a capability slot and wrap it in a [Shared\\_cap](#).*
- `template<typename T>`  
`Shared_del_cap< T > L4Re::make_shared_del_cap (L4Re::Cap_alloc *ca)`  
*Allocate a capability slot and wrap it in a [Shared\\_del\\_cap](#).*

## 17.414.1 Detailed Description

Shared\_cap / Shared\_del\_cap.

Definition in file [shared\\_cap](#).

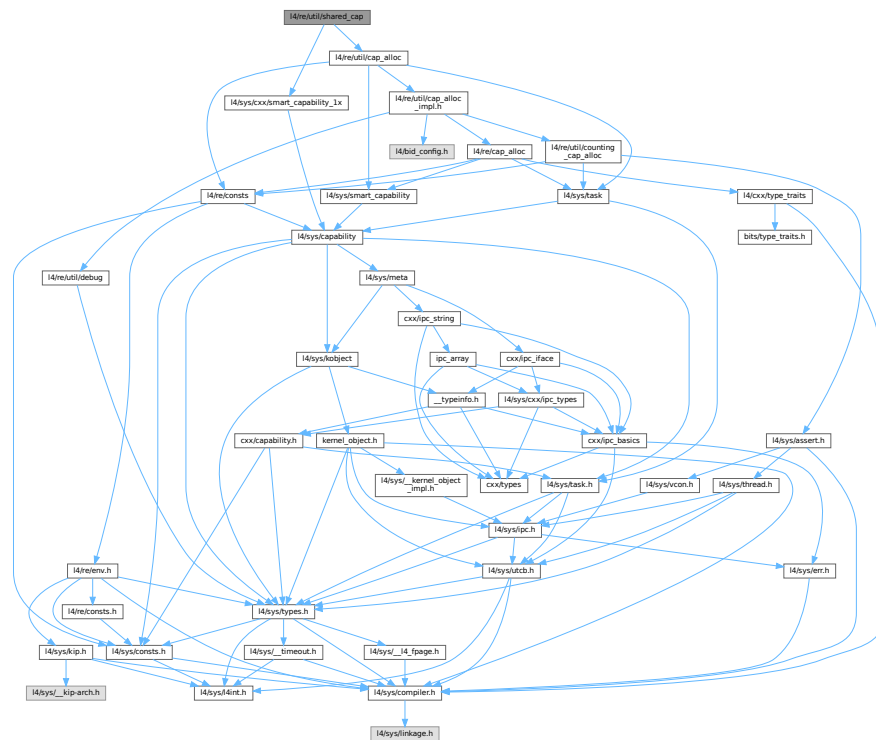
## 17.415 shared\_cap

[Go to the documentation of this file.](#)

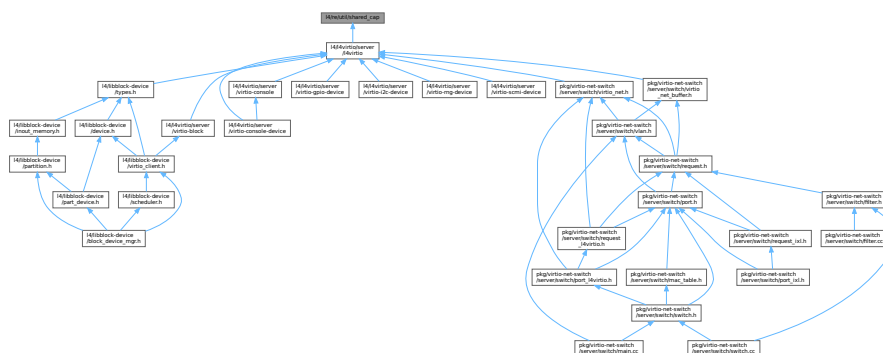
```
00001 // vim:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2018 Alexander Warg <alexander.warg@kernkonzept.com>
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011
00012 #pragma once
00013
00014 #include <l4/re/cap_alloc>
00015 #include <l4/sys/cxx/smart_capability_1x>
00016
00017 namespace L4Re {
00018
00032 template< typename T >
00033 using Shared_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_ALL_SPACES>;
00035 template< typename T >
00036 using shared_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_ALL_SPACES>;
00037
00047 template< typename T >
00048 Shared_cap<T>
00049 make_shared_cap(L4Re::Cap_alloc *ca)
00050 { return Shared_cap<T>(ca->alloc<T>(), ca); }
00051
00068 template< typename T >
00069 using Shared_del_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_DELETE_OBJ>;
00071 template<typename T>
00072 using shared_del_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_DELETE_OBJ>;
00073
00083 template< typename T >
00084 Shared_del_cap<T>
00085 make_shared_del_cap(L4Re::Cap_alloc *ca)
00086 { return Shared_del_cap<T>(ca->alloc<T>(), ca); }
00087
00088 } // namespace L4Re
```

$$\text{Shared\_cap} / \text{Shared\_del\_cap}.$$

Include dependency graph for shared\_cap:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace **L4Re**  
*L4Re C++ Interfaces.*
- namespace **L4Re::Util**

Documentation of the [L4 Runtime Environment](#) utility functionality in C++.

## Typedefs

- `template<typename T>`  
`using L4Re::Util::Shared_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_ALL_SPACES>>`  
*Shared capability that implements automatic free and unmap of the capability selector.*
- `template<typename T>`  
`using L4Re::Util::shared_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_ALL_SPACES>>`  
*Shared capability that implements automatic free and unmap of the capability selector.*
- `template<typename T>`  
`using L4Re::Util::Shared_del_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_DELETE_OBJ>>`  
*Shared capability that implements automatic free and unmap+delete of the capability selector.*
- `template<typename T>`  
`using L4Re::Util::shared_del_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_DELETE_OBJ>>`  
*Shared capability that implements automatic free and unmap+delete of the capability selector.*

## Functions

- `template<typename T>`  
`Shared_cap< T > L4Re::Util::make_shared_cap ()`  
*Allocate a capability slot and wrap it in a [Shared\\_cap](#).*
- `template<typename T>`  
`Shared_del_cap< T > L4Re::Util::make_shared_del_cap ()`  
*Allocate a capability slot and wrap it in a [Shared\\_del\\_cap](#).*

### 17.416.1 Detailed Description

`Shared_cap` / `Shared_del_cap`.

Definition in file [shared\\_cap](#).

## 17.417 shared\_cap

[Go to the documentation of this file.](#)

```
00001 // vim:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2017 Alexander Warg <alexander.warg@kernkonzept.com>
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011
00012 #pragma once
00013
00014 #include <l4/re/util/cap_alloc>
00015 #include <l4/sys/cxx/smart_capability_lx>
00016
00017 namespace L4Re { namespace Util {
00018
00047 template< typename T >
00048 using Shared_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_ALL_SPACES>;
00050 template< typename T >
00051 using shared_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_ALL_SPACES>;
00052
00058 template< typename T >
00059 Shared_cap<T>
00060 make_shared_cap()
00061 { return Shared_cap<T>(cap_alloc.alloc<T>()); }
00062
00097 template< typename T >
00098 using Shared_del_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_DELETE_OBJ>;
00100 template< typename T >
```



```

00101 using shared_del_cap = L4::Detail::Shared_cap_impl<T, Smart_count_cap<L4_FP_DELETE_OBJ>;
00102
00108 template< typename T >
00109 Shared_del_cap<T>
00110 make_shared_del_cap()
00111 { return Shared_del_cap<T>(cap_alloc.alloc<T>()); }
00112
00113 } // namespace L4Re::Util
00114

```

## 17.418 l4/re/unique\_cap File Reference

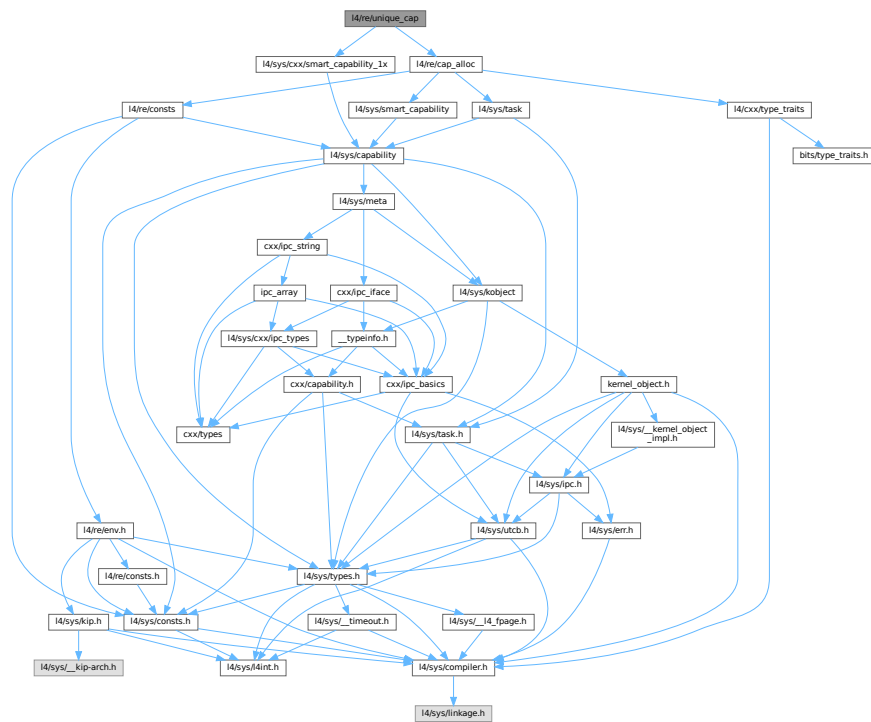
Unique\_cap / Unique\_del\_cap.

```

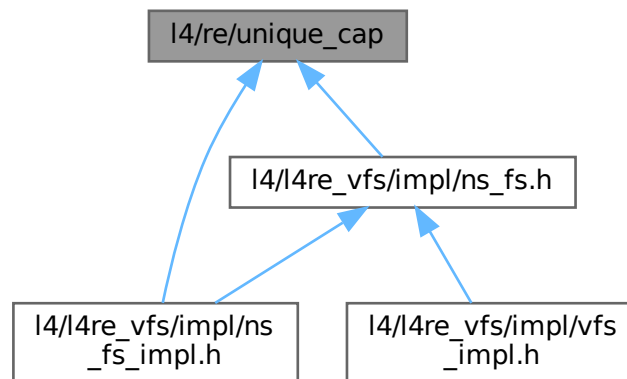
#include <l4/re/cap_alloc>
#include <l4/sys/cxx/smart_capability_1x>

```

Include dependency graph for unique\_cap:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [L4Re](#)  
[L4Re](#) C++ Interfaces.

## Typedefs

- `template<typename T>`  
`using L4Re::Unique\_cap = L4::Detail::Unique_cap_impl<T, Smart\_cap\_auto<L4\_FP\_ALL\_SPACES>>`  
*Unique capability that implements automatic free and unmap of the capability selector.*
- `template<typename T>`  
`using L4Re::unique\_cap = L4::Detail::Unique_cap_impl<T, Smart\_cap\_auto<L4\_FP\_ALL\_SPACES>>`  
*Unique capability that implements automatic free and unmap of the capability selector.*
- `template<typename T>`  
`using L4Re::Unique\_del\_cap = L4::Detail::Unique_cap_impl<T, Smart\_cap\_auto<L4\_FP\_DELETE\_OBJ>>`  
*Unique capability that implements automatic free and unmap+delete of the capability selector.*
- `template<typename T>`  
`using L4Re::unique\_del\_cap = L4::Detail::Unique_cap_impl<T, Smart\_cap\_auto<L4\_FP\_DELETE\_OBJ>>`  
*Unique capability that implements automatic free and unmap+delete of the capability selector.*

## Functions

- `template<typename T>`  
`Unique\_cap< T > L4Re::make\_unique\_cap (L4Re::Cap\_alloc *ca)`  
*Allocate a capability slot and wrap it in an [Unique\\_cap](#).*
- `template<typename T>`  
`Unique\_del\_cap< T > L4Re::make\_unique\_del\_cap (L4Re::Cap\_alloc *ca)`  
*Allocate a capability slot and wrap it in an [Unique\\_del\\_cap](#).*

## 17.418.1 Detailed Description

Unique\_cap / Unique\_del\_cap.

Definition in file [unique\\_cap](#).

## 17.419 unique\_cap

[Go to the documentation of this file.](#)

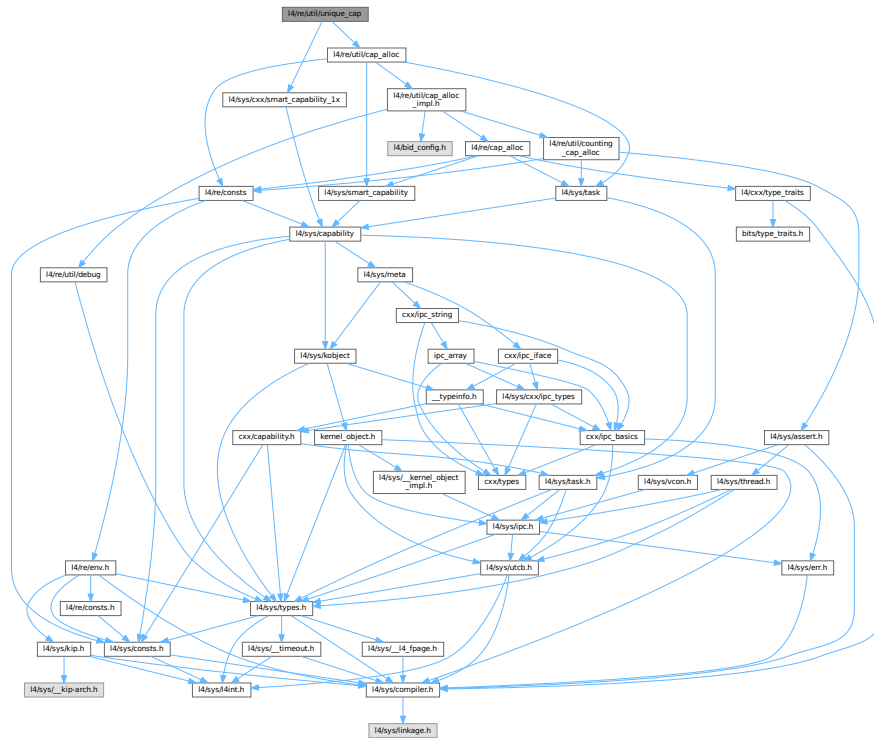
```
00001 // vim:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2017 Alexander Warg <alexander.warg@kernkonzept.com>
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011
00012 #pragma once
00013
00014 #include <l4/re/cap_alloc>
00015 #include <l4/sys/cxx/smart_capability_lx>
00016
00017 namespace L4Re {
00018
00030 template< typename T >
00031 using Unique_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_ALL_SPACES>;
00033 template< typename T >
00034 using unique_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_ALL_SPACES>;
00035
00045 template< typename T >
00046 Unique_cap<T>
00047 make_unique_cap(L4Re::Cap_alloc *ca)
00048 { return Unique_cap<T>(ca->alloc<T>(), ca); }
00049
00063 template< typename T >
00064 using Unique_del_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_DELETE_OBJ>;
00066 template<typename T>
00067 using unique_del_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_DELETE_OBJ>;
00068
00078 template< typename T >
00079 Unique_del_cap<T>
00080 make_unique_del_cap(L4Re::Cap_alloc *ca)
00081 { return Unique_del_cap<T>(ca->alloc<T>(), ca); }
00082
00083 }
```

## 17.420 l4/re/util/unique\_cap File Reference

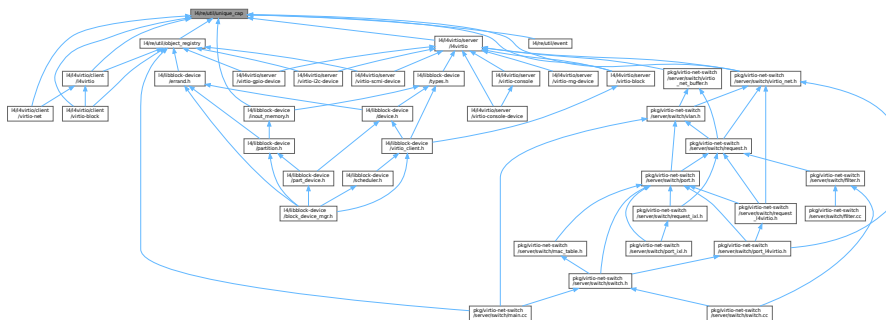
Unique\_cap / Unique\_del\_cap.

```
#include <l4/re/util/cap_alloc>
#include <l4/sys/cxx/smart_capability_lx>
```

Include dependency graph for unique\_cap:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [L4Re](#)  
[L4Re C++ Interfaces](#).
- namespace [L4Re::Util](#)  
*Documentation of the [L4 Runtime Environment](#) utility functionality in C++.*

## Typedefs

- `template<typename T>`  
`using L4Re::Util::Unique_cap = L4::Detail::Unique_cap impl<T, Smart_cap_auto<L4_FP_ALL_SPACES>>`

*Unique capability that implements automatic free and unmap of the capability selector.*

- `template<typename T>`  
`using L4Re::Util::unique_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_ALL_SPACES>>`

*Unique capability that implements automatic free and unmap of the capability selector.*

- `template<typename T>`  
`using L4Re::Util::Unique_del_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_DELETE_OBJ>>`

*Unique capability that implements automatic free and unmap+delete of the capability selector.*

- `template<typename T>`  
`using L4Re::Util::unique_del_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_DELETE_OBJ>>`

*Unique capability that implements automatic free and unmap+delete of the capability selector.*

## Functions

- `template<typename T>`  
`Unique_cap< T > L4Re::Util::make_unique_cap ()`  
*Allocate a capability slot and wrap it in an `Unique_cap`.*
- `template<typename T>`  
`Unique_del_cap< T > L4Re::Util::make_unique_del_cap ()`  
*Allocate a capability slot and wrap it in an `Unique_del_cap`.*

## 17.420.1 Detailed Description

Unique\_cap / Unique\_del\_cap.

Definition in file [unique\\_cap](#).

## 17.421 unique\_cap

[Go to the documentation of this file.](#)

```
00001 // vim:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2017 Alexander Warg <alexander.warg@kernkonzept.com>
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011
00012 #pragma once
00013
00014 #include <l4/re/util/cap_alloc>
00015 #include <l4/sys/cxx/smart_capability_1x>
00016
00017 namespace L4Re { namespace Util {
00018
00042 template< typename T >
00043 using Unique_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_ALL_SPACES>;
00044 template< typename T >
00045 using unique_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_ALL_SPACES>;
00046
00053 template< typename T >
00054 Unique_cap<T>
00055 make_unique_cap()
00056 { return Unique_cap<T>(cap_alloc.alloc<T>()); }
00057
00085 template< typename T >
00086 using Unique_del_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_DELETE_OBJ>;
00087 template< typename T >
00088 using unique_del_cap = L4::Detail::Unique_cap_impl<T, Smart_cap_auto<L4_FP_DELETE_OBJ>;
00089
00096 template< typename T >
00097 Unique_del_cap<T>
00098 make_unique_del_cap()
00099 { return Unique_del_cap<T>(cap_alloc.alloc<T>()); }
00100
00101 }}
00102
```

## 17.422 vcon\_svr

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2011 Alexander Warg <warg@os.inf.tu-dresden.de>,
00004 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00005 * Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00006 * economic rights: Technische Universität Dresden (Germany)
00007 *
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010 #pragma once
00011
00012 #include <l4/sys/types.h>
00013 #include <l4/sys/vcon>
00014 #include <l4/sys/cxx/ipc_legacy>
00015 #include <l4/cxx/minmax>
00016
00017 namespace L4Re { namespace Util {
00018
00035 template< typename SVR >
00036 class Vcon_svr
00037 {
00038 public:
00039 L4_RPC_LEGACY_DISPATCH(L4::Vcon);
00040
00041 l4_msgtag_t op_dispatch(l4_utcb_t *utcb, l4_msgtag_t tag, L4::Vcon::Rights)
00042 {
00043 l4_msg_regs_t *m = l4_utcb_mr_u(utcb);
00044 L4::Opcode op = m->mr[0];
00045
00046 switch (op)
00047 {
00048 case L4_VCON_WRITE_OP:
00049 if (tag.words() < 3)
00050 return l4_msgtag(-L4_ENOREPLY, 0, 0, 0);
00051
00052 this_vcon()->vcon_write(reinterpret_cast<char const *>(&m->mr[2]),
00053 m->mr[1]);
00054 return l4_msgtag(-L4_ENOREPLY, 0, 0, 0);
00055 case L4_VCON_SET_ATTR_OP:
00056 {
00057 if (tag.words() < 4)
00058 return l4_msgtag(-L4_EINVAL, 0, 0, 0);
00059
00060 auto attr = reinterpret_cast<l4_vcon_attr_t const *>(&m->mr[1]);
00061 return l4_msgtag(this_vcon()->vcon_set_attr(attr), 0, 0, 0);
00062 }
00063 case L4_VCON_GET_ATTR_OP:
00064 {
00065 auto attr = reinterpret_cast<l4_vcon_attr_t *>(&m->mr[1]);
00066 return l4_msgtag(this_vcon()->vcon_get_attr(attr), 4, 0, 0);
00067 }
00068 default:
00069 break;
00070 }
00071
00072 unsigned const max_size = sizeof(l4_utcb_mr()->mr) - sizeof(l4_utcb_mr()->mr[0]);
00073 char buf[max_size];
00074
00075 unsigned size = cxx::min<unsigned>(op > 16, max_size);
00076
00077 // Hmm, could we avoid the double copy here?
00078 l4_umword_t v = this_vcon()->vcon_read(buf, size);
00079 unsigned bytes = v & L4_VCON_READ_SIZE_MASK;
00080
00081 if (bytes <= size)
00082 v |= L4_VCON_READ_STAT_DONE;
00083
00084 m->mr[0] = v;
00085 __builtin_memcpy(&m->mr[1], buf, bytes);
00086
00087 return l4_msgtag(0, l4_bytes_to_mwords(bytes) + 1, 0, 0);
00088 }
00089
00090 unsigned vcon_read(char *buf, unsigned size) noexcept;
00091 void vcon_write(const char *buf, unsigned size) noexcept;
00092 int vcon_set_attr(l4_vcon_attr_t const *) noexcept
00093 { return -L4_EOK; }
00094 int vcon_get_attr(l4_vcon_attr_t *attr) noexcept
00095 {
00096 attr->l_flags = attr->o_flags = attr->i_flags = 0;
00097 return -L4_EOK;
00098 }
00099 }
00100

```

```

00101
00102 private:
00103 SVR const *this_vcon() const { return static_cast<SVR const *>(this); }
00104 SVR *this_vcon() { return static_cast<SVR *>(this); }
00105 };
00106
00107 }}

```

## 17.423 goos\_fb

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 #pragma once
00010
00011 #include <l4/re/env>
00012 #include <l4/re/namespace>
00013 #include <l4/re/rm>
00014 #include <l4/re/util/cap_alloc>
00015 #include <l4/re/util/env_ns>
00016 #include <l4/re/util/video/goos_fb>
00017 #include <l4/re/video/goos>
00018
00019 namespace L4Re { namespace Util { namespace Video {
00020
00021 class Goos_fb
00022 {
00023 private:
00024 L4::Cap<L4Re::Video::Goos> _goos;
00025 L4Re::Video::View _view;
00026 L4::Cap<L4Re::Dataspace> _buffer;
00027
00028 enum Flags
00029 {
00030 F_dyn_buffer = 0x01,
00031 F_dyn_view = 0x02,
00032 F_dyn_goos = 0x04,
00033 };
00034 unsigned _flags;
00035
00036 unsigned _buffer_index;
00037
00038 private:
00039 long init()
00040 {
00041 using namespace L4Re::Video;
00042
00043 Goos::Info gi;
00044 long ret = _goos->info(&gi);
00045 if (ret < 0)
00046 return ret;
00047
00048 if (gi.has_dynamic_views())
00049 {
00050 ret = _goos->create_view(&_view);
00051 if (ret < 0)
00052 return ret;
00053 _flags |= F_dyn_view;
00054 }
00055 else // we just assume view 0 to be our's and ignore other possible views
00056 _view = _goos->view(0);
00057
00058 View::Info vi;
00059 ret = _view.info(&vi);
00060 if (ret < 0)
00061 return ret;
00062 _buffer = cap_alloc.alloc<L4Re::Dataspace>();
00063 if (!_buffer)
00064 return -L4_ENOMEM;
00065
00066 if (vi.has_static_buffer())
00067 {
00068 ret = _goos->get_static_buffer(vi.buffer_index, _buffer);
00069 if (ret < 0)
00070 return ret;
00071 }
00072
00073 }
00074 }

```

```

00074 else
00075 {
00076 unsigned long buffer_sz = gi.pixel_info.bytes_per_pixel() * gi.width
00077 * gi.height;
00078 ret = _goos->create_buffer(buffer_sz, _buffer);
00079 if (ret < 0)
00080 return ret;
00081
00082 _buffer_index = static_cast<unsigned>(ret);
00083 _flags |= F_dyn_buffer;
00084
00085 // use the allocated buffer, at offset 0
00086 vi.buffer_index = _buffer_index;
00087 vi.buffer_offset = 0;
00088 vi.pixel_info = gi.pixel_info;
00089 vi.bytes_per_line = gi.width * gi.pixel_info.bytes_per_pixel();
00090
00091 // we want a fullscreen view
00092 vi.xpos = 0;
00093 vi.ypos = 0;
00094 vi.width = gi.width;
00095 vi.height = gi.height;
00096
00097 ret = _view.set_info(vi);
00098 if (ret < 0)
00099 return ret;
00100
00101 ret = _view.push_top();
00102 if (ret < 0)
00103 return ret;
00104 }
00105
00106 return 0;
00107 }
00108
00109 Goos_fb(Goos_fb const &);
00110 void operator = (Goos_fb const &);
00111
00112 public:
00113 Goos_fb()
00114 : _goos(L4_INVALID_CAP), _buffer(L4_INVALID_CAP), _flags(0), _buffer_index(0)
00115 {}
00116
00117 long init(L4Re::Cap<L4Re::Video::Goos> goos)
00118 {
00119 _goos = goos;
00120 return init();
00121 }
00122
00123 long init(char const *name)
00124 {
00125 Env_ns ns;
00126 _goos = ns.query<L4Re::Video::Goos>(name);
00127 if (!_goos)
00128 return _goos.cap();
00129
00130 _flags |= F_dyn_goos;
00131
00132 return init();
00133 }
00134
00135 ~Goos_fb()
00136 {
00137 if (!_goos.is_valid())
00138 return;
00139
00140 if (_flags & F_dyn_view)
00141 _goos->delete_view(_view);
00142
00143 if (_flags & F_dyn_buffer)
00144 _goos->delete_buffer(_buffer_index);
00145
00146 if (_buffer.is_valid())
00147 cap_alloc.free(_buffer);
00148
00149 if (_flags & F_dyn_goos)
00150 cap_alloc.free(_goos);
00151 }
00152
00153 int view_info(L4Re::Video::View::Info *info)
00154 { return _view.info(info); }
00155
00156 L4Re::Video::View const *view() const { return &_view; }
00157 L4Re::Video::View *view() { return &_view; }
00158
00159 L4::Cap<L4Re::Dataspace> buffer() const { return _buffer; }
00160 void *attach_buffer()

```



```

00161 {
00162 void *fb_addr = 0;
00163 if (!_goos)
00164 return nullptr;
00165
00166 long ret = L4Re::Env::env()->rm()
00167 ->attach(&fb_addr, _buffer->size(),
00168 L4Re::Rm::F::Search_addr | L4Re::Rm::F::RW, _buffer,
00169 0, L4_SUPERPAGESHIFT);
00170 if (ret < 0)
00171 return nullptr;
00172
00173 return fb_addr;
00174 }
00175
00176 int refresh(int x, int y, int w, int h)
00177 { return _view.refresh(x, y, w, h); }
00178
00179 L4::Cap<L4Re::Video::Goos> goos() const { return _goos; }
00180 };
00181 }}}

```

## 17.424 goos\_svr

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009
00010 #pragma once
00011
00012 #include <l4/re/dataspace>
00013 #include <l4/re/video/goos>
00014 #include <l4/re/video/goos-sys.h>
00015
00016 #include <l4/sys/capability>
00017 #include <l4/sys/cxx/ipc_legacy>
00018
00019 namespace L4Re { namespace Util { namespace Video {
00020
00021 class Goos_svr
00022 {
00023 public:
00024 typedef L4Re::Video::Goos::Rights Rights;
00025 protected:
00026 L4::Cap<L4Re::Dataspace> _fb_ds;
00027 L4Re::Video::Goos::Info _screen_info;
00028 L4Re::Video::View::Info _view_info;
00029 public:
00030 L4_RPC_LEGACY_DISPATCH(L4Re::Video::Goos);
00031 L4::Cap<L4Re::Dataspace> get_fb() const { return _fb_ds; }
00032
00033 L4Re::Video::Goos::Info const *screen_info() const { return &_screen_info; }
00034
00035 L4Re::Video::View::Info const *view_info() const { return &_view_info; }
00036
00037 virtual int refresh(int x, int y, int w, int h)
00038 { (void)x; (void)y; (void)w; (void)h; return -L4_ENOSYS; }
00039
00040 void init_infos()
00041 {
00042 using L4Re::Video::View;
00043
00044 _view_info.flags = View::F_none;
00045
00046 _view_info.view_index = 0;
00047 _view_info.xpos = 0;
00048 _view_info.ypos = 0;
00049 _view_info.width = _screen_info.width;
00050 _view_info.height = _screen_info.height;
00051 _view_info.pixel_info = _screen_info.pixel_info;
00052 _view_info.buffer_index = 0;
00053 }
00054
00055 virtual ~Goos_svr() {}
00056
00057 long op_view_info(Rights, unsigned idx, L4Re::Video::View::Info &info)
00058 {

```

```

00100 if (idx != 0)
00101 return -L4_ERANGE;
00102
00103 info = _view_info;
00104 return L4_EOK;
00105 }
00106
00107 long op_info(Rights, L4Re::Video::Goos::Info &info)
00108 {
00109 info = _screen_info;
00110 return L4_EOK;
00111 }
00112
00113 long op_get_static_buffer(Rights, unsigned idx,
00114 L4::Ipc::Cap<L4Re::Dataspace> &ds)
00115 {
00116 if (idx != 0)
00117 return -L4_ERANGE;
00118
00119 ds = L4::Ipc::Cap<L4Re::Dataspace>(_fb_ds, L4_CAP_FPAGE_RW);
00120 return L4_EOK;
00121 }
00122
00123 long op_refresh(Rights, int x, int y, int w, int h)
00124 { return refresh(x, y, w, h); }
00125
00126 long op_view_refresh(Rights, unsigned idx, int x, int y, int w, int h)
00127 {
00128 if (idx != 0)
00129 return -L4_ERANGE;
00130
00131 return refresh(x, y, w, h);
00132 }
00133
00134 long op_set_view_info(Rights, unsigned, L4Re::Video::View::Info)
00135 { return -L4_ENOSYS; }
00136
00137 long op_view_stack(Rights, unsigned, unsigned, bool)
00138 { return -L4_ENOSYS; }
00139
00140 long op_delete_view(Rights, unsigned)
00141 { return -L4_ENOSYS; }
00142
00143 long op_create_view(Rights)
00144 { return -L4_ENOSYS; }
00145
00146 long op_create_buffer(Rights, unsigned long,
00147 L4::Ipc::Cap<L4Re::Dataspace> &)
00148 { return -L4_ENOSYS; }
00149
00150 long op_delete_buffer(Rights, unsigned)
00151 { return -L4_ENOSYS; }
00152 };
00153
00154
00155 }}}

```

## 17.425 colors

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009
00010 #pragma once
00011
00012 #include <l4/sys/compiler.h>
00013 #include <l4/cxx/minmax>
00014
00015 namespace L4Re { namespace Video {
00016
00021 class L4_EXPORT Color_component
00022 {
00023 private:
00024 unsigned char _bits;
00025 unsigned char _shift;
00026
00027 public:
00029 Color_component() : _bits(0), _shift(0) {}

```

```

00030
00036 Color_component(unsigned char bits, unsigned char shift)
00037 : _bits(bits), _shift(shift) {}
00038
00043 unsigned char size() const { return _bits; }
00044
00049 unsigned char shift() const { return _shift; }
00050
00055 bool operator == (Color_component const &o) const
00056 { return _shift == o._shift && _bits == o._bits; }
00057
00063 int get(unsigned long v) const
00064 {
00065 return ((v > _shift) & ~(~0UL < _bits)) < (16UL - _bits);
00066 }
00067
00073 long unsigned set(int v) const
00074 { return (static_cast<unsigned long>(v) > (16UL - _bits)) < _shift; }
00075
00080 template< typename OUT >
00081 void dump(OUT &s) const
00082 {
00083 s.printf("%d(%d)", static_cast<int>(size()), static_cast<int>(shift()));
00084 }
00085 } __attribute__((packed));
00086
00094 class L4_EXPORT Pixel_info
00095 {
00096 private:
00097 Color_component _r, _g, _b, _a;
00098 unsigned char _bpp;
00099 public:
00105 Color_component const &r() const { return _r; }
00106
00111 Color_component const &g() const { return _g; }
00112
00117 Color_component const &b() const { return _b; }
00118
00123 Color_component const &a() const { return _a; }
00124
00131 Color_component const padding() const
00132 {
00133 unsigned char top_bit = cxx::max<unsigned char>(_r.size() + _r.shift(),
00134 _g.size() + _g.shift());
00135 top_bit = cxx::max<unsigned char>(top_bit, _b.size() + _b.shift());
00136 top_bit = cxx::max<unsigned char>(top_bit, _a.size() + _a.shift());
00137
00138 unsigned char bits = _bpp * 8;
00139
00140 if (top_bit < bits)
00141 return Color_component(bits - top_bit, top_bit);
00142
00143 return Color_component(0, 0);
00144 }
00145
00150 unsigned char bytes_per_pixel() const { return _bpp; }
00151
00156 unsigned char bits_per_pixel() const
00157 { return _r.size() + _g.size() + _b.size() + _a.size(); }
00158
00163 bool has_alpha() const { return _a.size() > 0; }
00164
00169 void r(Color_component const &c) { _r = c; }
00170
00175 void g(Color_component const &c) { _g = c; }
00176
00181 void b(Color_component const &c) { _b = c; }
00182
00187 void a(Color_component const &c) { _a = c; }
00188
00193 void bytes_per_pixel(unsigned char bpp) { _bpp = bpp; }
00194
00198 Pixel_info() : _bpp(0) {};
00199
00212 Pixel_info(unsigned char bpp, char r, char rs, char g, char gs,
00213 char b, char bs, char a = 0, char as = 0)
00214 : _r(r, rs), _g(g, gs), _b(b, bs), _a(a, as), _bpp(bpp)
00215 {}
00216
00223 template<typename VBI>
00224 explicit Pixel_info(VBI const *vbi)
00225 : _r(vbi->red_mask_size, vbi->red_field_position),
00226 _g(vbi->green_mask_size, vbi->green_field_position),
00227 _b(vbi->blue_mask_size, vbi->blue_field_position),
00228 _bpp((vbi->bits_per_pixel + 7) / 8)
00229 {}

```

```

00230
00236 bool operator == (Pixel_info const &o) const
00237 {
00238 return _r == o._r && _g == o._g && _b == o._b && _a == o._a && _bpp == o._bpp;
00239 }
00240
00245 template< typename OUT >
00246 void dump(OUT &s) const
00247 {
00248 s.printf("RGBA(%d):%d(%d):%d(%d):%d(%d)",
00249 static_cast<int>(bytes_per_pixel()),
00250 static_cast<int>(r().size()), static_cast<int>(r().shift()),
00251 static_cast<int>(g().size()), static_cast<int>(g().shift()),
00252 static_cast<int>(b().size()), static_cast<int>(b().shift()),
00253 static_cast<int>(a().size()), static_cast<int>(a().shift()));
00254 }
00255 };
00256
00257
00258 }}
00259
00260

```

## 17.426 goos

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 #pragma once
00010
00011 #include <l4/sys/capability>
00012 #include <l4/re/dataspace>
00013 #include <l4/re/video/colors>
00014 #include <l4/sys/cxx/ipc_iface>
00015
00016 namespace L4Re { namespace Video {
00017
00026 class L4_EXPORT Goos;
00027
00039 class L4_EXPORT View
00040 {
00041 private:
00042 friend class Goos;
00043
00044 L4::Cap<Goos> _goos;
00045 unsigned _view_idx;
00046
00047 View(l4_cap_idx_t goos, unsigned idx)
00048 : _goos(goos), _view_idx(_goos.is_valid() ? idx : ~0U) {}
00049
00050 unsigned view_index() const noexcept
00051 { return _goos.is_valid() ? _view_idx : ~0U; }
00052
00053 public:
00054 View() : _goos(L4::Cap<Goos>::Invalid), _view_idx(~0U) {}
00055
00059 enum Flags
00060 {
00061 F_none = 0x00,
00062 F_set_buffer = 0x01,
00063 F_set_buffer_offset = 0x02,
00064 F_set_bytes_per_line = 0x04,
00065 F_set_pixel = 0x08,
00066 F_set_position = 0x10,
00067 F_dyn_allocated = 0x20,
00068 F_set_background = 0x40,
00069 F_set_flags = 0x80,
00070
00072 F_fully_dynamic = F_set_buffer | F_set_buffer_offset | F_set_bytes_per_line
00073 | F_set_pixel | F_set_position | F_dyn_allocated,
00074 };
00075
00082 enum V_flags
00083 {
00084 F_above = 0x1000,
00085 F_flags_mask = 0xff000,
00086 };
00087

```

```

00091 struct Info
00092 {
00093 unsigned flags = 0;
00094 unsigned view_index = 0;
00095
00096 unsigned long xpos = 0;
00097 unsigned long ypos = 0;
00098 unsigned long width = 0;
00099 unsigned long height = 0;
00100 unsigned long buffer_offset = 0;
00101 unsigned long bytes_per_line = 0;
00102 Pixel_info pixel_info;
00103 unsigned buffer_index = 0;
00104
00106 bool has_static_buffer() const { return !(flags & F_set_buffer); }
00108 bool has_static_buffer_offset() const { return !(flags & F_set_buffer_offset); }
00109
00111 bool has_set_buffer() const { return flags & F_set_buffer; }
00113 bool has_set_buffer_offset() const { return flags & F_set_buffer_offset; }
00115 bool has_set_bytes_per_line() const { return flags & F_set_bytes_per_line; }
00117 bool has_set_pixel() const { return flags & F_set_pixel; }
00119 bool has_set_position() const { return flags & F_set_position; }
00120
00122 template< typename OUT >
00123 void dump(OUT &s) const
00124 {
00125 s.printf("View::Info:\n"
00126 " flags: %x\n"
00127 " size: %ldx%ld\n"
00128 " pos: %ldx%ld\n"
00129 " bytes_per_line: %ld\n"
00130 " buffer_offset: %lx\n"
00131 " ",
00132 flags, width, height, xpos, ypos,
00133 bytes_per_line, buffer_offset);
00134 pixel_info.dump(s);
00135 s.printf("\n");
00136 }
00137 };
00138
00146 int info(Info *info) const noexcept;
00147
00158 int set_info(Info const &info) const noexcept;
00159
00171 int set_viewport(int scr_x, int scr_y, int w, int h, unsigned long buf_offset) const noexcept;
00172
00182 int stack(View const &pivot, bool behind = true) const noexcept;
00183
00185 int push_top() const noexcept
00186 { return stack(View(), true); }
00187
00189 int push_bottom() const noexcept
00190 { return stack(View(), false); }
00191
00202 int refresh(int x, int y, int w, int h) const noexcept;
00203
00205 bool valid() const { return _goos.is_valid(); }
00206 };
00207
00208
00223 class L4_EXPORT Goos :
00224 public L4::Kobject_t<Goos, L4::Kobject, L4RE_PROTO_GOOS>
00225 {
00226 public:
00228 enum Flags
00229 {
00230 F_auto_refresh = 0x01,
00231 F_pointer = 0x02,
00232 F_dynamic_views = 0x04,
00233 F_dynamic_buffers = 0x08,
00234 };
00235
00237 struct Info
00238 {
00239 unsigned long width;
00240 unsigned long height;
00241 unsigned flags;
00242 unsigned num_static_views;
00243 unsigned num_static_buffers;
00244 Pixel_info pixel_info;
00245
00248 bool auto_refresh() const { return flags & F_auto_refresh; }
00250 bool has_pointer() const { return flags & F_pointer; }
00252 bool has_dynamic_views() const { return flags & F_dynamic_views; }
00254 bool has_dynamic_buffers() const { return flags & F_dynamic_buffers; }
00255
00256 Info()

```

```

00257 : width(0), height(0), flags(0), num_static_views(0),
00258 num_static_buffers(0) {}
00259 };
00260
00268 L4_INLINE_RPC(long, info, (Info *info));
00269
00278 L4_RPC(long, get_static_buffer, (unsigned idx,
00279 L4::Ipc::Out<L4::Cap<L4Re::Dataspace> > rbuf));
00280
00289 L4_RPC(long, create_buffer, (unsigned long size,
00290 L4::Ipc::Out<L4::Cap<L4Re::Dataspace> > rbuf));
00291
00299 L4_INLINE_RPC(long, delete_buffer, (unsigned idx));
00300
00301 // Use a wrapper for this RPC as we encapsulate the View
00302 L4_INLINE_RPC_NF(long, create_view, ());
00303
00312 int create_view(View *view, l4_utcb_t *utcb = l4_utcb()) const noexcept
00313 {
00314 long r = create_view_t::call(c(), utcb);
00315 if (r < 0)
00316 return r;
00317 *view = View(cap(), r);
00318 return r;
00319 }
00320
00321 // Use a wrapper as Views are encapsulated
00322 L4_INLINE_RPC_NF(long, delete_view, (unsigned index));
00323
00332 int delete_view(View const &v, l4_utcb_t *utcb = l4_utcb()) const noexcept
00333 {
00334 return delete_view_t::call(c(), v._view_idx, utcb);
00335 }
00336
00342 View view(unsigned index) const noexcept;
00343
00347 L4_INLINE_RPC(long, refresh, (int x, int y, int w, int h));
00348
00349 // those are used by the View
00350 L4_INLINE_RPC(long, view_info, (unsigned index, View::Info *info));
00351 L4_INLINE_RPC(long, set_view_info, (unsigned index, View::Info const &info));
00352 L4_INLINE_RPC(long, view_stack, (unsigned index, unsigned pivot, bool behind));
00353 L4_INLINE_RPC(long, view_refresh, (unsigned index, int x, int y, int w, int h));
00354
00355 typedef L4::Typeid::Rpc<
00356 info_t, get_static_buffer_t, create_buffer_t, create_view_t, delete_buffer_t,
00357 delete_view_t, view_info_t, set_view_info_t, view_stack_t, view_refresh_t,
00358 refresh_t
00359 > Rpcs;
00360 };
00361
00362 inline View
00363 Goos::view(unsigned index) const noexcept
00364 { return View(cap(), index); }
00365
00366 inline int
00367 View::info(Info *info) const noexcept
00368 { return _goos->view_info(_view_idx, info); }
00369
00370 inline int
00371 View::set_info(Info const &info) const noexcept
00372 { return _goos->set_view_info(_view_idx, info); }
00373
00374 inline int
00375 View::stack(View const &pivot, bool behind) const noexcept
00376 { return _goos->view_stack(_view_idx, pivot._view_idx, behind); }
00377
00378 inline int
00379 View::refresh(int x, int y, int w, int h) const noexcept
00380 { return _goos->view_refresh(_view_idx, x, y, w, h); }
00381
00382 inline int
00383 View::set_viewport(int scr_x, int scr_y, int w, int h,
00384 unsigned long buf_offset) const noexcept
00385 {
00386 Info i;
00387 i.flags = F_set_buffer_offset | F_set_position;
00388 i.buffer_offset = buf_offset;
00389 i.buffer_index = 0;
00390 i.view_index = 0;
00391 i.bytes_per_line = 0;
00392 i.pixel_info = Pixel_info();
00393 i.xpos = scr_x;
00394 i.ypos = scr_y;
00395 i.width = w;
00396 i.height = h;
00397 return set_info(i);

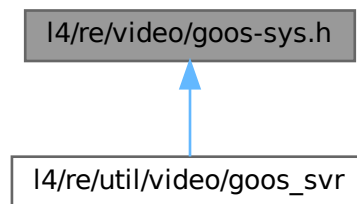
```

```
00398 }
00399
00400 }}
```

## 17.427 l4/re/video/goos-sys.h File Reference

Goos protocol definition.

This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [L4Re](#)  
*L4Re C++ Interfaces.*

### Enumerations

- enum [L4Re::Video::Goos\\_::Opcodes](#)  
*Frame buffer communication-protocol opcodes.*

### 17.427.1 Detailed Description

Goos protocol definition.

Definition in file [goos-sys.h](#).

## 17.428 goos-sys.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00009 * Björn Döbel <doebel@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014
00015 namespace L4Re { namespace Video {
00016 namespace Goos_
00017 {
00023 enum Opcodes
00024 {
00025 Info, Get_buffer, Create_buffer, Create_view,
00026 Delete_buffer, Delete_view,
00027 View_info, View_set_info, View_stack, View_refresh,
00028 Screen_refresh
00029 };
00030 };
00031 }
```

## 17.429 view

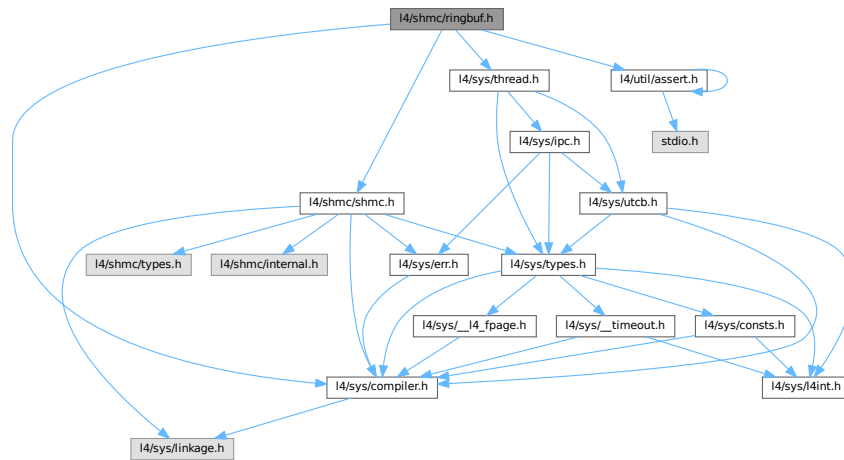
```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 #pragma once
00010
00011 #include <l4/re/video/goos>
00012
```

## 17.430 l4/shmc/ringbuf.h File Reference

```
#include <l4/shmc/shmc.h>
#include <l4/util/assert.h>
#include <l4/sys/compiler.h>
#include <l4/sys/thread.h>
```



Include dependency graph for ringbuf.h:



## Data Structures

- struct [l4shmc\\_ringbuf\\_head\\_t](#)  
*Head field of a ring buffer.*
- struct [l4shmc\\_ringbuf\\_t](#)  
*Ring buffer.*

## Macros

- #define [L4SHMC\\_RINGBUF\\_HEAD](#)(ringbuf)  
*Get ring buffer head pointer.*
- #define [L4SHMC\\_RINGBUF\\_DATA](#)(ringbuf)  
*Get ring buffer data pointer.*
- #define [L4SHMC\\_RINGBUF\\_DATA\\_SIZE](#)(ringbuf)  
*Get size of data area.*

## Functions

- int [l4shmc\\_rb\\_init\\_buffer](#) ([l4shmc\\_ringbuf\\_t](#) \*buf, [l4shmc\\_area\\_t](#) \*area, char const \*chunk\_name, char const \*signal\_name, unsigned size)  
*Initialize a ring buffer by creating an SHMC chunk and the corresponding signals.*
- void [l4shmc\\_rb\\_deinit\\_buffer](#) ([l4shmc\\_ringbuf\\_t](#) \*buf)  
*De-init a ring buffer.*
- int [l4shmc\\_rb\\_attach\\_sender](#) ([l4shmc\\_ringbuf\\_t](#) \*buf, char const \*signal\_name, [l4\\_cap\\_idx\\_t](#) owner)  
*Attach to sender signal of a ring buffer.*
- char \* [l4shmc\\_rb\\_sender\\_alloc\\_packet](#) ([l4shmc\\_ringbuf\\_head\\_t](#) \*head, unsigned psize)  
*Allocate a packet of a given size within the ring buffer.*
- void [l4shmc\\_rb\\_sender\\_put\\_data](#) ([l4shmc\\_ringbuf\\_t](#) \*buf, char \*addr, char \*data, unsigned dsize)  
*Copy data into a previously allocated packet.*
- int [l4shmc\\_rb\\_sender\\_next\\_copy\\_in](#) ([l4shmc\\_ringbuf\\_t](#) \*buf, char \*data, unsigned size, int block\_if\_↵ necessary)

- Copy in packet from an external data source.*
- void [l4shmc\\_rb\\_sender\\_commit\\_packet](#) ([l4shmc\\_ringbuf\\_t](#) \*buf)
  - Tell the consumer that new data is available.*
- int [l4shmc\\_rb\\_init\\_receiver](#) ([l4shmc\\_ringbuf\\_t](#) \*buf, [l4shmc\\_area\\_t](#) \*area, char const \*chunk\_name, char const \*signal\_name)
  - Initialize receive buffer.*
- void [l4shmc\\_rb\\_attach\\_receiver](#) ([l4shmc\\_ringbuf\\_t](#) \*buf, [l4\\_cap\\_idx\\_t](#) owner)
  - Attach to receiver signal of a ring buffer.*
- int [l4shmc\\_rb\\_receiver\\_wait\\_for\\_data](#) ([l4shmc\\_ringbuf\\_t](#) \*buf, int blocking)
  - Check if (and optionally block until) new data is ready.*
- int [l4shmc\\_rb\\_receiver\\_copy\\_out](#) ([l4shmc\\_ringbuf\\_head\\_t](#) \*head, char \*target, unsigned \*tsize)
  - Copy data out of the buffer.*
- void [l4shmc\\_rb\\_receiver\\_notify\\_done](#) ([l4shmc\\_ringbuf\\_t](#) \*buf)
  - Notify producer that space is available.*
- int [l4shmc\\_rb\\_receiver\\_read\\_next\\_size](#) ([l4shmc\\_ringbuf\\_head\\_t](#) \*head)
  - Have a look at the ring buffer and see which size the next packet to be read has.*

## 17.430.1 Function Documentation

### 17.430.1.1 [l4shmc\\_rb\\_attach\\_receiver\(\)](#)

```
void l4shmc_rb_attach_receiver (
 l4shmc_ringbuf_t * buf,
 l4_cap_idx_t owner)
```

Attach to receiver signal of a ring buffer.

Attach owner to the receiver-side signal of a ring buffer, which is triggered whenever new data has been produced.

This is split from initialization, because you may not know the owner cap when initializing the buffer.

#### Parameters

|              |                               |
|--------------|-------------------------------|
| <i>buf</i>   | pointer to ring buffer struct |
| <i>owner</i> | owner thread                  |

References [L4\\_CV](#).

### 17.430.1.2 [l4shmc\\_rb\\_attach\\_sender\(\)](#)

```
int l4shmc_rb_attach_sender (
 l4shmc_ringbuf_t * buf,
 char const * signal_name,
 l4_cap_idx_t owner)
```

Attach to sender signal of a ring buffer.

Attach owner to the sender-side signal of a ring buffer, which is triggered whenever new space has been freed in the buffer for the sender to write to.

This is split from initialization, because you may not know the owner cap when initializing the buffer.

#### Parameters

|                    |                               |
|--------------------|-------------------------------|
| <i>buf</i>         | pointer to ring buffer struct |
| <i>signal_name</i> | signal base name              |
| <i>owner</i>       | owner thread                  |

#### Returns

0 on success, error otherwise

References [L4\\_CV](#).

#### 17.430.1.3 l4shmc\_rb\_deinit\_buffer()

```
void l4shmc_rb_deinit_buffer (
 l4shmc_ringbuf_t * buf)
```

De-init a ring buffer.

#### Parameters

|            |                               |
|------------|-------------------------------|
| <i>buf</i> | pointer to ring buffer struct |
|------------|-------------------------------|

References [L4\\_CV](#).

#### 17.430.1.4 l4shmc\_rb\_init\_buffer()

```
int l4shmc_rb_init_buffer (
 l4shmc_ringbuf_t * buf,
 l4shmc_area_t * area,
 char const * chunk_name,
 char const * signal_name,
 unsigned size)
```

Initialize a ring buffer by creating an SHMC chunk and the corresponding signals.

This needs to be done by one of the participating parties when setting up communication channel.

#### Precondition

area has been attached using [l4shmc\\_attach\(\)](#).

#### Parameters

|                   |                                      |
|-------------------|--------------------------------------|
| <i>buf</i>        | pointer to ring buffer struct        |
| <i>area</i>       | pointer to SHMC area                 |
| <i>chunk_name</i> | name of SHMC chunk to create in area |

|                    |                                      |
|--------------------|--------------------------------------|
| <i>signal_name</i> | base name for SHMC signals to create |
| <i>size</i>        | chunk size                           |

**Returns**

0 on success, error otherwise

References [L4\\_CV](#).

**17.430.1.5 l4shmc\_rb\_init\_receiver()**

```
int l4shmc_rb_init_receiver (
 l4shmc_ringbuf_t * buf,
 l4shmc_area_t * area,
 char const * chunk_name,
 char const * signal_name)
```

Initialize receive buffer.

Initialize the receiver-side of a ring buffer. This requires the underlying SHMC chunk and the corresponding signals to be valid already (read: to be initialized by the sender).

**Precondition**

chunk & signals have been created and initialized by the sender side

**Parameters**

|                    |                                      |
|--------------------|--------------------------------------|
| <i>buf</i>         | pointer to ring buffer struct        |
| <i>area</i>        | pointer to SHMC area                 |
| <i>chunk_name</i>  | name of SHMC chunk to create in area |
| <i>signal_name</i> | base name for SHMC signals to create |

**Returns**

0 on success, error otherwise

References [L4\\_CV](#).

**17.430.1.6 l4shmc\_rb\_receiver\_copy\_out()**

```
int l4shmc_rb_receiver_copy_out (
 l4shmc_ringbuf_head_t * head,
 char * target,
 unsigned * tsize)
```

Copy data out of the buffer.

**Parameters**

|                |               |                                                                                  |
|----------------|---------------|----------------------------------------------------------------------------------|
|                | <i>head</i>   | ring buffer head pointer                                                         |
|                | <i>target</i> | valid target buffer                                                              |
| <i>in, out</i> | <i>size</i>   | size of target buffer (must be $\geq$ packet size!); contains the real data size |

**Returns**

0 on success, negative error otherwise

References [L4\\_CV](#).

**17.430.1.7 l4shmc\_rb\_receiver\_notify\_done()**

```
void l4shmc_rb_receiver_notify_done (
 l4shmc_ringbuf_t * buf)
```

Notify producer that space is available.

**Parameters**

|            |                               |
|------------|-------------------------------|
| <i>buf</i> | pointer to ring buffer struct |
|------------|-------------------------------|

References [L4\\_CV](#).

**17.430.1.8 l4shmc\_rb\_receiver\_read\_next\_size()**

```
int l4shmc_rb_receiver_read_next_size (
 l4shmc_ringbuf_head_t * head)
```

Have a look at the ring buffer and see which size the next packet to be read has.

Does not modify anything.

**Returns**

size of next buffer or -1 if no data available

References [L4\\_CV](#), and [L4\\_END\\_DECLS](#).

**17.430.1.9 l4shmc\_rb\_receiver\_wait\_for\_data()**

```
int l4shmc_rb_receiver_wait_for_data (
 l4shmc_ringbuf_t * buf,
 int blocking)
```

Check if (and optionally block until) new data is ready.

**Parameters**

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>buf</i>      | pointer to ring buffer struct              |
| <i>blocking</i> | block if data is not available immediately |

Returns immediately, if data is available.

#### Returns

0 success, data available, != 0 otherwise

References [L4\\_CV](#).

#### 17.430.1.10 l4shmc\_rb\_sender\_alloc\_packet()

```
char * l4shmc_rb_sender_alloc_packet (
 l4shmc_ringbuf_head_t * head,
 unsigned psize)
```

Allocate a packet of a given size within the ring buffer.

This packet may wrap around at the end of the buffer. Users need to be aware of that.

#### Parameters

|              |                          |
|--------------|--------------------------|
| <i>head</i>  | ring buffer head pointer |
| <i>psize</i> | packet size              |

#### Returns

valid address on success

#### Return values

|             |                               |
|-------------|-------------------------------|
| <i>NULL</i> | if not enough space available |
|-------------|-------------------------------|

References [L4\\_CV](#).

#### 17.430.1.11 l4shmc\_rb\_sender\_commit\_packet()

```
void l4shmc_rb_sender_commit_packet (
 l4shmc_ringbuf_t * buf)
```

Tell the consumer that new data is available.

#### Parameters

---

|            |                               |
|------------|-------------------------------|
| <i>buf</i> | pointer to ring buffer struct |
|------------|-------------------------------|

References [L4\\_CV](#).

#### 17.430.1.12 l4shmc\_rb\_sender\_next\_copy\_in()

```
int l4shmc_rb_sender_next_copy_in (
 l4shmc_ringbuf_t * buf,
 char * data,
 unsigned size,
 int block_if_necessary)
```

Copy in packet from an external data source.

This is the function you'll want to use. Just pass it a buffer pointer and let the lib do the work.

##### Parameters

|                           |                                      |
|---------------------------|--------------------------------------|
| <i>buf</i>                | pointer to ring buffer struct        |
| <i>data</i>               | valid buffer                         |
| <i>size</i>               | data size                            |
| <i>block_if_necessary</i> | bool: block if buffer currently full |

##### Return values

|                   |                                          |
|-------------------|------------------------------------------|
| <i>0</i>          | on success                               |
| <i>-L4_ENOMEM</i> | if block == false and no space available |

References [L4\\_CV](#).

#### 17.430.1.13 l4shmc\_rb\_sender\_put\_data()

```
void l4shmc_rb_sender_put_data (
 l4shmc_ringbuf_t * buf,
 char * addr,
 char * data,
 unsigned dsize)
```

Copy data into a previously allocated packet.

This function is wrap-around aware.

##### Parameters

|            |                               |
|------------|-------------------------------|
| <i>buf</i> | pointer to ring buffer struct |
|------------|-------------------------------|

|              |                                                                |
|--------------|----------------------------------------------------------------|
| <i>addr</i>  | valid destination (allocate with <code>alloc_packet()</code> ) |
| <i>data</i>  | data source                                                    |
| <i>dsize</i> | data size                                                      |

References [L4\\_CV](#).

## 17.431 ringbuf.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * (c) 2010 Björn Döbel <doebel@os.inf.tu-dresden.de>
00003 * economic rights: Technische Universität Dresden (Germany)
00004 * This file is part of TUD:OS and distributed under the terms of the
00005 * GNU Lesser General Public License 2.1.
00006 * Please see the COPYING-LGPL-2.1 file for details.
00007 */
00008
00012 #pragma once
00013
00014 #include <l4/shmc/shmc.h>
00015 #include <l4/util/assert.h>
00016 #include <l4/sys/compiler.h>
00017 #include <l4/sys/thread.h>
00018
00019 L4_BEGIN_DECLS
00020
00035
00046
00047 /*
00048 * Turn on ringbuf poisoning. This will add magic values to the ringbuf
00049 * header as well as each packet header and check that these values are
00050 * valid all the time.
00051 */
00052 #define L4SHMC_RINGBUF_POISONING 1
00053
00059 typedef struct
00060 {
00061 volatile l4_uint32_t lock;
00062 unsigned data_size;
00063 #if L4SHMC_RINGBUF_POISONING
00064 char magic1;
00065 #endif
00066 unsigned next_read;
00067 unsigned next_write;
00068 #if L4SHMC_RINGBUF_POISONING
00069 char magic2;
00070 #endif
00071 unsigned bytes_filled;
00072 unsigned sender_waits;
00073 #if L4SHMC_RINGBUF_POISONING
00074 char magic3;
00075 #endif
00076 char data[];
00077 } l4shmc_ringbuf_head_t;
00078
00079
00085 typedef struct
00086 {
00087 l4shmc_area_t *_area;
00088 l4_cap_idx_t _owner;
00089 l4shmc_chunk_t _chunk;
00090 unsigned _size;
00091 char *_chunkname;
00092 char *_signame;
00093 l4shmc_ringbuf_head_t *_addr;
00094 l4shmc_signal_t _signal_full;
00095 l4shmc_signal_t _signal_empty;
00096 } l4shmc_ringbuf_t;
00097
00098
00105 #define L4SHMC_RINGBUF_HEAD(ringbuf) ((l4shmc_ringbuf_head_t*) ((ringbuf)->_addr))
00106
00107
00114 #define L4SHMC_RINGBUF_DATA(ringbuf) (L4SHMC_RINGBUF_HEAD(ringbuf)->data)
00115

```



```

00116
00123 #define L4SHMC_RINGBUF_DATA_SIZE(ringbuf) ((ringbuf)->_size - sizeof(l4shmc_ringbuf_head_t))
00124
00125 enum lock_content
00126 {
00127 lock_cont_min = 4,
00128 locked = 5,
00129 unlocked = 6,
00130 lock_cont_max = 7,
00131 };
00132
00133 static L4_CV inline void l4shmc_rb_lock(l4shmc_ringbuf_head_t *head)
00134 {
00135 ASSERT_NOT_NULL(head);
00136 ASSERT_ASSERT(head->lock > lock_cont_min);
00137 ASSERT_ASSERT(head->lock < lock_cont_max);
00138
00139 while (!l4util_cmpxchg32(&head->lock, unlocked, locked))
00140 l4_thread_yield();
00141 }
00142
00143
00144 static L4_CV inline void l4shmc_rb_unlock(l4shmc_ringbuf_head_t *head)
00145 {
00146 ASSERT_NOT_NULL(head);
00147 ASSERT_ASSERT(head->lock > lock_cont_min);
00148 ASSERT_ASSERT(head->lock < lock_cont_max);
00149
00150 head->lock = unlocked;
00151 }
00152
00153 /*****
00154 * Initialization *
00155 *****/
00156
00173 L4_CV int l4shmc_rb_init_buffer(l4shmc_ringbuf_t *buf, l4shmc_area_t *area,
00174 char const *chunk_name,
00175 char const *signal_name, unsigned size);
00176
00182 L4_CV void l4shmc_rb_deinit_buffer(l4shmc_ringbuf_t *buf);
00183
00184
00185
00186 /*****
00187 * RINGBUF SENDER *
00188 *****/
00189
00206 L4_CV int l4shmc_rb_attach_sender(l4shmc_ringbuf_t *buf, char const *signal_name,
00207 l4_cap_idx_t owner);
00208
00209
00222 L4_CV char *l4shmc_rb_sender_alloc_packet(l4shmc_ringbuf_head_t *head,
00223 unsigned psize);
00224
00225
00236 L4_CV void l4shmc_rb_sender_put_data(l4shmc_ringbuf_t *buf, char *addr,
00237 char *data, unsigned dsize);
00238
00239
00254 L4_CV int l4shmc_rb_sender_next_copy_in(l4shmc_ringbuf_t *buf, char *data,
00255 unsigned size, int block_if_necessary);
00256
00257
00263 L4_CV void l4shmc_rb_sender_commit_packet(l4shmc_ringbuf_t *buf);
00264
00265
00266 /*****
00267 * RINGBUF RECEIVER *
00268 *****/
00269
00286 L4_CV int l4shmc_rb_init_receiver(l4shmc_ringbuf_t *buf, l4shmc_area_t *area,
00287 char const *chunk_name,
00288 char const *signal_name);
00289
00290
00303 L4_CV void l4shmc_rb_attach_receiver(l4shmc_ringbuf_t *buf, l4_cap_idx_t owner);
00304
00305
00316 L4_CV int l4shmc_rb_receiver_wait_for_data(l4shmc_ringbuf_t *buf, int blocking);
00317
00318
00328 L4_CV int l4shmc_rb_receiver_copy_out(l4shmc_ringbuf_head_t *head, char *target,
00329 unsigned *tsize);
00330
00331
00337 L4_CV void l4shmc_rb_receiver_notify_done(l4shmc_ringbuf_t *buf);
00338

```

```

00339
00346 L4_CV int l4shmc_rb_receiver_read_next_size(l4shmc_ringbuf_head_t *head);
00347
00348 L4_END_DECLS

```

## 17.432 l4/shmc/shmc.h File Reference

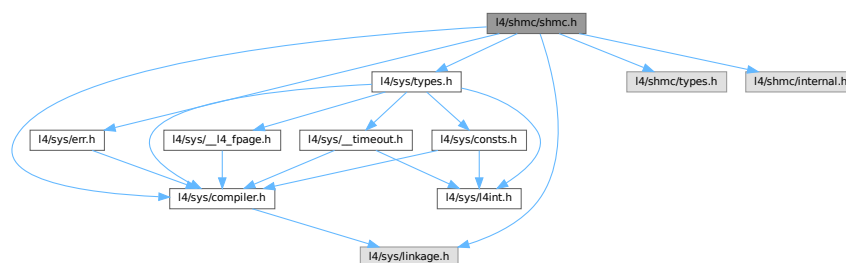
Shared memory library header file.

```

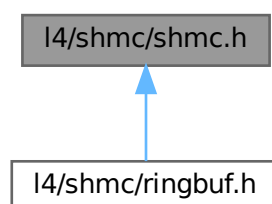
#include <l4/sys/compiler.h>
#include <l4/sys/linkage.h>
#include <l4/sys/types.h>
#include <l4/sys/err.h>
#include <l4/shmc/types.h>
#include <l4/shmc/internal.h>

```

Include dependency graph for shmc.h:



This graph shows which files directly or indirectly include this file:



### Functions

- [L4\\_BEGIN\\_DECLS](#) long [l4shmc\\_create](#) (char const \*shmc\_name)  
Create a shared memory area.
- long [l4shmc\\_attach](#) (char const \*shmc\_name, l4shmc\_area\_t \*shmarea)  
Attach to a shared memory area.
- long [l4shmc\\_get\\_client\\_nr](#) (l4shmc\_area\_t const \*shmarea)

- Determine the client number of the shared memory region.*

  - long [l4shmc\\_mark\\_client\\_initialized](#) (l4shmc\_area\_t \*shmarea)

*Mark this shared memory client as 'initialized'.*
- long [l4shmc\\_get\\_initialized\\_clients](#) (l4shmc\_area\_t \*shmarea, l4\_umword\_t \*bitmask)

*Fetch the `_clients_init_done` bitmask of the shared memory area.*
- long [l4shmc\\_add\\_chunk](#) (l4shmc\_area\_t \*shmarea, char const \*chunk\_name, l4\_umword\_t chunk\_capacity, l4shmc\_chunk\_t \*chunk)

*Add a chunk in the shared memory area.*
- long [l4shmc\\_add\\_signal](#) (l4shmc\_area\_t \*shmarea, char const \*signal\_name, l4shmc\_signal\_t \*signal)

*Add a signal for the shared memory area.*
- long [l4shmc\\_trigger](#) (l4shmc\_signal\_t \*signal)

*Trigger a signal.*
- long [l4shmc\\_chunk\\_try\\_to\\_take](#) (l4shmc\_chunk\_t \*chunk)

*Try to mark chunk busy.*
- long [l4shmc\\_chunk\\_try\\_to\\_take\\_for\\_writing](#) (l4shmc\_chunk\_t \*chunk)

*Try to mark chunk busy writing.*
- long [l4shmc\\_chunk\\_try\\_to\\_take\\_for\\_overwriting](#) (l4shmc\_chunk\_t \*chunk)

*Try to mark the chunk busy writing after it was ready for reading.*
- long [l4shmc\\_chunk\\_try\\_to\\_take\\_for\\_reading](#) (l4shmc\_chunk\_t \*chunk)

*Try to mark chunk busy reading.*
- long [l4shmc\\_chunk\\_ready](#) (l4shmc\_chunk\_t \*chunk, l4\_umword\_t size)

*Mark chunk as filled (ready).*
- long [l4shmc\\_chunk\\_ready\\_sig](#) (l4shmc\_chunk\_t \*chunk, l4\_umword\_t size)

*Mark chunk as filled (ready) and signal consumer.*
- long [l4shmc\\_get\\_chunk](#) (l4shmc\_area\_t \*shmarea, char const \*chunk\_name, l4shmc\_chunk\_t \*chunk)

*Get chunk out of shared memory area.*
- long [l4shmc\\_get\\_chunk\\_to](#) (l4shmc\_area\_t \*shmarea, char const \*chunk\_name, l4\_umword\_t timeout\_ms, l4shmc\_chunk\_t \*chunk)

*Get chunk out of shared memory area, with timeout.*
- long [l4shmc\\_iterate\\_chunk](#) (l4shmc\_area\_t const \*shmarea, char const \*\*chunk\_name, long offs)

*Iterate over names of all existing chunks.*
- long [l4shmc\\_attach\\_signal](#) (l4shmc\_area\_t \*shmarea, char const \*signal\_name, l4\_cap\_idx\_t thread, l4shmc\_signal\_t \*signal)

*Attach to signal.*
- long [l4shmc\\_get\\_signal](#) (l4shmc\_area\_t \*shmarea, char const \*signal\_name, l4shmc\_signal\_t \*signal)

*Get signal object from the shared memory area.*
- long [l4shmc\\_enable\\_signal](#) (l4shmc\_signal\_t \*signal)

*Enable a signal.*
- long [l4shmc\\_enable\\_chunk](#) (l4shmc\_chunk\_t \*chunk)

*Enable a signal connected with a chunk.*
- long [l4shmc\\_wait\\_any](#) (l4shmc\_signal\_t \*\*retsignal)

*Wait on any signal.*
- long [l4shmc\\_wait\\_any\\_try](#) (l4shmc\_signal\_t \*\*retsignal)

*Check whether any waited signal has an event pending.*
- long [l4shmc\\_wait\\_any\\_to](#) (l4\_timeout\_t timeout, l4shmc\_signal\_t \*\*retsignal)

*Wait for any signal with timeout.*
- long [l4shmc\\_wait\\_signal](#) (l4shmc\_signal\_t \*signal)

*Wait on a specific signal.*
- long [l4shmc\\_wait\\_signal\\_to](#) (l4shmc\_signal\_t \*signal, l4\_timeout\_t timeout)

*Wait on a specific signal, with timeout.*
- long [l4shmc\\_wait\\_signal\\_try](#) (l4shmc\_signal\_t \*signal)

- Check whether a specific signal has an event pending.*

  - long [l4shmc\\_wait\\_chunk](#) (l4shmc\_chunk\_t \*chunk)

*Wait on a specific chunk.*
- long [l4shmc\\_wait\\_chunk\\_to](#) (l4shmc\_chunk\_t \*chunk, [l4\\_timeout\\_t](#) timeout)

*Check whether a specific chunk has an event pending, with timeout.*
- long [l4shmc\\_wait\\_chunk\\_try](#) (l4shmc\_chunk\_t \*chunk)

*Check whether a specific chunk has an event pending.*
- long [l4shmc\\_chunk\\_consumed](#) (l4shmc\_chunk\_t \*chunk)

*Mark a chunk as free.*
- long [l4shmc\\_connect\\_chunk\\_signal](#) (l4shmc\_chunk\_t \*chunk, l4shmc\_signal\_t \*signal)

*Connect a signal with a chunk.*
- long [l4shmc\\_is\\_chunk\\_ready](#) (l4shmc\_chunk\_t const \*chunk)

*Check whether data is available.*
- long [l4shmc\\_is\\_chunk\\_clear](#) (l4shmc\_chunk\_t const \*chunk)

*Check whether chunk is free.*
- void \* [l4shmc\\_chunk\\_ptr](#) (l4shmc\_chunk\_t const \*chunk)

*Get data pointer to chunk.*
- long [l4shmc\\_chunk\\_size](#) (l4shmc\_chunk\_t const \*chunk)

*Get current size of a chunk.*
- long [l4shmc\\_chunk\\_capacity](#) (l4shmc\_chunk\_t const \*chunk)

*Get capacity of a chunk.*
- l4shmc\_signal\_t \* [l4shmc\\_chunk\\_signal](#) (l4shmc\_chunk\_t const \*chunk)

*Get the registered signal of a chunk.*
- [l4\\_cap\\_idx\\_t](#) [l4shmc\\_signal\\_cap](#) (l4shmc\_signal\_t const \*signal)

*Get the signal capability of a signal.*
- long [l4shmc\\_check\\_magic](#) (l4shmc\_chunk\_t const \*chunk)

*Check magic value of a chunk.*
- long [l4shmc\\_area\\_size](#) (l4shmc\_area\_t const \*shmarea)

*Get size of shared memory area.*
- long [l4shmc\\_area\\_size\\_free](#) (l4shmc\_area\_t const \*shmarea)

*Get free size of shared memory area.*
- long [l4shmc\\_area\\_overhead](#) (void)

*Get memory overhead per area that is not available for chunks.*
- long [l4shmc\\_chunk\\_overhead](#) (void)

*Get memory overhead required in addition to the chunk capacity for adding one chunk.*

### 17.432.1 Detailed Description

Shared memory library header file.

Definition in file [shmc.h](#).

## 17.433 shmc.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Björn Döbel <doebel@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 * This file is part of TUD:OS and distributed under the terms of the
00010 * GNU Lesser General Public License 2.1.
00011 * Please see the COPYING-LGPL-2.1 file for details.
00012 */
00013 #pragma once
00014
00015 #include <l4/sys/compiler.h>
00016 #include <l4/sys/linkage.h>
00017 #include <l4/sys/types.h>
00018 #include <l4/sys/err.h>
00019
00069
00070 #define __INCLUDED_FROM_L4SHMC_H__
00071 #include <l4/shmc/types.h>
00072
00073 L4_BEGIN_DECLS
00074
00087 L4_CV long
00088 l4shmc_create(char const *shmc_name);
00089
00111 L4_CV long
00112 l4shmc_attach(char const *shmc_name, l4shmc_area_t *shmarea);
00113
00122 L4_CV long
00123 l4shmc_get_client_nr(l4shmc_area_t const *shmarea);
00124
00137 L4_CV long
00138 l4shmc_mark_client_initialized(l4shmc_area_t *shmarea);
00139
00152 L4_CV long
00153 l4shmc_get_initialized_clients(l4shmc_area_t *shmarea, l4_umword_t *bitmask);
00154
00167 L4_CV long
00168 l4shmc_add_chunk(l4shmc_area_t *shmarea, char const *chunk_name,
00169 l4_umword_t chunk_capacity, l4shmc_chunk_t *chunk);
00170
00182 L4_CV long
00183 l4shmc_add_signal(l4shmc_area_t *shmarea, char const *signal_name,
00184 l4shmc_signal_t *signal);
00185
00195 L4_CV L4_INLINE long
00196 l4shmc_trigger(l4shmc_signal_t *signal);
00197
00207 L4_CV L4_INLINE long
00208 l4shmc_chunk_try_to_take(l4shmc_chunk_t *chunk);
00209
00221 L4_CV L4_INLINE long
00222 l4shmc_chunk_try_to_take_for_writing(l4shmc_chunk_t *chunk);
00223
00238 L4_CV L4_INLINE long
00239 l4shmc_chunk_try_to_take_for_overwriting(l4shmc_chunk_t *chunk);
00240
00250 L4_CV L4_INLINE long
00251 l4shmc_chunk_try_to_take_for_reading(l4shmc_chunk_t *chunk);
00252
00263 L4_CV L4_INLINE long
00264 l4shmc_chunk_ready(l4shmc_chunk_t *chunk, l4_umword_t size);
00265
00276 L4_CV L4_INLINE long
00277 l4shmc_chunk_ready_sig(l4shmc_chunk_t *chunk, l4_umword_t size);
00278
00290 L4_CV L4_INLINE long
00291 l4shmc_get_chunk(l4shmc_area_t *shmarea, char const *chunk_name,
00292 l4shmc_chunk_t *chunk);
00293
00307 L4_CV long
00308 l4shmc_get_chunk_to(l4shmc_area_t *shmarea, char const *chunk_name,
00309 l4_umword_t timeout_ms, l4shmc_chunk_t *chunk);
00310
00324 L4_CV long
00325 l4shmc_iterate_chunk(l4shmc_area_t const *shmarea, char const **chunk_name,
00326 long offs);
00327
00340 L4_CV long
00341 l4shmc_attach_signal(l4shmc_area_t *shmarea, char const *signal_name,
00342 l4_cap_idx_t thread, l4shmc_signal_t *signal);
00343

```

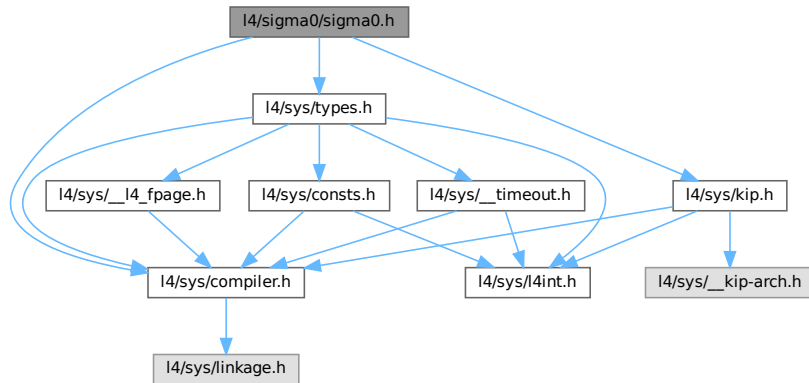
```
00344
00356 L4_CV long
00357 l4shmc_get_signal(l4shmc_area_t *shmarea, char const *signal_name,
00358 l4shmc_signal_t *signal);
00359
00373 L4_CV long
00374 l4shmc_enable_signal(l4shmc_signal_t *signal);
00375
00389 L4_CV long
00390 l4shmc_enable_chunk(l4shmc_chunk_t *chunk);
00391
00401 L4_CV L4_INLINE long
00402 l4shmc_wait_any(l4shmc_signal_t **retsignal);
00403
00417 L4_CV L4_INLINE long
00418 l4shmc_wait_any_try(l4shmc_signal_t **retsignal);
00419
00434 L4_CV long
00435 l4shmc_wait_any_to(l4_timeout_t timeout, l4shmc_signal_t **retsignal);
00436
00446 L4_CV L4_INLINE long
00447 l4shmc_wait_signal(l4shmc_signal_t *signal);
00448
00459 L4_CV long
00460 l4shmc_wait_signal_to(l4shmc_signal_t *signal, l4_timeout_t timeout);
00461
00475 L4_CV L4_INLINE long
00476 l4shmc_wait_signal_try(l4shmc_signal_t *signal);
00477
00487 L4_CV L4_INLINE long
00488 l4shmc_wait_chunk(l4shmc_chunk_t *chunk);
00489
00504 L4_CV long
00505 l4shmc_wait_chunk_to(l4shmc_chunk_t *chunk, l4_timeout_t timeout);
00506
00520 L4_CV L4_INLINE long
00521 l4shmc_wait_chunk_try(l4shmc_chunk_t *chunk);
00522
00532 L4_CV L4_INLINE long
00533 l4shmc_chunk_consumed(l4shmc_chunk_t *chunk);
00534
00545 L4_CV long
00546 l4shmc_connect_chunk_signal(l4shmc_chunk_t *chunk, l4shmc_signal_t *signal);
00547
00557 L4_CV L4_INLINE long
00558 l4shmc_is_chunk_ready(l4shmc_chunk_t const *chunk);
00559
00569 L4_CV L4_INLINE long
00570 l4shmc_is_chunk_clear(l4shmc_chunk_t const *chunk);
00571
00580 L4_CV L4_INLINE void *
00581 l4shmc_chunk_ptr(l4shmc_chunk_t const *chunk);
00582
00591 L4_CV L4_INLINE long
00592 l4shmc_chunk_size(l4shmc_chunk_t const *chunk);
00593
00602 L4_CV L4_INLINE long
00603 l4shmc_chunk_capacity(l4shmc_chunk_t const *chunk);
00604
00614 L4_CV L4_INLINE l4shmc_signal_t *
00615 l4shmc_chunk_signal(l4shmc_chunk_t const *chunk);
00616
00625 L4_CV L4_INLINE l4_cap_idx_t
00626 l4shmc_signal_cap(l4shmc_signal_t const *signal);
00627
00637 L4_CV L4_INLINE long
00638 l4shmc_check_magic(l4shmc_chunk_t const *chunk);
00639
00649 L4_CV long
00650 l4shmc_area_size(l4shmc_area_t const *shmarea);
00651
00661 L4_CV long
00662 l4shmc_area_size_free(l4shmc_area_t const *shmarea);
00663
00670 L4_CV long
00671 l4shmc_area_overhead(void);
00672
00680 L4_CV long
00681 l4shmc_chunk_overhead(void);
00682
00683 #include <l4/shmc/internal.h>
00684
00685 L4_END_DECLS
```

## 17.434 l4/sigma0/sigma0.h File Reference

Sigma0 interface.

```
#include <l4/sys/compiler.h>
#include <l4/sys/types.h>
#include <l4/sys/kip.h>
```

Include dependency graph for sigma0.h:



### Macros

- **#define SIGMA0\_REQ\_MAGIC** ~0xFFUL  
*Request magic.*
- **#define SIGMA0\_REQ\_MASK** ~0xFFUL  
*Request mask.*
- **#define SIGMA0\_REQ\_ID\_MASK** 0xF0  
*ID mask.*
- **#define SIGMA0\_REQ\_ID\_FPAGE\_RAM** 0x60  
*RAM.*
- **#define SIGMA0\_REQ\_ID\_FPAGE\_IOMEM** 0x70  
*I/O memory.*
- **#define SIGMA0\_REQ\_ID\_FPAGE\_IOMEM\_CACHED** 0x80  
*Cached I/O memory.*
- **#define SIGMA0\_REQ\_ID\_FPAGE\_ANY** 0x90  
*Any.*
- **#define SIGMA0\_REQ\_ID\_KIP** 0xA0  
*KIP.*
- **#define SIGMA0\_REQ\_ID\_DEBUG\_DUMP** 0xC0  
*Debug dump.*
- **#define SIGMA0\_IS\_MAGIC\_REQ**(d1)  
*Check if magic.*
- **#define SIGMA0\_REQ**(x)  
*Construct.*
- **#define SIGMA0\_REQ\_FPAGE\_RAM** (SIGMA0\_REQ(FPAGE\_RAM))  
*RAM.*

- #define **SIGMA0\_REQ\_FPAGE\_IOMEM** ([SIGMA0\\_REQ](#)(FPAGE\_IOMEM))  
*I/O memory.*
- #define **SIGMA0\_REQ\_FPAGE\_IOMEM\_CACHED** ([SIGMA0\\_REQ](#)(FPAGE\_IOMEM\_CACHED))  
*Cache I/O memory.*
- #define **SIGMA0\_REQ\_FPAGE\_ANY** ([SIGMA0\\_REQ](#)(FPAGE\_ANY))  
*Any.*
- #define **SIGMA0\_REQ\_KIP** ([SIGMA0\\_REQ](#)(KIP))  
*KIP.*
- #define **SIGMA0\_REQ\_DEBUG\_DUMP** ([SIGMA0\\_REQ](#)(DEBUG\_DUMP))  
*Debug dump.*

## Enumerations

- enum [l4sigma0\\_return\\_flags\\_t](#) {  
    [L4SIGMA0\\_OK](#) , [L4SIGMA0\\_NOTALIGNED](#) , [L4SIGMA0\\_IPCERROR](#) , [L4SIGMA0\\_NOFPAGE](#) ,  
    [L4SIGMA0\\_4](#) , [L4SIGMA0\\_5](#) , [L4SIGMA0\\_SMALLERFPAGE](#) }  
*Return flags of libsigma0 functions.*

## Functions

- [L4\\_BEGIN\\_DECLS](#) [l4\\_kernel\\_info\\_t](#) \* [l4sigma0\\_map\\_kip](#) ([l4\\_cap\\_idx\\_t](#) sigma0, void \*addr, unsigned log2↵\_size)  
*Map the kernel info page from sigma0 to addr.*
- int [l4sigma0\\_map\\_mem](#) ([l4\\_cap\\_idx\\_t](#) sigma0, [l4\\_addr\\_t](#) phys, [l4\\_addr\\_t](#) virt, [l4\\_addr\\_t](#) size)  
*Request a memory mapping from sigma0.*
- int [l4sigma0\\_map\\_iomem](#) ([l4\\_cap\\_idx\\_t](#) sigma0, [l4\\_addr\\_t](#) phys, [l4\\_addr\\_t](#) virt, [l4\\_addr\\_t](#) size, int cached)  
*Request IO memory from sigma0.*
- int [l4sigma0\\_map\\_anypage](#) ([l4\\_cap\\_idx\\_t](#) sigma0, [l4\\_addr\\_t](#) map\_area, unsigned log2\_map\_size, [l4\\_addr\\_t](#) \*base, unsigned sz)  
*Request an arbitrary free page of RAM.*
- void [l4sigma0\\_debug\\_dump](#) ([l4\\_cap\\_idx\\_t](#) sigma0)  
*Request sigma0 to dump internal debug information.*
- char const \* [l4sigma0\\_map\\_errstr](#) (int err)  
*Get user readable error messages for the return codes.*

### 17.434.1 Detailed Description

Sigma0 interface.

Definition in file [sigma0.h](#).



## 17.435 sigma0.h

[Go to the documentation of this file.](#)

```

00001
00007 /*
00008 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00009 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00010 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00011 * economic rights: Technische Universität Dresden (Germany)
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #ifndef __L4_SIGMA0_SIGMA0_H
00015 #define __L4_SIGMA0_SIGMA0_H
00016
00024
00025 #include <l4/sys/compiler.h>
00026 #include <l4/sys/types.h>
00027 #include <l4/sys/kip.h>
00028
00035 #undef SIGMA0_REQ_MAGIC
00036 #undef SIGMA0_REQ_MASK
00037
00038 # define SIGMA0_REQ_MAGIC ~0xFFUL
00039 # define SIGMA0_REQ_MASK ~0xFFUL
00040
00041 /* Starting with 0x60 allows to detect components which still use the old
00042 * constants (0x00 ... 0x50) */
00043 #define SIGMA0_REQ_ID_MASK 0xF0
00044 #define SIGMA0_REQ_ID_FPAGE_RAM 0x60
00045 #define SIGMA0_REQ_ID_FPAGE_IOMEM 0x70
00046 #define SIGMA0_REQ_ID_FPAGE_IOMEM_CACHED 0x80
00047 #define SIGMA0_REQ_ID_FPAGE_ANY 0x90
00048 #define SIGMA0_REQ_ID_KIP 0xA0
00049 #define SIGMA0_REQ_ID_DEBUG_DUMP 0xC0
00050
00051 #define SIGMA0_IS_MAGIC_REQ(d1) \
00052 ((d1 & SIGMA0_REQ_MASK) == SIGMA0_REQ_MAGIC)
00053
00054 #define SIGMA0_REQ(x) \
00055 (SIGMA0_REQ_MAGIC + SIGMA0_REQ_ID_ ## x)
00056
00057 /* Use these constants in your code! */
00058 #define SIGMA0_REQ_FPAGE_RAM (SIGMA0_REQ(FPAGE_RAM))
00059 #define SIGMA0_REQ_FPAGE_IOMEM (SIGMA0_REQ(FPAGE_IOMEM))
00060 #define SIGMA0_REQ_FPAGE_IOMEM_CACHED (SIGMA0_REQ(FPAGE_IOMEM_CACHED))
00061 #define SIGMA0_REQ_FPAGE_ANY (SIGMA0_REQ(FPAGE_ANY))
00062 #define SIGMA0_REQ_KIP (SIGMA0_REQ(KIP))
00063 #define SIGMA0_REQ_DEBUG_DUMP (SIGMA0_REQ(DEBUG_DUMP))
00064
00065
00070
00074 enum l4sigma0_return_flags_t {
00075 L4SIGMA0_OK,
00076 L4SIGMA0_NOTALIGNED,
00077 L4SIGMA0_IPCERROR,
00078 L4SIGMA0_NOFPAGE,
00079 L4SIGMA0_4,
00080 L4SIGMA0_5,
00081 L4SIGMA0_SMALLERFPAGE,
00082 };
00083
00084 L4_BEGIN_DECLS
00085
00095 L4_CV l4_kernel_info_t *
00096 l4sigma0_map_kip(l4_cap_idx_t sigma0, void *addr, unsigned log2_size);
00097
00127 L4_CV int l4sigma0_map_mem(l4_cap_idx_t sigma0,
00128 l4_addr_t phys, l4_addr_t virt, l4_addr_t size);
00129
00152 L4_CV int l4sigma0_map_iomem(l4_cap_idx_t sigma0, l4_addr_t phys,
00153 l4_addr_t virt, l4_addr_t size, int cached);
00178 L4_CV int l4sigma0_map_anypage(l4_cap_idx_t sigma0, l4_addr_t map_area,
00179 unsigned log2_map_size, l4_addr_t *base,
00180 unsigned sz);
00181
00190 L4_CV void l4sigma0_debug_dump(l4_cap_idx_t sigma0);
00191
00199 L4_INLINE char const *l4sigma0_map_errstr(int err);
00200
00202
00203
00204 /* Implementations */
00205
00206 L4_INLINE char const *l4sigma0_map_errstr(int err)
00207 {
00208 switch (err)

```

```

00209 {
00210 case 0: return "No error";
00211 case -1: return "Phys, virt or size not aligned";
00212 case -2: return "IPC error";
00213 case -3: return "No fpage received";
00214 #ifndef SIGMA0_REQ_MAGIC
00215 case -4: return "Bad physical address (old protocol only)";
00216 #endif
00217 case -6: return "Superpage requested but smaller flexpage received";
00218 case -7: return "Cannot map I/O memory cacheable (old protocol only)";
00219 default: return "Unknown error";
00220 }
00221 }
00222
00223
00224 L4_END_DECLS
00225
00226 #endif /* ! __L4_SIGMA0_SIGMA0_H */

```

## 17.436 \_\_kernel\_object\_impl.h

```

00001
00006 #pragma once
00007
00008 #include <l4/sys/ipc.h>
00009
00010 L4_INLINE l4_msgtag_t
00011 l4_invoke_debugger(l4_cap_idx_t obj, l4_msgtag_t tag, l4_utcb_t *utcb) L4_NOTHROW
00012 {
00013 l4_msgtag_t t2;
00014 unsigned const words = l4_msgtag_words(tag);
00015 l4_msg_regs_t *mr = l4_utcb_mr_u(utcb);
00016
00017 if (l4_is_invalid_cap(obj))
00018 return l4_msgtag(-L4_EINVAL, 0, 0, 0);
00019
00020 if (words + 2 > L4_UTCB_GENERIC_DATA_SIZE)
00021 return l4_msgtag(-L4_EMSTOOLONG, 0, 0, 0);
00022
00023 mr->mr[0] += 0x100;
00024 mr->mr[words] = L4_ITEM_MAP;
00025 mr->mr[words + 1] = l4_obj_fpage(obj, 0, L4_CAP_FPAGE_RWS).raw;
00026 t2 = l4_msgtag(L4_PROTO_DEBUGGER, words, 1, l4_msgtag_flags(tag));
00027
00028 return l4_ipc_call(L4_BASE_DEBUGGER_CAP, utcb, t2, L4_IPC_NEVER);
00029 }
00030

```

## 17.437 l4/sys/\_\_ktrace-impl.h File Reference

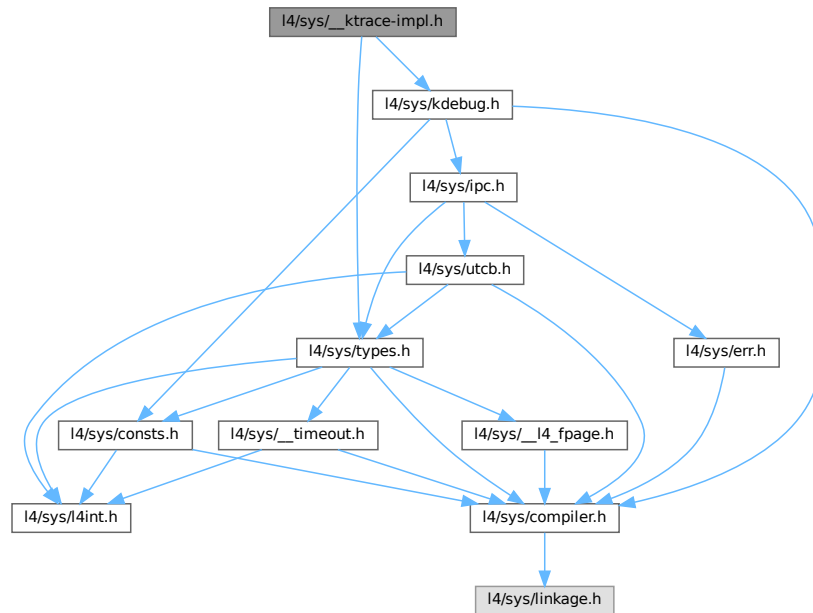
[L4](#) kernel event tracing.

```

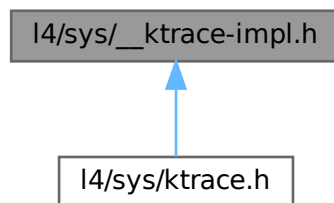
#include <l4/sys/types.h>
#include <l4/sys/kdebug.h>

```

Include dependency graph for \_\_ktrace-impl.h:



This graph shows which files directly or indirectly include this file:



## Functions

- [l4\\_umword\\_t fiasco\\_tbuf\\_log](#) (const char \*text)  
Create new trace-buffer entry with describing <text>.
- [l4\\_umword\\_t fiasco\\_tbuf\\_log\\_3val](#) (const char \*text, [l4\\_umword\\_t](#) v1, [l4\\_umword\\_t](#) v2, [l4\\_umword\\_t](#) v3)  
Create new trace-buffer entry with describing <text> and three additional values.
- void **fiasco\_tbuf\_clear** (void)  
Clear trace-buffer.
- void **fiasco\_tbuf\_dump** (void)  
Dump trace-buffer to kernel console.
- [l4\\_umword\\_t fiasco\\_tbuf\\_log\\_binary](#) (const unsigned char \*data)  
Create new trace-buffer entry with binary data.

## 17.437.1 Detailed Description

[L4](#) kernel event tracing.

Definition in file [\\_\\_ktrace-impl.h](#).

## 17.438 \_\_ktrace-impl.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Björn Döbel <doebel@os.inf.tu-dresden.de>,
00009 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #pragma once
00015
00016 #include <l4/sys/types.h>
00017 #include <l4/sys/kdebug.h>
00018
00019 /*****
00020 *** Implementation
00021 *****/
00022
00023 L4_INLINE l4_umword_t
00024 fiasco_tbuf_log(const char *text)
00025 {
00026 enum { TBUF_LOG = L4_KDEBUG_GROUP_TRACE + 0x01 };
00027 return l4_error(__kdebug_text(TBUF_LOG, text, __builtin_strlen(text)));
00028 }
00029
00030 L4_INLINE l4_umword_t
00031 fiasco_tbuf_log_3val(const char *text, l4_umword_t v1, l4_umword_t v2,
00032 l4_umword_t v3)
00033 {
00034 enum { TBUF_LOG_3VAL = L4_KDEBUG_GROUP_TRACE + 0x04 };
00035 return l4_error(__kdebug_3_text(TBUF_LOG_3VAL, text,
00036 __builtin_strlen(text), v1, v2, v3));
00037 }
00038
00039 L4_INLINE void
00040 fiasco_tbuf_clear(void)
00041 {
00042 enum { TBUF_CLEAR = L4_KDEBUG_GROUP_TRACE + 0x02 };
00043 __kdebug_op(TBUF_CLEAR);
00044 }
00045
00046 L4_INLINE void
00047 fiasco_tbuf_dump(void)
00048 {
00049 enum { TBUF_DUMP = L4_KDEBUG_GROUP_TRACE + 0x03 };
00050 __kdebug_op(TBUF_DUMP);
00051 }
00052
00053 L4_INLINE l4_umword_t
00054 fiasco_tbuf_log_binary(const unsigned char *data)
00055 {
00056 enum { TBUF_LOG_BIN = L4_KDEBUG_GROUP_TRACE + 0x08 };
00057 return l4_error(__kdebug_text(TBUF_LOG_BIN, (const char *)data, 24));
00058 }
00059

```

## 17.439 \_\_l4\_fpage.h

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00009 * Björn Döbel <doebel@os.inf.tu-dresden.de>,
00010 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>

```

```

00011 * economic rights: Technische Universität Dresden (Germany)
00012 *
00013 * License: see LICENSE.spdx (in this directory or the directories above)
00014 */
00015 #pragma once
00016
00017 #include <l4/sys/compiler.h>
00018
00043
00048 enum L4_fpage_consts
00049 {
00050 L4_FPAGE_RIGHTS_SHIFT = 0,
00051 L4_FPAGE_TYPE_SHIFT = 4,
00052 L4_FPAGE_SIZE_SHIFT = 6,
00053 L4_FPAGE_ADDR_SHIFT = 12,
00054
00055 L4_FPAGE_RIGHTS_BITS = 4,
00056 L4_FPAGE_TYPE_BITS = 2,
00057 L4_FPAGE_SIZE_BITS = 6,
00058 L4_FPAGE_ADDR_BITS = L4_MWORD_BITS - L4_FPAGE_ADDR_SHIFT,
00059
00061 L4_FPAGE_RIGHTS_MASK = ((1UL < L4_FPAGE_RIGHTS_BITS) - 1)
00062 < L4_FPAGE_RIGHTS_SHIFT,
00063 L4_FPAGE_TYPE_MASK = ((1UL < L4_FPAGE_TYPE_BITS) - 1)
00064 < L4_FPAGE_TYPE_SHIFT,
00065 L4_FPAGE_SIZE_MASK = ((1UL < L4_FPAGE_SIZE_BITS) - 1)
00066 < L4_FPAGE_SIZE_SHIFT,
00067 L4_FPAGE_ADDR_MASK = ~0UL < L4_FPAGE_ADDR_SHIFT,
00069 L4_FPAGE_RIGHTS_ALL = L4_FPAGE_RIGHTS_MASK,
00070 };
00071
00076 typedef union {
00077 l4_umword_t fpage;
00078 l4_umword_t raw;
00079 } l4_fpage_t;
00080
00084 enum
00085 {
00092 L4_WHOLE_ADDRESS_SPACE = 63
00093 };
00094
00099 typedef struct {
00100 l4_umword_t snd_base;
00101 l4_fpage_t fpage;
00102 } l4_snd_fpage_t;
00103
00104
00118 enum L4_fpage_rights
00119 {
00120 L4_FPAGE_X = 1,
00121 L4_FPAGE_W = 2,
00122 L4_FPAGE_RO = 4,
00123 L4_FPAGE_RW = L4_FPAGE_RO | L4_FPAGE_W,
00124 L4_FPAGE_RX = L4_FPAGE_RO | L4_FPAGE_X,
00125 L4_FPAGE_RWX = L4_FPAGE_RW | L4_FPAGE_X,
00126 };
00127
00148 enum L4_cap_fpage_rights
00149 {
00157 L4_CAP_FPAGE_W = 0x1,
00169 L4_CAP_FPAGE_S = 0x2,
00175 L4_CAP_FPAGE_R = 0x4,
00176 L4_CAP_FPAGE_RO = 0x4,
00185 L4_CAP_FPAGE_D = 0x8,
00192 L4_CAP_FPAGE_RW = L4_CAP_FPAGE_R | L4_CAP_FPAGE_W,
00199 L4_CAP_FPAGE_RS = L4_CAP_FPAGE_R | L4_CAP_FPAGE_S,
00206 L4_CAP_FPAGE_RWS = L4_CAP_FPAGE_RW | L4_CAP_FPAGE_S,
00212 L4_CAP_FPAGE_RWSD = L4_CAP_FPAGE_RWS | L4_CAP_FPAGE_D,
00218 L4_CAP_FPAGE_RWD = L4_CAP_FPAGE_RW | L4_CAP_FPAGE_D,
00224 L4_CAP_FPAGE_RSD = L4_CAP_FPAGE_RS | L4_CAP_FPAGE_D,
00225 };
00226
00230 enum L4_fpage_type
00231 {
00232 L4_FPAGE_SPECIAL = 0,
00235 L4_FPAGE_MEMORY = 1,
00236 L4_FPAGE_IO = 2,
00237 L4_FPAGE_OBJ = 3,
00238 };
00239
00243 enum L4_fpage_control
00244 {
00247 L4_FPAGE_CONTROL_OFFSET_SHIFT = 12,
00250 L4_FPAGE_CONTROL_MASK = ~0UL < L4_FPAGE_CONTROL_OFFSET_SHIFT,
00251 };
00252
00262 enum L4_obj_fpage_ctl

```

```

00263 {
00264 L4_FPAGE_C_REF_CNT = 0x00,
00265 L4_FPAGE_C_NO_REF_CNT = 0x10,
00266
00267 L4_FPAGE_C_OBJ_RIGHT1 = 0x20,
00268 L4_FPAGE_C_OBJ_RIGHT2 = 0x40,
00269 L4_FPAGE_C_OBJ_RIGHT3 = 0x80,
00270 L4_FPAGE_C_OBJ_RIGHTS = 0xe0,
00271
00277 L4_FPAGE_C_IPCGATE_SVR = L4_FPAGE_C_OBJ_RIGHT1
00278 };
00279
00280
00291 enum l4_fpage_cacheability_opt_t
00292 {
00295 L4_FPAGE_CACHE_OPT = 0x1,
00296
00299 L4_FPAGE_CACHEABLE = 0x3,
00300
00303 L4_FPAGE_BUFFERABLE = 0x5,
00304
00307 L4_FPAGE_UNCACHEABLE = 0x1
00308 };
00309
00310
00314 enum
00315 {
00319 L4_WHOLE_IOADDRESS_SPACE = 16,
00320
00322 L4_IOPORT_MAX = (1L << L4_WHOLE_IOADDRESS_SPACE)
00323 };
00324
00325
00326
00341 L4_INLINE l4_fpage_t
00342 l4_fpage(l4_addr_t address, unsigned int order, unsigned char rights) L4_NOTHROW;
00343
00355 L4_INLINE l4_fpage_t
00356 l4_fpage_all(void) L4_NOTHROW;
00357
00364 L4_INLINE l4_fpage_t
00365 l4_fpage_invalid(void) L4_NOTHROW;
00366
00367
00378 L4_INLINE l4_fpage_t
00379 l4_iofpage(unsigned long port, unsigned int order) L4_NOTHROW;
00380
00381
00394 L4_INLINE l4_fpage_t
00395 l4_obj_fpage(l4_cap_idx_t obj, unsigned int order, unsigned char rights) L4_NOTHROW;
00396
00406 L4_INLINE int
00407 l4_is_fpage_writable(l4_fpage_t fp) L4_NOTHROW;
00408
00409
00440
00456 L4_INLINE l4_umword_t
00457 l4_map_control(l4_umword_t spot, unsigned char cache, unsigned grant) L4_NOTHROW;
00458
00472 L4_INLINE l4_umword_t
00473 l4_map_obj_control(l4_umword_t spot, unsigned grant) L4_NOTHROW;
00474
00483 L4_INLINE unsigned
00484 l4_fpage_rights(l4_fpage_t f) L4_NOTHROW;
00485
00494 L4_INLINE unsigned
00495 l4_fpage_type(l4_fpage_t f) L4_NOTHROW;
00496
00507 L4_INLINE unsigned
00508 l4_fpage_size(l4_fpage_t f) L4_NOTHROW;
00509
00520 L4_INLINE unsigned long
00521 l4_fpage_page(l4_fpage_t f) L4_NOTHROW;
00522
00536 L4_INLINE l4_addr_t
00537 l4_fpage_memaddr(l4_fpage_t f) L4_NOTHROW;
00538
00552 L4_INLINE l4_cap_idx_t
00553 l4_fpage_obj(l4_fpage_t f) L4_NOTHROW;
00554
00568 L4_INLINE unsigned long
00569 l4_fpage_ioport(l4_fpage_t f) L4_NOTHROW;
00570
00580 L4_INLINE l4_fpage_t
00581 l4_fpage_set_rights(l4_fpage_t src, unsigned char new_rights) L4_NOTHROW;
00582
00594 L4_INLINE int

```

```

00595 l4_fpage_contains(l4_fpage_t fpage, l4_addr_t addr, unsigned order) L4_NOTHROW;
00596
00613 L4_INLINE unsigned char
00614 l4_fpage_max_order(unsigned char order, l4_addr_t addr,
00615 l4_addr_t min_addr, l4_addr_t max_addr,
00616 l4_addr_t hotspot L4_DEFAULT_PARAM(0));
00617
00627 L4_INLINE int
00628 l4_is_fpage_valid(l4_fpage_t fp) L4_NOTHROW;
00629
00630 /*****
00631 * Implementations
00632 *****/
00633
00634 L4_INLINE unsigned
00635 l4_fpage_rights(l4_fpage_t f) L4_NOTHROW
00636 {
00637 return (f.raw & L4_FPAGE_RIGHTS_MASK) » L4_FPAGE_RIGHTS_SHIFT;
00638 }
00639
00640 L4_INLINE unsigned
00641 l4_fpage_type(l4_fpage_t f) L4_NOTHROW
00642 {
00643 return (f.raw & L4_FPAGE_TYPE_MASK) » L4_FPAGE_TYPE_SHIFT;
00644 }
00645
00646 L4_INLINE unsigned
00647 l4_fpage_size(l4_fpage_t f) L4_NOTHROW
00648 {
00649 return (f.raw & L4_FPAGE_SIZE_MASK) » L4_FPAGE_SIZE_SHIFT;
00650 }
00651
00652 L4_INLINE unsigned long
00653 l4_fpage_page(l4_fpage_t f) L4_NOTHROW
00654 {
00655 return (f.raw & L4_FPAGE_ADDR_MASK) » L4_FPAGE_ADDR_SHIFT;
00656 }
00657
00658 L4_INLINE unsigned long
00659 l4_fpage_ioport(l4_fpage_t f) L4_NOTHROW
00660 {
00661 return (f.raw & L4_FPAGE_ADDR_MASK) » L4_FPAGE_ADDR_SHIFT;
00662 }
00663
00664 L4_INLINE l4_addr_t
00665 l4_fpage_memaddr(l4_fpage_t f) L4_NOTHROW
00666 {
00667 return f.raw & L4_FPAGE_ADDR_MASK;
00668 }
00669
00670 L4_INLINE l4_cap_idx_t
00671 l4_fpage_obj(l4_fpage_t f) L4_NOTHROW
00672 {
00673 return f.raw & L4_FPAGE_ADDR_MASK;
00674 }
00675
00677 L4_INLINE l4_fpage_t
00678 __l4_fpage_generic(unsigned long address, unsigned int type,
00679 unsigned int order, unsigned char rights) L4_NOTHROW;
00680
00681 L4_INLINE l4_fpage_t
00682 __l4_fpage_generic(unsigned long address, unsigned int type,
00683 unsigned int order, unsigned char rights) L4_NOTHROW
00684 {
00685 l4_fpage_t t;
00686 t.raw = ((rights « L4_FPAGE_RIGHTS_SHIFT) & L4_FPAGE_RIGHTS_MASK)
00687 | ((type « L4_FPAGE_TYPE_SHIFT) & L4_FPAGE_TYPE_MASK)
00688 | ((order « L4_FPAGE_SIZE_SHIFT) & L4_FPAGE_SIZE_MASK)
00689 | ((address & L4_FPAGE_ADDR_MASK) & L4_FPAGE_ADDR_MASK);
00690 return t;
00691 }
00692
00693 L4_INLINE l4_fpage_t
00694 l4_fpage_set_rights(l4_fpage_t src, unsigned char new_rights) L4_NOTHROW
00695 {
00696 l4_fpage_t f;
00697 f.raw = ((L4_FPAGE_TYPE_MASK | L4_FPAGE_SIZE_MASK | L4_FPAGE_ADDR_MASK) & src.raw)
00698 | ((new_rights « L4_FPAGE_RIGHTS_SHIFT) & L4_FPAGE_RIGHTS_MASK);
00699 return f;
00700 }
00701
00702 L4_INLINE l4_fpage_t
00703 l4_fpage(l4_addr_t address, unsigned int order, unsigned char rights) L4_NOTHROW
00704 {
00705 return __l4_fpage_generic(address, L4_FPAGE_MEMORY, order, rights);
00706 }
00707

```

```

00708 L4_INLINE l4_fpage_t
00709 l4_iofpage(unsigned long port, unsigned int order) L4_NOTHROW
00710 {
00711 return __l4_fpage_generic(port < L4_FPAGE_ADDR_SHIFT, L4_FPAGE_IO, order, L4_FPAGE_RW);
00712 }
00713
00714 L4_INLINE l4_fpage_t
00715 l4_obj_fpage(l4_cap_idx_t obj, unsigned int order, unsigned char rights) L4_NOTHROW
00716 {
00717 static_assert((unsigned long)L4_CAP_SHIFT >= L4_FPAGE_ADDR_SHIFT,
00718 "Capability index does not fit into fpage.");
00719 return __l4_fpage_generic(obj, L4_FPAGE_OBJ, order, rights);
00720 }
00721
00722 L4_INLINE l4_fpage_t
00723 l4_fpage_all(void) L4_NOTHROW
00724 {
00725 return __l4_fpage_generic(0, L4_FPAGE_SPECIAL, L4_WHOLE_ADDRESS_SPACE, 0);
00726 }
00727
00728 L4_INLINE l4_fpage_t
00729 l4_fpage_invalid(void) L4_NOTHROW
00730 {
00731 return __l4_fpage_generic(0, L4_FPAGE_SPECIAL, 0, 0);
00732 }
00733
00734
00735 L4_INLINE int
00736 l4_is_fpage_writable(l4_fpage_t fp) L4_NOTHROW
00737 {
00738 return l4_fpage_rights(fp) & L4_FPAGE_W;
00739 }
00740
00741 L4_INLINE l4_umword_t
00742 l4_map_control(l4_umword_t snd_base, unsigned char cache, unsigned grant) L4_NOTHROW
00743 {
00744 return (snd_base & L4_FPAGE_CONTROL_MASK)
00745 | ((l4_umword_t)cache << 4) | L4_ITEM_MAP | grant;
00746 }
00747
00748 L4_INLINE l4_umword_t
00749 l4_map_obj_control(l4_umword_t snd_base, unsigned grant) L4_NOTHROW
00750 {
00751 return l4_map_control(snd_base, 0, grant);
00752 }
00753
00754 L4_INLINE int
00755 l4_fpage_contains(l4_fpage_t fpage, l4_addr_t addr, unsigned log2size) L4_NOTHROW
00756 {
00757 l4_addr_t fa = l4_fpage_memaddr(fpage);
00758 return (fa <= addr)
00759 && (fa + (1UL << l4_fpage_size(fpage)) >= addr + (1UL << log2size));
00760 }
00761
00762 L4_INLINE unsigned char
00763 l4_fpage_max_order(unsigned char order, l4_addr_t addr,
00764 l4_addr_t min_addr, l4_addr_t max_addr,
00765 l4_addr_t hotspot)
00766 {
00767 while (order < 30 /* limit to 1GB flexpages */)
00768 {
00769 l4_addr_t mask;
00770 l4_addr_t base = l4_trunc_size(addr, order + 1);
00771 if (base < min_addr)
00772 return order;
00773
00774 if (base + (1UL << (order + 1)) - 1 > max_addr - 1)
00775 return order;
00776
00777 mask = ~(~0UL << (order + 1));
00778 if (hotspot == ~0UL || ((addr ^ hotspot) & mask))
00779 break;
00780
00781 ++order;
00782 }
00783
00784 return order;
00785 }
00786
00787 L4_INLINE int
00788 l4_is_fpage_valid(l4_fpage_t fp) L4_NOTHROW
00789 {
00790 return l4_fpage_type(fp) != L4_FPAGE_SPECIAL || l4_fpage_size(fp) != 0;
00791 }

```



## 17.440 \_\_platform\_control-arm.h

```

00001 /*
00002 * Copyright (C) 2024 Kernkonzept GmbH.
00003 * Author(s): Jan Klötzke <jan.kloetzke@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/sys/types.h>
00010
00011
00029 L4_INLINE l4_msgtag_t
00030 l4_platform_ctl_set_task_asid(l4_cap_idx_t pfc,
00031 l4_cap_idx_t task,
00032 l4_umword_t asid) L4_NOTHROW;
00033
00034
00038 L4_INLINE l4_msgtag_t
00039 l4_platform_ctl_set_task_asid_u(l4_cap_idx_t pfc,
00040 l4_cap_idx_t task,
00041 l4_umword_t asid,
00042 l4_utcb_t *utcb) L4_NOTHROW;
00043
00044 /* IMPLEMENTATION -----*/
00045
00046 L4_INLINE l4_msgtag_t
00047 l4_platform_ctl_set_task_asid_u(l4_cap_idx_t pfc,
00048 l4_cap_idx_t task,
00049 l4_umword_t asid,
00050 l4_utcb_t *utcb) L4_NOTHROW
00051 {
00052 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00053 v->mr[0] = L4_PLATFORM_CTL_SET_TASK_ASID_OP;
00054 v->mr[1] = asid;
00055 v->mr[2] = l4_map_obj_control(0, 0);
00056 v->mr[3] = l4_obj_fpage(task, 0, L4_CAP_FPAGE_RWS).raw;
00057 return l4_ipc_call(pfc, utcb, l4_msgtag(L4_PROTO_PLATFORM_CTL, 2, 1, 0),
00058 L4_IPC_NEVER);
00059 }
00060
00061 L4_INLINE l4_msgtag_t
00062 l4_platform_ctl_set_task_asid(l4_cap_idx_t pfc,
00063 l4_cap_idx_t task,
00064 l4_umword_t asid) L4_NOTHROW
00065 {
00066 return l4_platform_ctl_set_task_asid_u(pfc, task, asid, l4_utcb());
00067 }

```

## 17.441 \_\_task-arm.h

```

00001 /*
00002 * (c) 2018 Adam Lackorzynski <adam@l4re.org>
00003 *
00004 * License: see LICENSE.spdx (in this directory or the directories above)
00005 */
00006 #pragma once
00007
00008 #include <l4/sys/types.h>
00009
00013 L4_INLINE l4_msgtag_t
00014 l4_task_vgicc_map_u(l4_cap_idx_t task, l4_fpage_t vgicc_fpage,
00015 l4_utcb_t *u) L4_NOTHROW;
00016
00028 L4_INLINE l4_msgtag_t
00029 l4_task_vgicc_map(l4_cap_idx_t task, l4_fpage_t vgicc_fpage) L4_NOTHROW;
00030
00031 /* IMPLEMENTATION -----*/
00032
00033 #include <l4/sys/ipc.h>
00034
00035 L4_INLINE l4_msgtag_t
00036 l4_task_vgicc_map_u(l4_cap_idx_t task, l4_fpage_t vgicc_fpage,
00037 l4_utcb_t *u) L4_NOTHROW
00038 {
00039 l4_msg_regs_t *v = l4_utcb_mr_u(u);
00040 v->mr[0] = L4_TASK_MAP_VGICC_ARM_OP;
00041 v->mr[1] = vgicc_fpage.raw;
00042 return l4_ipc_call(task, u, l4_msgtag(L4_PROTO_TASK, 2, 0, 0), L4_IPC_NEVER);
00043 }
00044

```

```

00045 L4_INLINE l4_msgtag_t
00046 l4_task_vgicc_map(l4_cap_idx_t task, l4_fpage_t vgicc_fpage) L4_NOTHROW
00047 {
00048 return l4_task_vgicc_map_u(task, vgicc_fpage, l4_utcb());
00049 }

```

## 17.442 \_\_timeout.h

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00009 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #ifndef L4_SYS_TIMEOUT_H__
00015 #define L4_SYS_TIMEOUT_H__
00016
00017 #include <l4/sys/l4int.h>
00018 #include <l4/sys/compiler.h>
00019
00025
00040 typedef struct l4_timeout_s {
00041 l4_uint16_t t;
00042 } __attribute__((packed)) l4_timeout_s;
00043
00044
00052 typedef union l4_timeout_t
00053 {
00054 l4_uint32_t raw;
00055 struct
00056 {
00057 #ifdef __BIG_ENDIAN__
00058 l4_timeout_s snd;
00059 l4_timeout_s rcv;
00060 #else
00061 l4_timeout_s rcv;
00062 l4_timeout_s snd;
00063 #endif
00064 } p;
00065 } l4_timeout_t;
00066
00067
00073 #define L4_IPC_TIMEOUT_0 ((l4_timeout_s){0x0400})
00074 #define L4_IPC_TIMEOUT_NEVER ((l4_timeout_s){0})
00075 #define L4_IPC_NEVER_INITIALIZER {0}
00076 #define L4_IPC_NEVER ((l4_timeout_t){0})
00077 #define L4_IPC_RECV_TIMEOUT_0 ((l4_timeout_t){0x00000400})
00078 #define L4_IPC_SEND_TIMEOUT_0 ((l4_timeout_t){0x04000000})
00079 #define L4_IPC_BOTH_TIMEOUT_0 ((l4_timeout_t){0x04000400})
00080
00085 #define L4_TIMEOUT_US_NEVER (~0ULL)
00086
00091 #define L4_TIMEOUT_US_MAX ((1ULL << 41) - 1)
00092
00094
00104 L4_CONSTEXPR L4_INLINE
00105 l4_timeout_s l4_timeout_rel(unsigned man, unsigned exp) L4_NOTHROW;
00106
00107
00117 L4_CONSTEXPR L4_INLINE
00118 l4_timeout_t l4_ipc_timeout(unsigned snd_man, unsigned snd_exp,
00119 unsigned rcv_man, unsigned rcv_exp) L4_NOTHROW;
00120
00130 L4_CONSTEXPR L4_INLINE
00131 l4_timeout_t l4_timeout(l4_timeout_s snd, l4_timeout_s rcv) L4_NOTHROW;
00132
00140 L4_CONSTEXPR L4_INLINE
00141 void l4_snd_timeout(l4_timeout_s snd, l4_timeout_t *to) L4_NOTHROW;
00142
00150 L4_CONSTEXPR L4_INLINE
00151 void l4_rcv_timeout(l4_timeout_s rcv, l4_timeout_t *to) L4_NOTHROW;
00152
00161 L4_CONSTEXPR L4_INLINE
00162 l4_kernel_clock_t l4_timeout_rel_get(l4_timeout_s to) L4_NOTHROW;
00163
00164
00173 L4_CONSTEXPR L4_INLINE
00174 unsigned l4_timeout_is_absolute(l4_timeout_s to) L4_NOTHROW;
00175
00185 L4_CONSTEXPR L4_INLINE

```

```

00186 l4_kernel_clock_t l4_timeout_get(l4_kernel_clock_t cur, l4_timeout_s to) L4_NOTHROW;
00187
00195 L4_CONSTEXPR L4_INLINE
00196 l4_timeout_s l4_timeout_from_us(l4_uint64_t us) L4_NOTHROW;
00197
00198 /*
00199 * Implementation
00200 */
00201
00202 L4_CONSTEXPR L4_INLINE
00203 l4_timeout_t l4_ipc_timeout(unsigned snd_man, unsigned snd_exp,
00204 unsigned rcv_man, unsigned rcv_exp) L4_NOTHROW
00205 {
00206 l4_uint16_t snd = (snd_man & 0x3ff) | ((snd_exp << 10) & 0x7c00);
00207 l4_uint16_t rcv = (rcv_man & 0x3ff) | ((rcv_exp << 10) & 0x7c00);
00208 return l4_timeout((l4_timeout_s){snd}, (l4_timeout_s){rcv});
00209 }
00210
00211
00212 L4_CONSTEXPR L4_INLINE
00213 l4_timeout_t l4_timeout(l4_timeout_s snd, l4_timeout_s rcv) L4_NOTHROW
00214 {
00215 return (l4_timeout_t){ ((l4_uint32_t){snd.t} << 16) | rcv.t };
00216 }
00217
00218
00219 L4_CONSTEXPR L4_INLINE
00220 void l4_snd_timeout(l4_timeout_s snd, l4_timeout_t *to) L4_NOTHROW
00221 {
00222 to->p.snd = snd;
00223 }
00224
00225
00226 L4_CONSTEXPR L4_INLINE
00227 void l4_rcv_timeout(l4_timeout_s rcv, l4_timeout_t *to) L4_NOTHROW
00228 {
00229 to->p.rcv = rcv;
00230 }
00231
00232
00233 L4_CONSTEXPR L4_INLINE
00234 l4_timeout_s l4_timeout_rel(unsigned man, unsigned exp) L4_NOTHROW
00235 {
00236 return (l4_timeout_s){(l4_uint16_t)((man & 0x3ff) | ((exp << 10) & 0x7c00))};
00237 }
00238
00239
00240 L4_CONSTEXPR L4_INLINE
00241 l4_kernel_clock_t l4_timeout_rel_get(l4_timeout_s to) L4_NOTHROW
00242 {
00243 if (to.t == 0)
00244 return ~0ULL;
00245 return (l4_kernel_clock_t)(to.t & 0x3ff) << ((to.t >> 10) & 0x1f);
00246 }
00247
00248
00249 L4_CONSTEXPR L4_INLINE
00250 unsigned l4_timeout_is_absolute(l4_timeout_s to) L4_NOTHROW
00251 {
00252 return to.t & 0x8000;
00253 }
00254
00255
00256 L4_CONSTEXPR L4_INLINE
00257 l4_kernel_clock_t l4_timeout_get(l4_kernel_clock_t cur, l4_timeout_s to) L4_NOTHROW
00258 {
00259 if (l4_timeout_is_absolute(to))
00260 return 0; /* We cannot retrieve the value ... */
00261 else
00262 return cur + l4_timeout_rel_get(to);
00263 }
00264
00265 L4_CONSTEXPR L4_INLINE
00266 l4_timeout_s l4_timeout_from_us(l4_uint64_t us) L4_NOTHROW
00267 {
00268 if (us == 0)
00269 return L4_IPC_TIMEOUT_0;
00270 else if (us == L4_TIMEOUT_US_NEVER || us > L4_TIMEOUT_US_MAX)
00271 return L4_IPC_TIMEOUT_NEVER;
00272 else
00273 {
00274 /* Here it is certain that at least one bit in 'us' is set. */
00275
00276 l4_uint16_t m = 0; // initialization required by constexpr, optimized away
00277 l4_uint16_t v = 0; // initialization required by constexpr, optimized away
00278 int e = (63 - __builtin_clzll(us)) - 9;
00279 if (e < 0)

```

```

00280 e = 0;
00281
00282 /* Here it is certain that '0 <= e <= 31' and '1 <= 2^e <= 2^31':
00283 * L4_TIMEOUT_US_MAX = 2^41-1 = 0x000001fffffffffff => e = 31.
00284 * Note: 2^41-1 (0x000001fffffffffff) > 1023*2^31 (0x00001ff800000000). */
00285
00286 m = us >> e;
00287
00288 /* Here it is certain that '1 <= m <= 1023. Consider the following cases:
00289 * o 1 <= us <= 1023: e = 0; 2^e = 1; 1 <= us/1 <= 1023
00290 * o 1024 <= us <= 2047: e = 1; 2^e = 2; 512 <= us/2 <= 1023
00291 * o 2048 <= us <= 4095: e = 2; 2^e = 4; 512 <= us/4 <= 1023
00292 * ...
00293 * o 2^31 <= us <= 2^32-1: e = 22; 512 <= us/2^22 <= 1023
00294 * o 2^40 <= us <= 2^41-1: e = 31; 512 <= us/2^31 <= 1023
00295 *
00296 * Dividing by (1<e) ensures that for all us < 2^41: m < 2^10.
00297 *
00298 * Maximum possible timeout using this format: L4_TIMEOUT_US_MAX = 2^41-1:
00299 * e = 31, m = 1023 => 2'196'875'771'904 us = 610h 14m 35s.
00300 */
00301
00302 /* Without introducing 'v' we had to type-cast the expression to
00303 * l4_uint16_t. This cannot be avoided by declaring m and e_pow_10 as
00304 * l4_uint16_t due to C++ integer promotion. */
00305 v = (e < 10) | m;
00306 return (l4_timeout_s){v};
00307 }
00308 }
00309
00310 #endif

```

## 17.443 I4/sys/\_\_typeinfo.h File Reference

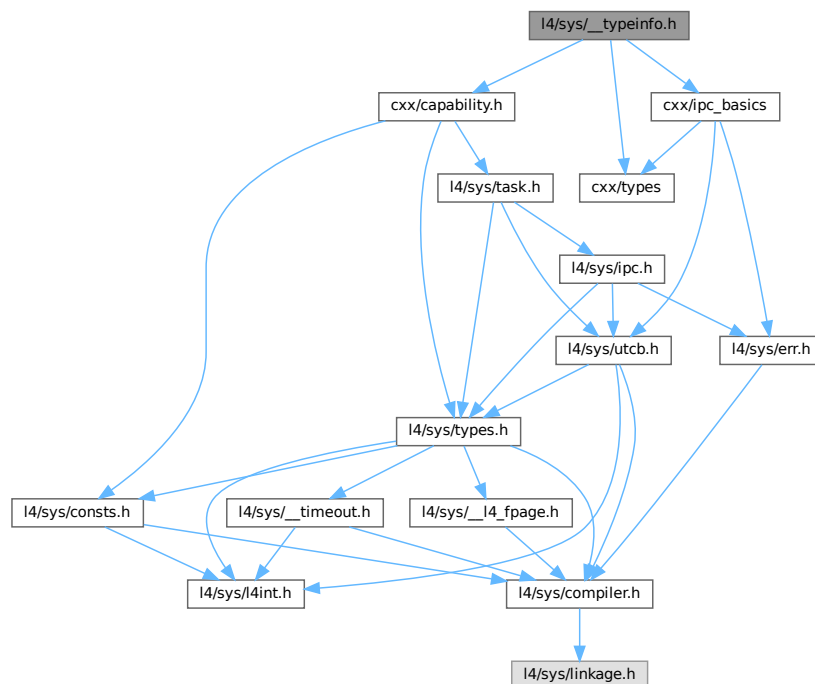
Type information handling.

```

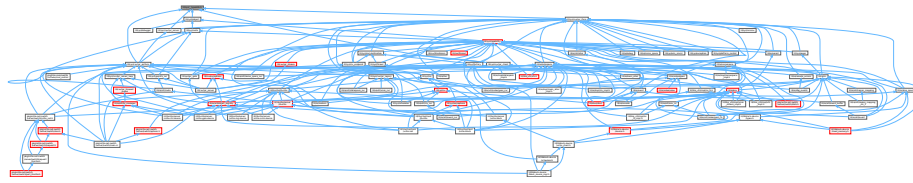
#include "cxx/types"
#include "cxx/ipc_basics"
#include "cxx/capability.h"

```

Include dependency graph for \_\_typeinfo.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [L4::Typeid::P\\_dispatch< LIST >](#)  
*Use for protocol based dispatch stage.*
- struct [L4::Typeid::Detail::Rpc\\_end](#)  
*Internal end-of-list marker.*
- struct [L4::Typeid::Detail::\\_Rpc< OPCODE, O, X >](#)  
*Empty list of RPCs.*
- struct [L4::Typeid::Detail::\\_Rpc< OPCODE, O, R, X... >](#)  
*Non-empty list of RPCs.*
- struct [L4::Typeid::Detail::\\_Rpc< OPCODE, O, R, X... >::Rpc< Y >](#)  
*Find the given RPC in the list.*
- struct [L4::Typeid::Detail::\\_Rpc< OPCODE, O, Default\\_op< R > >::Rpc< Y >](#)  
*Find the given RPC in the list.*
- struct [L4::Typeid::Raw\\_ipc< CLASS >](#)  
*RPCs list for passing raw incoming IPC to the server object.*
- struct [L4::Typeid::Rpc< RPCS >](#)  
*Standard list of RPCs of an interface.*
- struct [L4::Typeid::Rpc\\_code< OPCODE\\_TYPE >](#)  
*List of RPCs of an interface using a special opcode type.*
- struct [L4::Typeid::Rpc\\_code< OPCODE\\_TYPE >::F< RPCS >](#)
- struct [L4::Typeid::Rpc\\_nocode< OPERATION >](#)  
*List of RPCs of an interface using a single operation without an opcode.*
- struct [L4::Typeid::Rpc\\_sys< ARG >](#)  
*List of RPCs typically used for kernel interfaces.*
- struct [L4::Type\\_info](#)  
*Dynamic Type Information for [L4Re](#) Interfaces.*
- class [L4::Type\\_info::Demand](#)  
*Data type for expressing the needed receive buffers at the server-side of an interface.*
- struct [L4::Type\\_info::Demand\\_t< CAPS, FLAGS, MEM, PORTS >](#)  
*Template type statically describing demand of receive buffers.*
- struct [L4::Type\\_info::Demand\\_union\\_t< D1, D2 >](#)  
*Template type statically describing the combination of two [Demand](#) object.*
- struct [L4::Kobject\\_typeid< T >](#)  
*Meta object for handling access to type information of Kobjects.*
- struct [L4::Kobject\\_typeid< void >](#)  
*Minimalistic ID for `void` interface.*
- class [L4::Kobject\\_t< Derived, Base, PROTO, S\\_DEMAND >](#)  
*Helper class to create an [L4Re](#) interface class that is derived from a single base class.*
- class [L4::Kobject\\_2t< Derived, Base1, Base2, PROTO, S\\_DEMAND >](#)  
*Helper class to create an [L4Re](#) interface class that is derived from two base classes (see [L4::Kobject\\_t](#)).*
- struct [L4::Kobject\\_3t< Derived, Base1, Base2, Base3, PROTO, S\\_DEMAND >](#)  
*Helper class to create an [L4Re](#) interface class that is derived from three base classes (see [L4::Kobject\\_t](#)).*
- struct [L4::Proto\\_t< P >](#)  
*Data type for defining protocol numbers.*

## Namespaces

- namespace [L4](#)  
*L4 low-level kernel interface.*
- namespace [L4::Typeid](#)  
*Definition of interface data-type helpers.*

## Typedefs

- typedef int [L4::Opcode](#)  
*Data type for RPC opcodes.*

## Enumerations

- enum { [L4::PROTO\\_ANY](#) = 0 , [L4::PROTO\\_EMPTY](#) = -19 }

## Functions

- template<typename T>  
[Type\\_info](#) const \* [L4::kobject\\_typeid](#) () noexcept  
*Get the [L4::Type\\_info](#) for the [L4Re](#) interface given in T.*

## 17.443.1 Detailed Description

Type information handling.

Definition in file [\\_\\_typeinfo.h](#).

## 17.444 [\\_\\_typeinfo.h](#)

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * Copyright (C) 2014-2017, 2019, 2022-2024 Kernkonzept GmbH.
00007 * Author(s): Alexander Warg <alexander.warg@kernkonzept.com>
00008 */
00009 /*
00010 * (c) 2010 Alexander Warg <warg@os.inf.tu-dresden.de>
00011 * economic rights: Technische Universität Dresden (Germany)
00012 *
00013 * License: see LICENSE.spdx (in this directory or the directories above)
00014 */
00015 #pragma once
00016 #pragma GCC system_header
00017
00018 #include "cxx/types"
00019 #include "cxx/ipc_basics"
00020 #include "cxx/capability.h"
00021
00022 #if defined(__GXX_RTTI) && !defined(L4_NO_RTTI)
00023 # include <typeinfo>
00024 typedef std::type_info const *L4_std_type_info_ptr;
00025 # define L4_KOBJECT_META_RTTI(type) (&typeid(type))
00026 inline char const *L4_kobject_type_name(L4_std_type_info_ptr n) noexcept
00027 { return n ? n->name() : 0; }
00028 #else
00029 typedef void const *L4_std_type_info_ptr;
00030 # define L4_KOBJECT_META_RTTI(type) (0)

```

```

00031 inline char const *L4_kobject_type_name(L4_std_type_info_ptr) noexcept
00032 { return 0; }
00033 #endif
00034
00035 namespace L4 {
00036 typedef int Opcode;
00037 // internal max helpers
00038 namespace __I {
00039 // internal max of A nd B helper
00040 template< unsigned char A, unsigned char B>
00041 struct Max { enum { Res = A > B ? A : B }; };
00042 } // namespace __I
00043
00044 enum
00045 {
00047 PROTO_ANY = 0,
00049 PROTO_EMPTY = -19,
00050 };
00051
00052 namespace Typeid {
00071 using namespace L4::Types;
00072
00073 /*****
00074 template<long P, typename T>
00075 struct Iface
00076 {
00077 typedef Iface type;
00078 typedef T iface_type;
00079 enum { Proto = P };
00080 };
00081
00082 /*****
00083 struct Iface_list_end
00084 {
00085 typedef Iface_list_end type;
00086 static bool contains(long) noexcept { return false; }
00087 };
00088
00089 template<typename I, typename N = Iface_list_end>
00090 struct Iface_list
00091 {
00092 typedef Iface_list<I, N> type;
00093
00094 typedef typename I::iface_type iface_type;
00095 typedef N Next;
00096
00097 enum { Proto = I::Proto };
00098
00099 static bool contains(long proto) noexcept
00100 { return (proto == Proto) || Next::contains(proto); }
00101 };
00102
00103 // do not insert PROTO_EMPTY interfaces
00104 template<typename I, typename N>
00105 struct Iface_list<Iface<PROTO_EMPTY, I>, N> : N {};
00106
00107 // do not insert 'void' type interfaces
00108 template<long P, typename N>
00109 struct Iface_list<Iface<P, void>, N> : N {};
00110
00111 /*****
00112 * \internal
00113 * Test if an interface I is in list L
00114 * \tparam I Interface for lookup
00115 * \tparam L Iface_list for search
00116 */
00117 template< typename I, typename L >
00118 struct _In_list;
00119
00120 template< typename I >
00121 struct _In_list<I, Iface_list_end> : False {};
00122
00123 template< typename I, typename N >
00124 struct _In_list<I, Iface_list<I, N> > : True {};
00125
00126 template< typename I, typename I2, typename N >
00127 struct _In_list<I, Iface_list<I2, N> > : _In_list<I, typename N::type> {};
00128
00129 template<typename I, typename L>
00130 struct In_list : _In_list<typename I::type, typename L::type> {};
00131
00132 }

```

```

00156
00157 /*****
00158 /*
00159 * \internal
00160 * Add Helper: add I to interface list L if ADD is true
00161 * \ingroup l4_cxx_ipc_internal
00162 */
00163 template< bool ADD, typename I, typename L>
00164 struct _Iface_list_add;
00165
00166 template< typename I, typename L>
00167 struct _Iface_list_add<false, I, L> : L {};
00168
00169 template< typename I, typename L>
00170 struct _Iface_list_add<true, I, L> : Iface_list<I, L> {};
00171
00172 /*
00173 * \internal
00174 * Add Helper: add I to interface list L if not already in L.
00175 * \ingroup l4_cxx_ipc_internal
00176 */
00177 template< typename I, typename L >
00178 struct Iface_list_add :
00179 _Iface_list_add<
00180 !In_list<I, typename L::type>::value, I, typename L::type>
00181 > {};
00182
00183 /*****
00184 /*
00185 * \internal
00186 * Helper: checking for a conflict between I1 and I2.
00187 * A conflict means I1 and I2 have the same protocol ID but a different
00188 * iface_type.
00189 */
00190 template< typename I1, typename I2 >
00191 struct __Iface_conflict : Bool<I1::Proto != PROTO_EMPTY && I1::Proto == I2::Proto> {};
00192
00193 template< typename I >
00194 struct __Iface_conflict<I, I> : False {};
00195
00196 /*
00197 * \internal
00198 * Helper: checking for a conflict between I and any interface in LIST.
00199 */
00200 template< typename I, typename LIST >
00201 struct _Iface_conflict;
00202
00203 template< typename I >
00204 struct _Iface_conflict<I, Iface_list_end> : False {};
00205
00206 template< typename I, typename I2, typename LIST >
00207 struct _Iface_conflict<I, Iface_list<I2, LIST> > :
00208 Bool<__Iface_conflict<I, I2>::value || _Iface_conflict<I, typename LIST::type>::value>
00209 > {};
00210
00211 template< typename I, typename LIST >
00212 struct Iface_conflict : _Iface_conflict<typename I::type, typename LIST::type> {};
00213
00214 /*****
00215 /*
00216 * \internal
00217 * Helper: merge two interface lists
00218 */
00219 template< typename L1, typename L2 >
00220 struct _Merge_list;
00221
00222 template< typename L >
00223 struct _Merge_list<Iface_list_end, L> : L {};
00224
00225 template< typename I, typename L1, typename L2 >
00226 struct _Merge_list<Iface_list<I, L1>, L2> :
00227 _Merge_list<typename L1::type, typename Iface_list_add<I, L2>::type> {};
00228
00229 template<typename L1, typename L2>
00230 struct Merge_list : _Merge_list<typename L1::type, typename L2::type> {};
00231
00232 /*****
00233 /*
00234 * \internal
00235 * check for conflicts among all interfaces in L1 with any interfaces in L2.
00236 */
00237 template< typename L1, typename L2 >
00238 struct _Conflict;
00239
00240 template< typename L >
00241 struct _Conflict<Iface_list_end, L> : False {};
00242
00243
00244
00245
00246

```



```

00247 template< typename I, typename L1, typename L2 >
00248 struct _Conflict<Iface_list<I, L1>, L2> :
00249 Bool<Iface_conflict<I, typename L2::type>::value
00250 || _Conflict<typename L1::type, typename L2::type>::value> {};
00251
00252 template< typename L1, typename L2 >
00253 struct Conflict : _Conflict<typename L1::type, typename L2::type> {};
00254
00255 // to be removed -----
00256 // p_dispatch code -- for legacy dispatch -----
00257 /*****
00258 /*
00259 * \internal
00260 * helper: Dispatch helper for calling server-side p_dispatch() functions.
00261 */
00262 template<typename LIST>
00263 struct _P_dispatch;
00264
00265 // No matching dispatcher found
00266 template<>
00267 struct _P_dispatch<Iface_list_end>
00268 {
00269 template< typename THIS, typename A1, typename A2 >
00270 static int f(THIS *, long, A1, A2 &) noexcept
00271 { return -L4_EBADPROTO; }
00272 };
00273
00274
00275 // call matching p_dispatch() function
00276 template< typename I, typename LIST >
00277 struct _P_dispatch<Iface_list<I, LIST> >
00278 {
00279 // special handling for the meta protocol, to avoid 'using' murx
00280 template< typename THIS, typename A1, typename A2 >
00281 static int _f(THIS self, A1, A2 &a2, True::type)
00282 {
00283 return self->dispatch_meta_request(a2);
00284 }
00285
00286 // normal p_dispatch() dispatching
00287 template< typename THIS, typename A1, typename A2 >
00288 static int _f(THIS self, A1 a1, A2 &a2, False::type)
00289 {
00290 return self->p_dispatch(reinterpret_cast<typename I::iface_type *>(0),
00291 a1, a2);
00292 }
00293
00294 // dispatch function with switch for meta protocol
00295 template< typename THIS, typename A1, typename A2 >
00296 static int f(THIS *self, long proto, A1 a1, A2 &a2)
00297 {
00298 if (I::Proto == proto)
00299 return _f(self, a1, a2,
00300 Bool<I::Proto == static_cast<long>(L4_PROTO_META)>());
00301
00302 return _P_dispatch<typename LIST::type>::f(self, proto, a1, a2);
00303 }
00304 };
00305
00306 template<typename LIST>
00307 struct P_dispatch : _P_dispatch<typename LIST::type> {};
00308
00309 // end: p_dispatch -----
00310 // end: to be removed -----
00311
00312 template<typename RPC> struct Default_op;
00313
00314 namespace Detail {
00315
00316 struct Rpcs_end
00317 {
00318 typedef void opcode_type;
00319 typedef Rpcs_end rpc;
00320 typedef Rpcs_end type;
00321 };
00322
00323 template<typename O1, typename O2, typename RPCS>
00324 struct _Rpc : _Rpc<typename RPCS::next::rpc, O2, typename RPCS::next>::type {};
00325
00326 template<typename O1, typename O2>
00327 struct _Rpc<O1, O2, Rpcs_end> {};
00328
00329 template<typename OP, typename RPCS>
00330 struct _Rpc<OP, OP, RPCS> : RPCS
00331 {
00332 typedef _Rpc type;
00333 };
00334
00335
00336
00337

```

```

00338 template<typename OP, typename RPCS>
00339 struct Rpc : _Rpc<typename RPCS::rpc, OP, RPCS> {};
00340
00341 template<typename T, unsigned CODE>
00342 struct _Get_opcode
00343 {
00344 template<bool, typename> struct Invalid_opcode {};
00345 template<typename X> struct Invalid_opcode<true, X>;
00346
00347 private:
00348 template<typename U, U> struct _chk;
00349 template<typename U> static long _opc(_chk<int, U::Opcode> *);
00350 template<typename U> static char _opc(...);
00351
00352 template<unsigned SZ, typename U>
00353 struct _Opc { enum { value = CODE }; };
00354
00355 template<typename U>
00356 struct _Opc<sizeof(long), U> { enum { value = U::Opcode }; };
00357
00358 public:
00359 enum { value = _Opc<sizeof(_opc<T>{0}), T::value };
00360 Invalid_opcode<(value < CODE), T> invalid_opcode;
00361 };
00362
00363 template<typename OPCODE, unsigned O, typename ...X>
00364 struct _Rpc : Rpc_end {};
00365
00366 template<typename OPCODE, unsigned O, typename R, typename ...X>
00367 struct _Rpc<OPCODE, O, R, X...>
00368 {
00369 typedef _Rpc type;
00370 typedef OPCODE opcode_type;
00371 typedef R rpc;
00372 typedef typename _Rpc<OPCODE, _Get_opcode<R, O>::value + 1, X...>::type next;
00373 enum { Opcode = _Get_opcode<R, O>::value };
00374 template<typename Y> struct Rpc : Typeid::Detail::Rpc<Y, _Rpc> {};
00375 };
00376
00377 template<typename OPCODE, unsigned O, typename R>
00378 struct _Rpc<OPCODE, O, Default_op<R> >
00379 {
00380 typedef _Rpc type;
00381 typedef void opcode_type;
00382 typedef R rpc;
00383 typedef Rpc_end next;
00384 enum { Opcode = -99 };
00385 template<typename Y> struct Rpc : Typeid::Detail::Rpc<Y, _Rpc> {};
00386 };
00387
00388 } // namespace Detail
00389
00390 template<typename CLASS>
00391 struct Raw_ipc
00392 {
00393 typedef Raw_ipc type;
00394 typedef Detail::Rpc_end next;
00395 typedef void opcode_type;
00396 };
00397
00398 template<typename ...RPCS>
00399 struct Rpc : Detail::_Rpc<L4::Opcode, 0, RPCS...> {};
00400
00401 template<typename OPCODE_TYPE>
00402 struct Rpc_code
00403 {
00404 template<typename ...RPCS>
00405 struct F : Detail::_Rpc<OPCODE_TYPE, 0, RPCS...> {};
00406 };
00407
00408 template<typename OPERATION>
00409 struct Rpc_nocode : Detail::_Rpc<void, 0, OPERATION> {};
00410
00411 template<typename ...ARG>
00412 struct Rpc_sys : Detail::_Rpc<l4_umword_t, 0, ARG...> {};
00413
00414 template<typename CLASS>
00415 struct Rights
00416 {
00417 unsigned rights;
00418 Rights(unsigned rights) noexcept : rights(rights) {}
00419 unsigned operator & (unsigned rhs) const noexcept { return rights & rhs; }
00420 };
00421
00422 } // namespace Typeid
00423
00424 struct L4_EXPORT Type_info

```

```

00500 {
00506 class L4_EXPORT Demand
00507 {
00508 private:
00510 static unsigned char max(unsigned char a, unsigned char b) noexcept
00511 { return a > b ? a : b; }
00512 public:
00514 unsigned char caps;
00515 unsigned char flags;
00516 unsigned char mem;
00517 unsigned char ports;
00518
00526 explicit
00527 Demand(unsigned char caps = 0, unsigned char flags = 0,
00528 unsigned char mem = 0, unsigned char ports = 0) noexcept
00529 : caps(caps), flags(flags), mem(mem), ports(ports) {}
00530
00532 bool no_demand() const noexcept
00533 { return caps == 0 && mem == 0 && ports == 0 && flags == 0; }
00534
00536 Demand operator | (Demand const &rhs) const noexcept
00537 {
00538 return Demand(max(caps, rhs.caps), flags | rhs.flags,
00539 max(mem, rhs.mem), max(ports, rhs.ports));
00540 }
00541 };
00542
00551 template<unsigned char CAPS = 0, unsigned char FLAGS = 0,
00552 unsigned char MEM = 0, unsigned char PORTS = 0>
00553 struct Demand_t : Demand
00554 {
00555 enum
00556 {
00557 Caps = CAPS,
00558 Flags = FLAGS,
00559 Mem = MEM,
00560 Ports = PORTS
00561 };
00562 Demand_t() noexcept : Demand(CAPS, FLAGS, MEM, PORTS) {}
00563 };
00564
00572 template<typename D1, typename D2>
00573 struct Demand_union_t : Demand_t<__I::Max<D1::Caps, D2::Caps>::Res,
00574 D1::Flags | D2::Flags,
00575 __I::Max<D1::Mem, D2::Mem>::Res,
00576 __I::Max<D1::Ports, D2::Ports>::Res>
00577 {};
00578
00579 L4_std_type_info_ptr _type;
00580 Type_info const *const *_bases;
00581 unsigned _num_bases;
00582 long _proto;
00583
00584 L4_std_type_info_ptr type() const noexcept { return _type; }
00585 Type_info const *base(unsigned idx) const noexcept { return _bases[idx]; }
00586 unsigned num_bases() const noexcept { return _num_bases; }
00587 long proto() const noexcept { return _proto; }
00588 char const *name() const noexcept { return L4_kobject_type_name(type()); }
00589 bool has_proto(long proto) const noexcept
00590 {
00591 if (_proto && _proto == proto)
00592 return true;
00593
00594 if (!proto)
00595 return false;
00596
00597 for (unsigned i = 0; i < _num_bases; ++i)
00598 if (base(i)->has_proto(proto))
00599 return true;
00600
00601 return false;
00602 }
00603 };
00604
00610 template<typename T> struct Kobject_typeid
00611 {
00622 typedef typename T::__Kobject_typeid::Demand Demand;
00623 typedef typename T::__Iface::iface_type Iface;
00624 typedef typename T::__Iface_list Iface_list;
00625
00630 static Type_info const *id() noexcept { return &T::__Kobject_typeid::_m; }
00631
00639 static Type_info::Demand demand() noexcept
00640 { return T::__Kobject_typeid::Demand(); }
00641
00642 // to be removed -----

```

```

00643 // p_dispatch -----
00659 template<typename THIS, typename A1, typename A2>
00660 static int proto_dispatch(THIS *self, long proto, A1 a1, A2 &a2)
00661 { return typeid::P_dispatch<typename T::__Iface_list>::f(self, proto, a1, a2); }
00662 // p_dispatch -----
00663 // end: to be removed -----
00664 };
00665
00667 template<> struct Kobject_typeid<void>
00668 {
00669 typedef Type_info::Demand_t<> Demand;
00670 };
00671
00680 template<typename T>
00681 inline
00682 Type_info const *kobject_typeid() noexcept
00683 { return Kobject_typeid<T>::id(); }
00684
00689 #define L4___GEN_TI(t...)
00690 Type_info const t::__Kobject_typeid::_m =
00691 {
00692 L4_KOBJECT_META_RTTI(Derived),
00693 &t::__Kobject_typeid::_b[0],
00694 sizeof(t::__Kobject_typeid::_b) / sizeof(t::__Kobject_typeid::_b[0]),
00695 PROTO
00696 }
00697
00702 #define L4___GEN_TI_MEMBERS(BASE_DEMAND...)
00703 private:
00704 template< typename T > friend struct Kobject_typeid;
00705 protected:
00706 struct __Kobject_typeid {
00707 typedef Type_info::Demand_union_t<S_DEMAND, BASE_DEMAND> Demand;
00708 static Type_info const *const _b[];
00709 static Type_info const _m;
00710 };
00711 public:
00712 static long const Protocol = PROTO;
00713 typedef L4::Typeid::Rights<Class> Rights;
00714
00743 template<
00744 typename Derived,
00745 typename Base,
00746 long PROTO = PROTO_ANY,
00747 typename S_DEMAND = Type_info::Demand_t<>
00748 >
00749 class Kobject_t : public Base
00750 {
00751 protected:
00753 typedef Derived Class;
00755 typedef Typeid::Iface<PROTO, Derived> __Iface;
00757 typedef Typeid::Merge_list<
00758 Typeid::Iface_list<__Iface>, typename Base::__Iface_list
00759 > __Iface_list;
00760
00762 static void __check_protocols__() noexcept
00763 {
00764 typedef Typeid::Iface_conflict<__Iface, typename Base::__Iface_list> Base_conflict;
00765 static_assert(!Base_conflict::value, "ambiguous protocol ID: protocol also used by Base");
00766 }
00767
00769 L4::Cap<Class> c() const noexcept { return L4::Cap<Class>(this->cap()); }
00770
00771 // Generate the remaining type information
00772 L4___GEN_TI_MEMBERS(typename Base::__Kobject_typeid::Demand)
00773 };
00774
00775
00777 template< typename Derived, typename Base, long PROTO, typename S_DEMAND>
00778 Type_info const *const
00779 Kobject_t<Derived, Base, PROTO, S_DEMAND>::
00780 __Kobject_typeid::_b[] = { &Base::__Kobject_typeid::_m };
00782
00787 template< typename Derived, typename Base, long PROTO, typename S_DEMAND>
00788 L4___GEN_TI(Kobject_t<Derived, Base, PROTO, S_DEMAND>);
00789
00790
00820 template<
00821 typename Derived,
00822 typename Base1,
00823 typename Base2,
00824 long PROTO = PROTO_ANY,
00825 typename S_DEMAND = Type_info::Demand_t<>
00826 >
00827 class Kobject_2t : public Base1, public Base2
00828 {
00829 protected:

```

```

00831 typedef Derived Class;
00833 typedef Typeid::Iface<PROTO, Derived> __Iface;
00835 typedef Typeid::Merge_list<
00836 Typeid::Iface_list<__Iface>,
00837 Typeid::Merge_list<
00838 typename Base1::__Iface_list,
00839 typename Base2::__Iface_list
00840 >
00841 > __Iface_list;
00842
00844 static void __check_protocols__() noexcept
00845 {
00846 typedef typename Base1::__Iface_list Base1_proto_list;
00847 typedef typename Base2::__Iface_list Base2_proto_list;
00848
00849 typedef Typeid::Iface_conflict<__Iface, Base1_proto_list> Base1_conflict;
00850 typedef Typeid::Iface_conflict<__Iface, Base2_proto_list> Base2_conflict;
00851 static_assert(!Base1_conflict::value, "ambiguous protocol ID, also in Base1");
00852 static_assert(!Base2_conflict::value, "ambiguous protocol ID, also in Base2");
00853
00854 typedef Typeid::Conflict<Base1_proto_list, Base2_proto_list> Bases_conflict;
00855 static_assert(!Bases_conflict::value, "ambiguous protocol IDs in base classes");
00856 }
00857
00858 // disambiguate cap()
00859 l4_cap_idx_t cap() const noexcept
00860 { return Base1::cap(); }
00861
00863 L4::Cap<Class> c() const noexcept { return L4::Cap<Class>(this->cap()); }
00864
00865 L4__GEN_TI_MEMBERS(Type_info::Demand_union_t<
00866 typename Base1::__Kobject_typeid::Demand,
00867 typename Base2::__Kobject_typeid::Demand>
00868)
00869 public:
00871 // Provide non-ambiguous conversion to Kobject
00872 operator Kobject const & () const noexcept
00873 { return *static_cast<Base1 const *>(this); }
00874
00875 // Provide non-ambiguous access of dec_refcnt()
00876 l4_msgtag_t dec_refcnt(l4_mword_t diff, l4_utcb_t *utcb = l4_utcb())
00877 noexcept(noexcept(static_cast<Base1*>(nullptr)->dec_refcnt(diff, utcb)))
00878 { return Base1::dec_refcnt(diff, utcb); }
00879 };
00880
00881 template< typename Derived, typename Base1, typename Base2,
00882 long PROTO, typename S_DEMAND >
00885 Type_info const *const
00886 Kobject_2t<Derived, Base1, Base2, PROTO, S_DEMAND>::__Kobject_typeid::b[] =
00887 {
00888 &Base1::__Kobject_typeid::_m,
00889 &Base2::__Kobject_typeid::_m
00890 };
00892 template< typename Derived, typename Base1, typename Base2,
00893 long PROTO, typename S_DEMAND >
00899 L4__GEN_TI(Kobject_2t<Derived, Base1, Base2, PROTO, S_DEMAND>);
00900
00901
00902 template<
00923 typename Derived,
00924 typename Base1,
00925 typename Base2,
00926 typename Base3,
00927 long PROTO = PROTO_ANY,
00928 typename S_DEMAND = Type_info::Demand_t<>
00929 >
00930 struct Kobject_3t : Base1, Base2, Base3
00931 {
00932 protected:
00934 typedef Derived Class;
00936 typedef Typeid::Iface<PROTO, Derived> __Iface;
00938 typedef Typeid::Merge_list<
00939 Typeid::Iface_list<__Iface>,
00940 Typeid::Merge_list<
00941 typename Base1::__Iface_list,
00942 Typeid::Merge_list<
00943 typename Base2::__Iface_list,
00944 typename Base3::__Iface_list
00945 >
00946 >
00947 > __Iface_list;
00948
00950 static void __check_protocols__() noexcept

```

```

00951 {
00952 typedef typename Base1::__Iface_list Base1_proto_list;
00953 typedef typename Base2::__Iface_list Base2_proto_list;
00954 typedef typename Base3::__Iface_list Base3_proto_list;
00955
00956 typedef Typeid::Iface_conflict<__Iface, Base1_proto_list> Base1_conflict;
00957 typedef Typeid::Iface_conflict<__Iface, Base2_proto_list> Base2_conflict;
00958 typedef Typeid::Iface_conflict<__Iface, Base3_proto_list> Base3_conflict;
00959
00960 static_assert(!Base1_conflict::value, "ambiguous protocol ID, also in Base1");
00961 static_assert(!Base2_conflict::value, "ambiguous protocol ID, also in Base2");
00962 static_assert(!Base3_conflict::value, "ambiguous protocol ID, also in Base3");
00963
00964 typedef Typeid::Conflict<Base1_proto_list, Base2_proto_list> Conflict_bases12;
00965 typedef Typeid::Conflict<Base1_proto_list, Base3_proto_list> Conflict_bases13;
00966 typedef Typeid::Conflict<Base2_proto_list, Base3_proto_list> Conflict_bases23;
00967
00968 static_assert(!Conflict_bases12::value, "ambiguous protocol IDs in base classes: Base1 and
Base2");
00969 static_assert(!Conflict_bases13::value, "ambiguous protocol IDs in base classes: Base1 and
Base3");
00970 static_assert(!Conflict_bases23::value, "ambiguous protocol IDs in base classes: Base2 and
Base3");
00971 }
00972
00973 // disambiguate cap()
00974 l4_cap_idx_t cap() const noexcept
00975 { return Base1::cap(); }
00976
00977 L4::Cap<Class> c() const noexcept { return L4::Cap<Class>(this->cap()); }
00978
00979 L4____GEN_TI_MEMBERS(Type_info::Demand_union_t<Type_info::Demand_union_t<
00980 typename Base1::__Kobject_typeid::Demand,
00981 typename Base2::__Kobject_typeid::Demand>,
00982 typename Base3::__Kobject_typeid::Demand>
00983)
00984
00985 public:
00986 // Provide non-ambiguous conversion to Kobject
00987 operator Kobject const & () const noexcept
00988 { return *static_cast<Base1 const *>(this); }
00989
00990 // Provide non-ambiguous access of dec_refcnt()
00991 l4_msgtag_t dec_refcnt(l4_mword_t diff, l4_utcb_t *utcb = l4_utcb())
00992 noexcept(noexcept(static_cast<Base1*>(nullptr)->dec_refcnt(diff, utcb)))
00993 { return Base1::dec_refcnt(diff, utcb); }
00994
00995 };
00996
00997
00998 template< typename Derived, typename Base1, typename Base2, typename Base3,
00999 long PROTO, typename S_DEMAND >
01000 Type_info const *const
01001 Kobject_3t<Derived, Base1, Base2, Base3, PROTO, S_DEMAND>::__Kobject_typeid::__b[] =
01002 {
01003 &Base1::__Kobject_typeid::__m,
01004 &Base2::__Kobject_typeid::__m,
01005 &Base3::__Kobject_typeid::__m
01006 };
01007
01008
01009 template< typename Derived, typename Base1, typename Base2, typename Base3,
01010 long PROTO, typename S_DEMAND >
01011 L4____GEN_TI(Kobject_3t<Derived, Base1, Base2, Base3, PROTO, S_DEMAND>);
01012
01013
01014 #if __cplusplus >= 201103L
01015 namespace L4 {
01016
01017 template< typename ...T >
01018 struct Kobject_demand;
01019
01020 template<>
01021 struct Kobject_demand<> : Type_info::Demand_t<> {};
01022
01023 template<typename T>
01024 struct Kobject_demand<T> : Kobject_typeid<T>::Demand {};
01025
01026 template<typename T1, typename ...T2>
01027 struct Kobject_demand<T1, T2...> :
01028 Type_info::Demand_union_t<typename Kobject_typeid<T1>::Demand,
01029 Kobject_demand<T2...> >
01030 {};
01031
01032 namespace Typeid_xx {
01033
01034 template<typename ...LISTS>

```

```

01048 struct Merge_list;
01049
01050 template<typename L>
01051 struct Merge_list<L> : L {};
01052
01053 template<typename L1, typename L2>
01054 struct Merge_list<L1, L2> : Typeid::Merge_list<L1, L2> {};
01055
01056 template<typename L1, typename L2, typename ...LISTS>
01057 struct Merge_list<L1, L2, LISTS...> :
01058 Merge_list<typename Typeid::Merge_list<L1, L2>::type, LISTS...> {};
01059
01060 template< typename I, typename ...LIST >
01061 struct Iface_conflict;
01062
01063 template< typename I >
01064 struct Iface_conflict<I> : Typeid::False {};
01065
01066 template< typename I, typename L, typename ...LIST >
01067 struct Iface_conflict<I, L, LIST...> :
01068 Typeid::Bool<Typeid::Iface_conflict<typename I::type, typename L::type>::value
01069 || Iface_conflict<I, LIST...>::value>
01070 {};
01071
01072 template< typename ...LIST >
01073 struct Conflict;
01074
01075 template< typename L >
01076 struct Conflict<L> : Typeid::False {};
01077
01078 template< typename L1, typename L2, typename ...LIST >
01079 struct Conflict<L1, L2, LIST...> :
01080 Typeid::Bool<Typeid::Conflict<typename L1::type, typename L2::type>::value
01081 || Conflict<L1, LIST...>::value
01082 || Conflict<L2, LIST...>::value>
01083 {};
01084
01085 template< typename T >
01086 struct Is_demand
01087 {
01088 static long test(Type_info::Demand const *);
01089 static char test(...);
01090 enum { value = sizeof(test(static_cast<T*>(nullptr))) == sizeof(long) };
01091 };
01092
01093 template< typename T, typename ... >
01094 struct First : T { typedef T type; };
01095 } // Typeid
01096
01102 template< typename Derived, long PROTO, typename S_DEMAND, typename ...BASES>
01103 struct __Kobject_base : BASES...
01104 {
01105 protected:
01106 typedef Derived Class;
01107 typedef Typeid::Iface<PROTO, Derived> __Iface;
01108 typedef Typeid_xx::Merge_list<
01109 Typeid::Iface_list<__Iface>,
01110 typename BASES::__Iface_list...
01111 > __Iface_list;
01112
01113 static void __check_protocols__() noexcept
01114 {
01115 typedef Typeid_xx::Iface_conflict<__Iface, typename BASES::__Iface_list...> Conflict;
01116 static_assert(!Conflict::value, "ambiguous protocol ID, protocol also used in base class");
01117
01118 typedef Typeid_xx::Conflict<typename BASES::__Iface_list...> Base_conflict;
01119 static_assert(!Base_conflict::value, "ambiguous protocol IDs in base classes");
01120 }
01121
01122 // disambiguate cap()
01123 l4_cap_idx_t cap() const noexcept
01124 { return Typeid_xx::First<BASES...>::type::cap(); }
01125
01126 L4::Cap<Class> c() const noexcept { return L4::Cap<Class>(this->cap()); }
01127
01128 L4__GEN_TI_MEMBERS(Kobject_demand<BASES...>)
01129
01130 private:
01131 // This function returns the first base class (used below)
01132 template<typename B1, typename ...> struct Basel { typedef B1 type; };
01133
01134 public:
01135 // Provide non-ambiguous conversion to Kobject
01136 operator Kobject const & () const noexcept
01137 { return *static_cast<typename Basel<BASES...>::type const *>(this); }
01138
01139 // Provide non-ambiguous access of dec_refcnt()

```

```

01140 l4_msgtag_t dec_refcnt(l4_mword_t diff, l4_utcb_t *utcb = l4_utcb())
01141 noexcept(noexcept(static_cast<typename Base1<BASES...>::type *>(nullptr)
01142 ->dec_refcnt(diff, utcb)))
01143 { return Base1<BASES...>::type::dec_refcnt(diff, utcb); }
01144 };
01145
01146 template< typename Derived, long PROTO, typename S_DEMAND, typename ...BASES>
01147 Type_info const *const
01148 __Kobject_base<Derived, PROTO, S_DEMAND, BASES...>::__Kobject_typeid::_b[] =
01149 {
01150 (&BASES::__Kobject_typeid::_m)...
01151 };
01152
01153 template< typename Derived, long PROTO, typename S_DEMAND, typename ...BASES>
01154 L4___GEN_TI(__Kobject_base<Derived, PROTO, S_DEMAND, BASES...>);
01155
01156 // Test if the there is a Demand argument to Kobject_x
01157 template< typename Derived, long PROTO, bool HAS_DEMAND, typename DEMAND, typename ...ARGS >
01158 struct __Kobject_x_proto;
01159
01160 // YES: pass it to __Kobject_base
01161 template< typename Derived, long PROTO, typename DEMAND, typename ...BASES>
01162 struct __Kobject_x_proto<Derived, PROTO, true, DEMAND, BASES...> :
01163 __Kobject_base<Derived, PROTO, DEMAND, BASES...> {};
01164
01165 // NO: pass it empty Type_info::Demand_t
01166 template< typename Derived, long PROTO, typename B1, typename ...BASES>
01167 struct __Kobject_x_proto<Derived, PROTO, false, B1, BASES...> :
01168 __Kobject_base<Derived, PROTO, Type_info::Demand_t<>, B1, BASES...> {};
01169
01170 template< long P = PROTO_EMPTY >
01171 struct Proto_t {};
01172
01173 template< typename Derived, typename ...ARGS >
01174 struct Kobject_x;
01175
01176 template< typename Derived, typename A, typename ...ARGS >
01177 struct Kobject_x<Derived, A, ARGS...> :
01178 __Kobject_x_proto<Derived, PROTO_ANY, Typeid_xx::Is_demand<A>::value, A, ARGS...>
01179 {};
01180
01181 template< typename Derived, long PROTO, typename A, typename ...ARGS >
01182 struct Kobject_x<Derived, Proto_t<PROTO>, A, ARGS...> :
01183 __Kobject_x_proto<Derived, PROTO, Typeid_xx::Is_demand<A>::value, A, ARGS...>
01184 {};
01185
01186 }
01187 #endif
01188 #undef L4___GEN_TI
01189 #undef L4___GEN_TI_MEMBERS

```

## 17.445 \_\_vcpu-arm.h

```

00001 /*
00002 * (c) 2017 Alexander Warg <alexander.warg@kernkonzept.com>
00003 *
00004 * License: see LICENSE.spdx (in this directory or the directories above)
00005 */
00006 #pragma once
00007
00008 typedef struct l4_arm_vcpu_e_info_t
00009 {
00010 l4_uint8_t version; // must be 0
00011 l4_uint8_t gic_version;
00012 l4_uint8_t _rsvd0[2];
00013 l4_uint32_t features;
00014 l4_uint32_t _rsvd1[14];
00015 l4_umword_t user[8];
00016 } l4_arm_vcpu_e_info_t;
00017
00018 L4_INLINE void *l4_vcpu_e_ptr(void const *vcpu, unsigned id) L4_NOTHROW;
00019
00020 enum L4_vcpu_e_consts
00021 {
00022 L4_VCPU_E_NUM_LR = 4,
00023 };
00024
00025 L4_INLINE l4_arm_vcpu_e_info_t const *
00026 l4_vcpu_e_info(void const *vcpu) L4_NOTHROW;
00027
00028 L4_INLINE l4_umword_t *

```



```

00029 l4_vcpu_e_info_user(void *vcpu) L4_NOTHROW;
00030
00031 L4_INLINE l4_umword_t *
00032 l4_vcpu_e_info_user(void *vcpu) L4_NOTHROW
00033 {
00034 return ((l4_arm_vcpu_e_info_t *)l4_vcpu_e_info(vcpu))->user;
00035 }
00036
00037
00045 L4_INLINE l4_uint32_t
00046 l4_vcpu_e_read_32(void const *vcpu, unsigned id) L4_NOTHROW;
00047
00048 L4_INLINE l4_uint32_t
00049 l4_vcpu_e_read_32(void const *vcpu, unsigned id) L4_NOTHROW
00050 { return *(l4_uint32_t const *)l4_vcpu_e_ptr(vcpu, id); }
00051
00059 L4_INLINE void
00060 l4_vcpu_e_write_32(void *vcpu, unsigned id, l4_uint32_t val) L4_NOTHROW;
00061
00062 L4_INLINE void
00063 l4_vcpu_e_write_32(void *vcpu, unsigned id, l4_uint32_t val) L4_NOTHROW
00064 { *((l4_uint32_t *)l4_vcpu_e_ptr(vcpu, + id)) = val; }
00065
00073 L4_INLINE l4_uint64_t
00074 l4_vcpu_e_read_64(void const *vcpu, unsigned id) L4_NOTHROW;
00075
00076 L4_INLINE l4_uint64_t
00077 l4_vcpu_e_read_64(void const *vcpu, unsigned id) L4_NOTHROW
00078 { return *(l4_uint64_t const *)l4_vcpu_e_ptr(vcpu, id); }
00079
00087 L4_INLINE void
00088 l4_vcpu_e_write_64(void *vcpu, unsigned id, l4_uint64_t val) L4_NOTHROW;
00089
00090 L4_INLINE void
00091 l4_vcpu_e_write_64(void *vcpu, unsigned id, l4_uint64_t val) L4_NOTHROW
00092 { *((l4_uint64_t *)l4_vcpu_e_ptr(vcpu, id)) = val; }
00093
00101 L4_INLINE l4_umword_t
00102 l4_vcpu_e_read(void const *vcpu, unsigned id) L4_NOTHROW;
00103
00104 L4_INLINE l4_umword_t
00105 l4_vcpu_e_read(void const *vcpu, unsigned id) L4_NOTHROW
00106 { return *(l4_umword_t const *)l4_vcpu_e_ptr(vcpu, id); }
00107
00115 L4_INLINE void
00116 l4_vcpu_e_write(void *vcpu, unsigned id, l4_umword_t val) L4_NOTHROW;
00117
00118 L4_INLINE void
00119 l4_vcpu_e_write(void *vcpu, unsigned id, l4_umword_t val) L4_NOTHROW
00120 { *((l4_umword_t *)l4_vcpu_e_ptr(vcpu, id)) = val; }

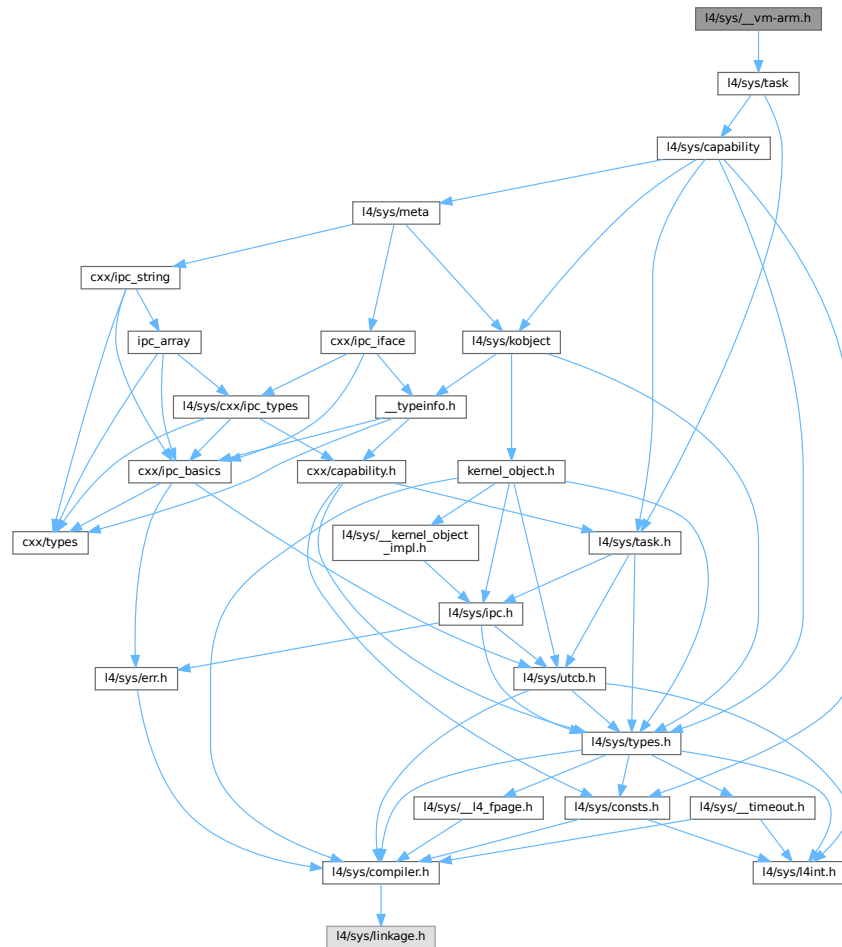
```

## 17.446 l4/sys/\_vm-arm.h File Reference

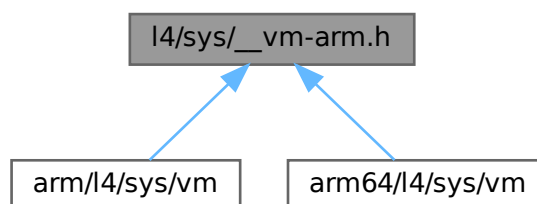
Virtualization interface.

```
#include <l4/sys/task>
```

Include dependency graph for `__vm-arm.h`:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class [L4::Vm](#)

*Virtual machine host address space.*

## Namespaces

- namespace [L4](#)  
*L4 low-level kernel interface.*

## 17.446.1 Detailed Description

Virtualization interface.

Definition in file [\\_\\_vm-arm.h](#).

## 17.447 \_\_vm-arm.h

[Go to the documentation of this file.](#)

```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2018 Adam Lackorzynski <adam@l4re.org>
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #pragma once
00012
00013 #include <l4/sys/task>
00014
00015 namespace L4 {
00016
00017 class Vm : public Kobject_t<Vm, Task, L4_PROTO_VM>
00018 {
00019 public:
00030 l4_msgtag_t vgicc_map(l4_fpage_t const vgicc_fpage,
00031 l4_utcb_t *utcb = l4_utcb()) noexcept
00032 { return l4_task_vgicc_map_u(cap(), vgicc_fpage, utcb); }
00033
00034 protected:
00035 Vm();
00036
00037 private:
00038 Vm(Vm const &);
00039 void operator = (Vm const &);
00040 };
00041
00042 }
```

## 17.448 \_\_vm-svm.h

```
00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #pragma once
00014
00015 #include <l4/sys/types.h>
00016
00022
00023
00028 typedef struct l4_vm_svm_vmcb_control_area
00029 {
00030 l4_uint16_t intercept_rd_crX;
00031 l4_uint16_t intercept_wr_crX;
00032
00033 l4_uint16_t intercept_rd_drX;
00034 l4_uint16_t intercept_wr_drX;
00035
00036 l4_uint32_t intercept_exceptions;
```

```

00037
00038 14_uint32_t intercept_instruction0;
00039 14_uint32_t intercept_instruction1;
00040
00041 14_uint8_t _reserved0[40];
00042
00043 14_uint16_t pause_filter_threshold;
00044 14_uint16_t pause_filter_count;
00045
00046 14_uint64_t iopm_base_pa;
00047 14_uint64_t msrpm_base_pa;
00048 14_uint64_t tsc_offset;
00049 14_uint64_t guest_asid_tlb_ctl;
00050 14_uint64_t interrupt_ctl;
00051 14_uint64_t interrupt_shadow;
00052 14_uint64_t exitcode;
00053 14_uint64_t exitinfo1;
00054 14_uint64_t exitinfo2;
00055 14_uint64_t exitintinfo;
00056 14_uint64_t np_enable;
00057
00058 14_uint8_t _reserved1[16];
00059
00060 14_uint64_t eventinj;
00061 14_uint64_t n_cr3;
00062 14_uint64_t lbr_virtualization_enable;
00063 14_uint64_t clean_bits;
00064 14_uint64_t n_rip;
00065
00066 14_uint8_t _reserved2[816];
00067 } __attribute__((packed)) 14_vm_svm_vmcb_control_area_t;
00068
00073 typedef struct 14_vm_svm_vmcb_state_save_area_seg
00074 {
00075 14_uint16_t selector;
00076 14_uint16_t attrib;
00077 14_uint32_t limit;
00078 14_uint64_t base;
00079 } __attribute__((packed)) 14_vm_svm_vmcb_state_save_area_seg_t;
00080
00085 typedef struct 14_vm_svm_vmcb_state_save_area
00086 {
00087 struct 14_vm_svm_vmcb_state_save_area_seg es;
00088 struct 14_vm_svm_vmcb_state_save_area_seg cs;
00089 struct 14_vm_svm_vmcb_state_save_area_seg ss;
00090 struct 14_vm_svm_vmcb_state_save_area_seg ds;
00091 struct 14_vm_svm_vmcb_state_save_area_seg fs;
00092 struct 14_vm_svm_vmcb_state_save_area_seg gs;
00093 struct 14_vm_svm_vmcb_state_save_area_seg gdtr;
00094 struct 14_vm_svm_vmcb_state_save_area_seg ldtr;
00095 struct 14_vm_svm_vmcb_state_save_area_seg idtr;
00096 struct 14_vm_svm_vmcb_state_save_area_seg tr;
00097
00098 14_uint8_t _reserved0[43];
00099
00100 14_uint8_t cpl;
00101
00102 14_uint32_t _reserved1;
00103
00104 14_uint64_t efer;
00105
00106 14_uint8_t _reserved2[112];
00107
00108 14_uint64_t cr4;
00109 14_uint64_t cr3;
00110 14_uint64_t cr0;
00111 14_uint64_t dr7;
00112 14_uint64_t dr6;
00113 14_uint64_t rflags;
00114 14_uint64_t rip;
00115
00116 14_uint8_t _reserved3[88];
00117
00118 14_uint64_t rsp;
00119
00120 14_uint8_t _reserved4[24];
00121
00122 14_uint64_t rax;
00123 14_uint64_t star;
00124 14_uint64_t lstar;
00125 14_uint64_t cstar;
00126 14_uint64_t sfmask;
00127 14_uint64_t kernelgsbase;
00128 14_uint64_t sysenter_cs;
00129 14_uint64_t sysenter_esp;
00130 14_uint64_t sysenter_eip;
00131 14_uint64_t cr2;

```

```

00132
00133 l4_uint8_t _reserved5[32];
00134
00135 l4_uint64_t g_pat;
00136 l4_uint64_t dbgctl;
00137 l4_uint64_t br_from;
00138 l4_uint64_t br_to;
00139 l4_uint64_t lastexcpfrom;
00140 l4_uint64_t last_excpto;
00141
00142 // this field is _NOT_ part of the official VMCB specification
00143 // a (userlevel) VMM needs this for proper FPU state virtualization
00144 l4_uint64_t xcr0;
00145
00146 l4_uint8_t _reserved6[2400];
00147 } __attribute__((packed)) l4_vm_svm_vmc_b_state_save_area_t;
00148
00149
00154 typedef struct l4_vm_svm_vmc_b_t
00155 {
00156 l4_vm_svm_vmc_b_control_area_t control_area;
00157 l4_vm_svm_vmc_b_state_save_area_t state_save_area;
00158 } l4_vm_svm_vmc_b_t;

```

## 17.449 \_\_vm-vmx.h

```

00001
00006 /*
00007 * (c) 2010-2013 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013
00014 #pragma once
00015
00016 #include <l4/sys/vcpu.h>
00017
00023
00028 enum l4_vm_vmx_caps_regs
00029 {
00030 L4_VM_VMX_BASIC_REG = 0,
00031 L4_VM_VMX_TRUE_PINBASED_CTLs_REG = 1,
00032 L4_VM_VMX_TRUE_PROCBASED_CTLs_REG = 2,
00033 L4_VM_VMX_TRUE_EXIT_CTLs_REG = 3,
00034 L4_VM_VMX_TRUE_ENTRY_CTLs_REG = 4,
00035 L4_VM_VMX_MISC_REG = 5,
00036 L4_VM_VMX_CR0_FIXED0_REG = 6,
00037 L4_VM_VMX_CR0_FIXED1_REG = 7,
00038 L4_VM_VMX_CR4_FIXED0_REG = 8,
00039 L4_VM_VMX_CR4_FIXED1_REG = 9,
00040 L4_VM_VMX_VMCS_ENUM_REG = 10,
00041 L4_VM_VMX_PROCBASED_CTLs2_REG = 11,
00042 L4_VM_VMX_EPT_VPID_CAP_REG = 12,
00043 L4_VM_VMX_NESTED_REVISION = 13,
00044 L4_VM_VMX_NUM_CAPS_REGS
00045 };
00046
00051 enum l4_vm_vmx_dfl1_regs
00052 {
00053 L4_VM_VMX_PINBASED_CTLs_DFL1_REG = 0,
00054 L4_VM_VMX_PROCBASED_CTLs_DFL1_REG = 1,
00055 L4_VM_VMX_EXIT_CTLs_DFL1_REG = 2,
00056 L4_VM_VMX_ENTRY_CTLs_DFL1_REG = 3,
00057 L4_VM_VMX_NUM_DFL1_REGS
00058 };
00059
00069 enum l4_vm_vmx_sw_fields
00070 {
00078 L4_VM_VMX_VMCS_CR2 = 0x6880,
00080 L4_VM_VMX_VMCS_NAT_ARG0 = 0x6882,
00082 L4_VM_VMX_VMCS_NAT_ARG1 = 0x6884,
00084 L4_VM_VMX_VMCS_NAT_ARG2 = 0x6886,
00086 L4_VM_VMX_VMCS_NAT_ARG3 = 0x6888,
00088 L4_VM_VMX_VMCS_XCR0 = 0x2880,
00090 L4_VM_VMX_VMCS_MSR_SYSCALL_MASK = 0x2882,
00092 L4_VM_VMX_VMCS_MSR_LSTAR = 0x2884,
00094 L4_VM_VMX_VMCS_MSR_CSTAR = 0x2886,
00096 L4_VM_VMX_VMCS_MSR_TSC_AUX = 0x2888,
00098 L4_VM_VMX_VMCS_MSR_STAR = 0x288a,
00100 L4_VM_VMX_VMCS_MSR_KERNEL_GS_BASE = 0x288c,
00101 };

```

```

00102
00155 typedef struct l4_vmx_offset_table_t
00156 {
00157 l4_uint8_t offsets[4][4];
00158 l4_uint8_t limits[4][4];
00159 l4_uint8_t index_shifts[4];
00160 l4_uint8_t base_offset;
00161 l4_uint8_t size;
00162
00163 l4_uint8_t reserved[2];
00164 } l4_vmx_offset_table_t;
00165
00170 enum L4_vm_vmx_vmcs_sizes
00171 {
00173 L4_VM_VMX_VMCS_SIZE_VALUES = 2560,
00175 L4_VM_VMX_VMCS_SIZE_DIRTY_BITMAP = 320,
00176 };
00177
00205 typedef struct l4_vm_vmx_vcpu_vmcs_t
00206 {
00207 l4_uint64_t reserved0;
00208
00209 l4_uint64_t user_data;
00210 l4_uint32_t cr2_index;
00211 l4_uint8_t reserved1[4];
00212
00213 l4_cap_idx_t vmcs;
00214
00215 /*
00216 * Since the capability type size depends on the platform, we add a 32-bit
00217 * padding if necessary.
00218 */
00219
00220 #if L4_MWORD_BITS == 32
00221 l4_uint32_t padding0;
00222 #elif L4_MWORD_BITS == 64
00223 /* No padding needed. */
00224 #else
00225 #error Unsupported machine word size.
00226 #endif
00227
00228 l4_vmx_offset_table_t offset_table;
00229 l4_uint8_t reserved2[120];
00230
00231 l4_uint8_t values[L4_VM_VMX_VMCS_SIZE_VALUES];
00232 l4_uint8_t dirty_bitmap[L4_VM_VMX_VMCS_SIZE_DIRTY_BITMAP];
00233 } l4_vm_vmx_vcpu_vmcs_t;
00234
00239 typedef struct l4_vm_vmx_vcpu_infos_t
00240 {
00242 l4_uint64_t caps[L4_VM_VMX_NUM_CAPS_REGS];
00243
00246 l4_uint32_t dfll[L4_VM_VMX_NUM_DFL1_REGS];
00247 } l4_vm_vmx_vcpu_infos_t;
00248
00267 typedef struct l4_vm_vmx_vcpu_state_t
00268 {
00269 l4_vcpu_state_t vcpu_state;
00270 l4_uint8_t padding0[L4_VCPU_OFFSET_EXT_INFOS - sizeof(l4_vcpu_state_t)];
00271
00272 l4_vm_vmx_vcpu_infos_t infos;
00273 l4_uint8_t padding1[L4_VCPU_OFFSET_EXT_STATE - L4_VCPU_OFFSET_EXT_INFOS
00274 - sizeof(l4_vm_vmx_vcpu_infos_t)];
00275
00276 l4_vm_vmx_vcpu_vmcs_t vmcs;
00277 } l4_vm_vmx_vcpu_state_t;
00278
00288 L4_INLINE
00289 l4_uint64_t
00290 l4_vm_vmx_get_caps(l4_vm_vmx_vcpu_state_t const *vcpu_state,
00291 enum L4_vm_vmx_caps_regs caps_reg) L4_NOTHROW;
00292
00303 L4_INLINE
00304 l4_uint32_t
00305 l4_vm_vmx_get_caps_default1(l4_vm_vmx_vcpu_state_t const *vcpu_state,
00306 enum L4_vm_vmx_dfll_regs dfll_reg) L4_NOTHROW;
00307
00315 L4_INLINE
00316 unsigned
00317 l4_vm_vmx_field_len(unsigned field) L4_NOTHROW;
00318
00326 L4_INLINE
00327 unsigned
00328 l4_vm_vmx_field_order(unsigned field) L4_NOTHROW;
00329
00344 L4_INLINE
00345 void *

```

```

00346 l4_vm_vmx_field_ptr(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field) L4_NOTHROW;
00347
00357 L4_INLINE
00358 void
00359 l4_vm_vmx_clear(l4_vm_vmx_vcpu_vmcs_t *vmcs,
00360 l4_vm_vmx_vcpu_vmcs_t *dest_vmcs) L4_NOTHROW;
00361
00371 L4_INLINE
00372 void
00373 l4_vm_vmx_ptr_load(l4_vm_vmx_vcpu_vmcs_t *vmcs,
00374 l4_vm_vmx_vcpu_vmcs_t *src_vmcs) L4_NOTHROW;
00375
00391 L4_INLINE
00392 l4_uint32_t
00393 l4_vm_vmx_get_cr2_index(l4_vm_vmx_vcpu_vmcs_t const *vmcs) L4_NOTHROW;
00394
00404 L4_INLINE
00405 l4_umword_t
00406 l4_vm_vmx_read_nat(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field) L4_NOTHROW;
00407
00417 L4_INLINE
00418 l4_uint16_t
00419 l4_vm_vmx_read_16(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field) L4_NOTHROW;
00420
00430 L4_INLINE
00431 l4_uint32_t
00432 l4_vm_vmx_read_32(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field) L4_NOTHROW;
00433
00443 L4_INLINE
00444 l4_uint64_t
00445 l4_vm_vmx_read_64(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field) L4_NOTHROW;
00446
00456 L4_INLINE
00457 l4_uint64_t
00458 l4_vm_vmx_read(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field) L4_NOTHROW;
00459
00468 L4_INLINE
00469 void
00470 l4_vm_vmx_write_nat(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field,
00471 l4_umword_t val) L4_NOTHROW;
00472
00481 L4_INLINE
00482 void
00483 l4_vm_vmx_write_16(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field,
00484 l4_uint16_t val) L4_NOTHROW;
00485
00494 L4_INLINE
00495 void
00496 l4_vm_vmx_write_32(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field,
00497 l4_uint32_t val) L4_NOTHROW;
00498
00507 L4_INLINE
00508 void
00509 l4_vm_vmx_write_64(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field,
00510 l4_uint64_t val) L4_NOTHROW;
00511
00520 L4_INLINE
00521 void
00522 l4_vm_vmx_write(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field,
00523 l4_uint64_t val) L4_NOTHROW;
00524
00571 L4_INLINE
00572 void
00573 l4_vm_vmx_set_hw_vmcs(l4_vm_vmx_vcpu_vmcs_t *vmcs,
00574 l4_cap_idx_t vmcs_cap) L4_NOTHROW;
00575
00585 L4_INLINE
00586 l4_cap_idx_t
00587 l4_vm_vmx_get_hw_vmcs(l4_vm_vmx_vcpu_vmcs_t *vmcs) L4_NOTHROW;
00588
00589 /* Implementations */
00590
00591 L4_INLINE
00592 unsigned
00593 l4_vm_vmx_field_len(unsigned field) L4_NOTHROW
00594 {
00595 return 1 « l4_vm_vmx_field_order(field);
00596 }
00597
00598 L4_INLINE
00599 unsigned
00600 l4_vm_vmx_field_order(unsigned field) L4_NOTHROW
00601 {
00602 unsigned size = (field » 13) & 0x03U;
00603
00604 switch (size)
00605 {

```

```

00606 case 0: return 1; /* 16 bits */
00607 case 1: return 3; /* 64 bits */
00608 case 2: return 2; /* 32 bits */
00609 case 3: return (sizeof(l4_umword_t) == 8) ? 3 : 2; /* Natural width */
00610 }
00611
00612 __builtin_trap();
00613 }
00614
00620 L4_INLINE
00621 unsigned
00622 l4_vm_vmx_field_offset(l4_vm_vmx_vcpu_vmcs_t const *vmcs,
00623 unsigned field) L4_NOTHROW
00624 {
00625 unsigned index = field & 0x3feU;
00626 unsigned size = (field >> 13) & 0x03U;
00627 unsigned group = (field >> 10) & 0x03U;
00628
00629 unsigned shifted_index = index << vmcs->offset_table.index_shifts[size];
00630
00631 if (shifted_index >= (unsigned)vmcs->offset_table.limits[size][group] * 64)
00632 return ~0U;
00633
00634 return (unsigned)vmcs->offset_table.offsets[size][group] * 64
00635 + shifted_index;
00636 }
00637
00638 L4_INLINE
00639 void *
00640 l4_vm_vmx_field_ptr(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field) L4_NOTHROW
00641 {
00642 unsigned offset = l4_vm_vmx_field_offset(vmcs, field);
00643 if (offset == ~0U)
00644 return 0;
00645
00646 return (void *) (vmcs->values + offset);
00647 }
00648
00654 L4_INLINE
00655 void *
00656 l4_vm_vmx_field_ptr_offset(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field,
00657 unsigned *offset) L4_NOTHROW
00658 {
00659 *offset = l4_vm_vmx_field_offset(vmcs, field);
00660 if (*offset == ~0U)
00661 return 0;
00662
00663 return (void *) (vmcs->values + *offset);
00664 }
00665
00671 L4_INLINE
00672 void
00673 l4_vm_vmx_offset_dirty(l4_vm_vmx_vcpu_vmcs_t *vmcs,
00674 unsigned offset) L4_NOTHROW
00675 {
00676 vmcs->dirty_bitmap[offset / 8] |= 1U << (offset % 8);
00677 }
00678
00683 L4_INLINE
00684 void
00685 l4_vm_vmx_copy_values(l4_vm_vmx_vcpu_vmcs_t const *vmcs, l4_uint8_t *_dst,
00686 l4_uint8_t const *_src) L4_NOTHROW
00687 {
00688 unsigned base_offset = vmcs->offset_table.base_offset * 64;
00689 unsigned size = vmcs->offset_table.size * 64;
00690
00691 void *dst = _dst + base_offset;
00692 void const *src = _src + base_offset;
00693 __builtin_memcpy(dst, src, size);
00694 }
00695
00696 L4_INLINE
00697 void
00698 l4_vm_vmx_clear(l4_vm_vmx_vcpu_vmcs_t *vmcs,
00699 l4_vm_vmx_vcpu_vmcs_t *dest_vmcs) L4_NOTHROW
00700 {
00701 l4_vm_vmx_vcpu_vmcs_t **current_vmcs_ptr
00702 = (l4_vm_vmx_vcpu_vmcs_t **)&vmcs->user_data;
00703
00704 if (*current_vmcs_ptr != dest_vmcs)
00705 return;
00706
00707 l4_vm_vmx_set_hw_vmcs(dest_vmcs, l4_vm_vmx_get_hw_vmcs(vmcs));
00708 l4_vm_vmx_copy_values(vmcs, dest_vmcs->values, vmcs->values);
00709
00710 /* Due to its size, the dirty bitmap is always copied in its entirety. */
00711 __builtin_memcpy(dest_vmcs->dirty_bitmap, vmcs->dirty_bitmap,

```



```

00712 L4_VM_VMX_VMCS_SIZE_DIRTY_BITMAP);
00713
00714 *current_vmcs_ptr = 0;
00715 }
00716
00717 L4_INLINE
00718 void
00719 l4_vm_vmx_ptr_load(l4_vm_vmx_vcpu_vmcs_t *vmcs,
00720 l4_vm_vmx_vcpu_vmcs_t *src_vmcs) L4_NOTHROW
00721 {
00722 l4_vm_vmx_vcpu_vmcs_t **current_vmcs_ptr
00723 = (l4_vm_vmx_vcpu_vmcs_t **)&vmcs->user_data;
00724
00725 if (*current_vmcs_ptr == src_vmcs)
00726 return;
00727
00728 if (*current_vmcs_ptr && *current_vmcs_ptr != src_vmcs)
00729 l4_vm_vmx_clear(vmcs, *current_vmcs_ptr);
00730
00731 *current_vmcs_ptr = src_vmcs;
00732
00733 l4_vm_vmx_set_hw_vmcs(vmcs, l4_vm_vmx_get_hw_vmcs(src_vmcs));
00734 l4_vm_vmx_copy_values(vmcs, vmcs->values, src_vmcs->values);
00735
00736 /* Due to its size, the dirty bitmap is always compiled in its entirety. */
00737 __builtin_memcpy(vmcs->dirty_bitmap, src_vmcs->dirty_bitmap,
00738 L4_VM_VMX_VMCS_SIZE_DIRTY_BITMAP);
00739 }
00740
00741 L4_INLINE
00742 l4_umword_t
00743 l4_vm_vmx_read_nat(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field) L4_NOTHROW
00744 {
00745 l4_umword_t *ptr = (l4_umword_t *)l4_vm_vmx_field_ptr(vmcs, field);
00746 if (!ptr)
00747 return 0;
00748
00749 return *ptr;
00750 }
00751
00752 L4_INLINE
00753 l4_uint16_t
00754 l4_vm_vmx_read_16(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field) L4_NOTHROW
00755 {
00756 l4_uint16_t *ptr = (l4_uint16_t *)l4_vm_vmx_field_ptr(vmcs, field);
00757 if (!ptr)
00758 return 0;
00759
00760 return *ptr;
00761 }
00762
00763 L4_INLINE
00764 l4_uint32_t
00765 l4_vm_vmx_read_32(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field) L4_NOTHROW
00766 {
00767 l4_uint32_t *ptr = (l4_uint32_t *)l4_vm_vmx_field_ptr(vmcs, field);
00768 if (!ptr)
00769 return 0;
00770
00771 return *ptr;
00772 }
00773
00774 L4_INLINE
00775 l4_uint64_t
00776 l4_vm_vmx_read_64(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field) L4_NOTHROW
00777 {
00778 l4_uint64_t *ptr = (l4_uint64_t *)l4_vm_vmx_field_ptr(vmcs, field);
00779 if (!ptr)
00780 return 0;
00781
00782 return *ptr;
00783 }
00784
00785 L4_INLINE
00786 l4_uint64_t
00787 l4_vm_vmx_read(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field) L4_NOTHROW
00788 {
00789 unsigned size = (field >> 13) & 0x03U;
00790
00791 switch (size)
00792 {
00793 case 0: return l4_vm_vmx_read_16(vmcs, field);
00794 case 1: return l4_vm_vmx_read_64(vmcs, field);
00795 case 2: return l4_vm_vmx_read_32(vmcs, field);
00796 case 3: return l4_vm_vmx_read_nat(vmcs, field);
00797 }
00798 }

```

```

00799 __builtin_trap();
00800 }
00801
00802 L4_INLINE
00803 void
00804 l4_vm_vmx_write_nat(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field,
00805 l4_umword_t val) L4_NOTHROW
00806 {
00807 unsigned offset;
00808 l4_umword_t *ptr
00809 = (l4_umword_t *)l4_vm_vmx_field_ptr_offset(vmcs, field, &offset);
00810
00811 if ((ptr) && (*ptr != val))
00812 {
00813 *ptr = val;
00814 l4_vm_vmx_offset_dirty(vmcs, offset);
00815 }
00816 }
00817
00818 L4_INLINE
00819 void
00820 l4_vm_vmx_write_16(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field,
00821 l4_uint16_t val) L4_NOTHROW
00822 {
00823 unsigned offset;
00824 l4_uint16_t *ptr
00825 = (l4_uint16_t *)l4_vm_vmx_field_ptr_offset(vmcs, field, &offset);
00826
00827 if ((ptr) && (*ptr != val))
00828 {
00829 *ptr = val;
00830 l4_vm_vmx_offset_dirty(vmcs, offset);
00831 }
00832 }
00833
00834 L4_INLINE
00835 void
00836 l4_vm_vmx_write_32(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field,
00837 l4_uint32_t val) L4_NOTHROW
00838 {
00839 unsigned offset;
00840 l4_uint32_t *ptr
00841 = (l4_uint32_t *)l4_vm_vmx_field_ptr_offset(vmcs, field, &offset);
00842
00843 if ((ptr) && (*ptr != val))
00844 {
00845 *ptr = val;
00846 l4_vm_vmx_offset_dirty(vmcs, offset);
00847 }
00848 }
00849
00850 L4_INLINE
00851 void
00852 l4_vm_vmx_write_64(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field,
00853 l4_uint64_t val) L4_NOTHROW
00854 {
00855 unsigned offset;
00856 l4_uint64_t *ptr
00857 = (l4_uint64_t *)l4_vm_vmx_field_ptr_offset(vmcs, field, &offset);
00858
00859 if ((ptr) && (*ptr != val))
00860 {
00861 *ptr = val;
00862 l4_vm_vmx_offset_dirty(vmcs, offset);
00863 }
00864 }
00865
00866 L4_INLINE
00867 void
00868 l4_vm_vmx_write(l4_vm_vmx_vcpu_vmcs_t *vmcs, unsigned field,
00869 l4_uint64_t val) L4_NOTHROW
00870 {
00871 unsigned size = (field >> 13) & 0x03U;
00872
00873 switch (size)
00874 {
00875 case 0: l4_vm_vmx_write_16(vmcs, field, val); break;
00876 case 1: l4_vm_vmx_write_64(vmcs, field, val); break;
00877 case 2: l4_vm_vmx_write_32(vmcs, field, val); break;
00878 case 3: l4_vm_vmx_write_nat(vmcs, field, val); break;
00879 }
00880 }
00881
00882 L4_INLINE
00883 l4_uint64_t
00884 l4_vm_vmx_get_caps(l4_vm_vmx_vcpu_state_t const *vcpu_state,
00885 enum L4_vm_vmx_caps_regs caps_reg) L4_NOTHROW

```

```

00886 {
00887 return vcpu_state->infos.caps[caps_reg];
00888 }
00889
00890 L4_INLINE
00891 l4_uint32_t
00892 l4_vm_vmx_get_caps_default1(l4_vm_vmx_vcpu_state_t const *vcpu_state,
00893 enum l4_vm_vmx_dfll_regs dfll_reg) L4_NOTHROW
00894 {
00895 return vcpu_state->infos.dfll[dfll_reg];
00896 }
00897
00898 L4_INLINE
00899 l4_uint32_t
00900 l4_vm_vmx_get_cr2_index(l4_vm_vmx_vcpu_vmcs_t const *vmcs) L4_NOTHROW
00901 {
00902 return vmcs->cr2_index;
00903 }
00904
00905 L4_INLINE
00906 void
00907 l4_vm_vmx_set_hw_vmcs(l4_vm_vmx_vcpu_vmcs_t *vmcs,
00908 l4_cap_idx_t vmcs_cap) L4_NOTHROW
00909 {
00910 vmcs->vmcs = vmcs_cap;
00911 }
00912
00913 L4_INLINE
00914 l4_cap_idx_t
00915 l4_vm_vmx_get_hw_vmcs(l4_vm_vmx_vcpu_vmcs_t *vmcs) L4_NOTHROW
00916 {
00917 return vmcs->vmcs & L4_CAP_MASK;
00918 }

```

## 17.450 l4/sys/arm\_smccc File Reference

ARM secure monitor call functions.

```

#include <l4/sys/capability>
#include <l4/sys/cxx/ipc_iface>

```



## 17.451 arm\_smccc

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * Copyright (C) 2018, 2022, 2024 Kernkonzept GmbH.
00004 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00012 #pragma once
00013
00014 #include <l4/sys/capability>
00015 #include <l4/sys/cxx/ipc_iface>
00016
00017 namespace L4 {
00018
00023 class L4_EXPORT Arm_smccc : public Kobject_0t<Arm_smccc, L4_PROTO_SMCCC>
00024 {
00025 public:
00064 L4_INLINE_RPC(l4_msgtag_t, call,
00065 (l4_umword_t func, l4_umword_t in0, l4_umword_t in1,
00066 l4_umword_t in2, l4_umword_t in3, l4_umword_t in4,
00067 l4_umword_t in5, l4_umword_t *out0, l4_umword_t *out1,
00068 l4_umword_t *out2, l4_umword_t *out3,
00069 l4_umword_t client_id));
00070
00071 typedef L4::Typeid::Rpc_nocode<call_t> Rpccs;
00072 };
00073
00074 }

```

## 17.452 l4/sys/arm\_smccc.h File Reference

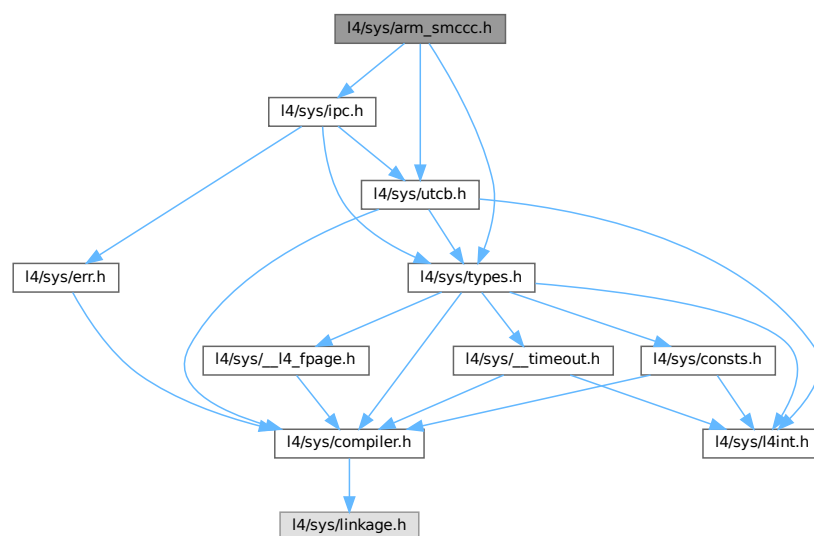
ARM secure monitor call functions.

```

#include <l4/sys/types.h>
#include <l4/sys/utcb.h>
#include <l4/sys/ipc.h>

```

Include dependency graph for arm\_smccc.h:



## Functions

- `l4_msgtag_t l4_arm_smccc_call (l4_cap_idx_t pfc, l4_umword_t func, l4_umword_t in0, l4_umword_t in1, l4_umword_t in2, l4_umword_t in3, l4_umword_t in4, l4_umword_t in5, l4_umword_t *out0, l4_umword_t *out1, l4_umword_t *out2, l4_umword_t *out3, l4_umword_t client_id) L4_NOTHROW`

C interface for calling the ARM secure monitor, see [L4::Arm\\_smccc::call\(\)](#) for the C++ interface.

### 17.452.1 Detailed Description

ARM secure monitor call functions.

Definition in file [arm\\_smccc.h](#).

### 17.452.2 Function Documentation

#### 17.452.2.1 l4\_arm\_smccc\_call()

```
l4_msgtag_t l4_arm_smccc_call (
 l4_cap_idx_t pfc,
 l4_umword_t func,
 l4_umword_t in0,
 l4_umword_t in1,
 l4_umword_t in2,
 l4_umword_t in3,
 l4_umword_t in4,
 l4_umword_t in5,
 l4_umword_t * out0,
 l4_umword_t * out1,
 l4_umword_t * out2,
 l4_umword_t * out3,
 l4_umword_t client_id) [inline]
```

C interface for calling the ARM secure monitor, see [L4::Arm\\_smccc::call\(\)](#) for the C++ interface.

## Parameters

|  |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <i>pfc</i>  | Capability of the SMC kernel object. The input parameters consist of a function identifier, 6 arguments and a client id. Results are returned in 4 output parameters.                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|  | <i>func</i> | Function identifier. <ul style="list-style-type: none"> <li>• Bit 31 has to be set: This marks the call as <i>Fast Call</i>. <i>Yielding Calls</i> (bit 31 unset) are rejected by the kernel.</li> <li>• Bit 30 defines the calling convention:</li> <li>• Bit 30 == 1: 64-bit calling convention.</li> <li>• Bit 30 == 0: 32-bit calling convention.</li> <li>• Bits 24..29 determine the service call ID. The permitted IDs are set in the kernel configuration. By default only service IDs &gt;= 0x30000000 (<i>Trusted Application Calls</i> and <i>Trusted OS Calls</i>) are allowed.</li> </ul> |

|     |                  |                                                                                              |
|-----|------------------|----------------------------------------------------------------------------------------------|
| in  | <i>in0</i>       | First input parameter.                                                                       |
| in  | <i>in1</i>       | Second input parameter.                                                                      |
| in  | <i>in2</i>       | Third input parameter.                                                                       |
| in  | <i>in3</i>       | Fourth input parameter.                                                                      |
| in  | <i>in4</i>       | Fifth input parameter.                                                                       |
| in  | <i>in5</i>       | Sixth input parameter.                                                                       |
| out | <i>out0</i>      | First output parameter.                                                                      |
| out | <i>out1</i>      | Second output parameter.                                                                     |
| out | <i>out2</i>      | Third output parameter.                                                                      |
| out | <i>out3</i>      | Fourth output parameter.                                                                     |
| in  | <i>client_id</i> | Client ID. According to the specification, this value might be ignored by certain functions. |

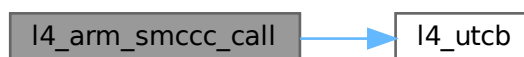
### Return values

|                         |                                                                                                               |
|-------------------------|---------------------------------------------------------------------------------------------------------------|
| <code>-L4_ENOSYS</code> | Either bit 31 of the function call not set or service ID outside the range permitted by kernel configuration. |
| <code>-L4_EINVAL</code> | Invalid number of parameters.                                                                                 |
| <code>&lt; 0</code>     | Other <a href="#">L4</a> error.                                                                               |
| <code>0</code>          | Success.                                                                                                      |

Definition at line 42 of file [arm\\_smccc.h](#).

References [L4\\_NOTHROW](#), and [l4\\_utcb\(\)](#).

Here is the call graph for this function:



## 17.453 arm\_smccc.h

[Go to the documentation of this file.](#)

```
00001 /*
00002 * Copyright (C) 2018, 2022, 2024 Kernkonzept GmbH.
00003 * Author(s): Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00011 #pragma once
00012
00013 #include <l4/sys/types.h>
00014 #include <l4/sys/utcb.h>
```

```

00015
00016 L4_INLINE l4_msgtag_t
00017 l4_arm_smccc_call(l4_cap_idx_t pfc, l4_umword_t func, l4_umword_t in0,
00018 l4_umword_t in1, l4_umword_t in2, l4_umword_t in3,
00019 l4_umword_t in4, l4_umword_t in5, l4_umword_t *out0,
00020 l4_umword_t *out1, l4_umword_t *out2, l4_umword_t *out3,
00021 l4_umword_t client_id) L4_NOTHROW;
00022
00023 L4_INLINE l4_msgtag_t
00024 l4_arm_smccc_call_u(l4_cap_idx_t pfc, l4_umword_t func, l4_umword_t in0,
00025 l4_umword_t in1, l4_umword_t in2, l4_umword_t in3,
00026 l4_umword_t in4, l4_umword_t in5, l4_umword_t *out0,
00027 l4_umword_t *out1, l4_umword_t *out2, l4_umword_t *out3,
00028 l4_umword_t client_id, l4_utcb_t *utcb) L4_NOTHROW;
00029
00030 /* IMPLEMENTATION ----- */
00031
00032 #include <l4/sys/ipc.h>
00033
00041 L4_INLINE l4_msgtag_t
00042 l4_arm_smccc_call(l4_cap_idx_t pfc, l4_umword_t func,
00043 l4_umword_t in0, l4_umword_t in1,
00044 l4_umword_t in2, l4_umword_t in3,
00045 l4_umword_t in4, l4_umword_t in5,
00046 l4_umword_t *out0, l4_umword_t *out1,
00047 l4_umword_t *out2, l4_umword_t *out3,
00048 l4_umword_t client_id) L4_NOTHROW
00049 {
00050 return l4_arm_smccc_call_u(pfc, func, in0, in1, in2, in3, in4, in5,
00051 out0, out1, out2, out3, client_id, l4_utcb());
00052 }
00053
00054
00055 L4_INLINE l4_msgtag_t
00056 l4_arm_smccc_call_u(l4_cap_idx_t pfc, l4_umword_t func, l4_umword_t in0,
00057 l4_umword_t in1, l4_umword_t in2, l4_umword_t in3,
00058 l4_umword_t in4, l4_umword_t in5, l4_umword_t *out0,
00059 l4_umword_t *out1, l4_umword_t *out2, l4_umword_t *out3,
00060 l4_umword_t client_id, l4_utcb_t *utcb) L4_NOTHROW
00061 {
00062 l4_msgtag_t ret;
00063 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00064 v->mr[0] = func;
00065 v->mr[1] = in0;
00066 v->mr[2] = in1;
00067 v->mr[3] = in2;
00068 v->mr[4] = in3;
00069 v->mr[5] = in4;
00070 v->mr[6] = in5;
00071 v->mr[7] = client_id;
00072
00073 ret = l4_ipc_call(pfc, utcb, l4_msgtag(L4_PROTO_SMCCC, 8, 0, 0),
00074 L4_IPC_NEVER);
00075
00076 if (l4_error(ret) >= 0)
00077 {
00078 *out0 = v->mr[0];
00079 *out1 = v->mr[1];
00080 *out2 = v->mr[2];
00081 *out3 = v->mr[3];
00082 }
00083
00084 return ret;
00085 }

```

## 17.454 amd64/l4/sys/cache.h File Reference

Cache functions.

### Functions

- int [l4\\_cache\\_clean\\_data](#) (unsigned long start, unsigned long end) [L4\\_NOTHROW](#)  
Cache clean a range in D-cache; writes back to PoC.
- int [l4\\_cache\\_flush\\_data](#) (unsigned long start, unsigned long end) [L4\\_NOTHROW](#)  
Cache flush a range; writes back to PoC.



- int [l4\\_cache\\_inv\\_data](#) (unsigned long start, unsigned long end) [L4\\_NOTHROW](#)  
*Cache invalidate a range; might write back to PoC.*
- int [l4\\_cache\\_coherent](#) (unsigned long start, unsigned long end) [L4\\_NOTHROW](#)  
*Make memory coherent between I-cache and D-cache; writes back to PoU.*
- int [l4\\_cache\\_dma\\_coherent](#) (unsigned long start, unsigned long end) [L4\\_NOTHROW](#)  
*Make memory coherent for use with external memory; writes back to PoC.*
- int [l4\\_cache\\_dma\\_coherent\\_full](#) (void) [L4\\_NOTHROW](#)  
*Make memory coherent for use with external memory; writes back to PoC.*

### 17.454.1 Detailed Description

Cache functions.

Definition in file [cache.h](#).

## 17.455 cache.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #ifndef __L4SYS__INCLUDE__ARCH_AMD64__CACHE_H__
00012 #define __L4SYS__INCLUDE__ARCH_AMD64__CACHE_H__
00013
00014 #include_next <l4/sys/cache.h>
00015
00016 L4_INLINE int
00017 l4_cache_clean_data(unsigned long start,
00018 unsigned long end) L4_NOTHROW
00019 {
00020 (void)start; (void)end;
00021 return 0;
00022 }
00023
00024 L4_INLINE int
00025 l4_cache_flush_data(unsigned long start,
00026 unsigned long end) L4_NOTHROW
00027 {
00028 (void)start; (void)end;
00029 return 0;
00030 }
00031
00032 L4_INLINE int
00033 l4_cache_inv_data(unsigned long start,
00034 unsigned long end) L4_NOTHROW
00035 {
00036 (void)start; (void)end;
00037 return 0;
00038 }
00039
00040 L4_INLINE int
00041 l4_cache_coherent(unsigned long start,
00042 unsigned long end) L4_NOTHROW
00043 {
00044 (void)start; (void)end;
00045 return 0;
00046 }
00047
00048 L4_INLINE int
00049 l4_cache_dma_coherent(unsigned long start,
00050 unsigned long end) L4_NOTHROW
00051 {
00052 (void)start; (void)end;
00053 return 0;
00054 }
00055

```

```

00056 L4_INLINE int
00057 l4_cache_dma_coherent_full(void) L4_NOTHROW
00058 {
00059 return 0;
00060 }
00061
00062 #endif /* ! __L4SYS__INCLUDE__ARCH_AMD64__CACHE_H__ */

```

## 17.456 arm/l4/sys/cache.h File Reference

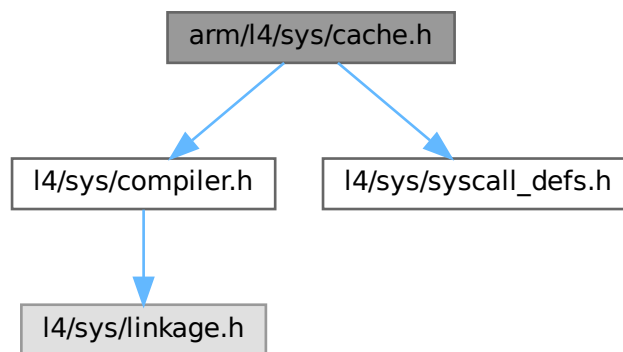
Cache functions.

```

#include <l4/sys/compiler.h>
#include <l4/sys/syscall_defs.h>

```

Include dependency graph for cache.h:



### Functions

- `int l4_cache_clean_data` (unsigned long start, unsigned long end) `L4_NOTHROW`  
Cache clean a range in D-cache; writes back to PoC.
- `int l4_cache_flush_data` (unsigned long start, unsigned long end) `L4_NOTHROW`  
Cache flush a range; writes back to PoC.
- `int l4_cache_inv_data` (unsigned long start, unsigned long end) `L4_NOTHROW`  
Cache invalidate a range; might write back to PoC.
- `int l4_cache_coherent` (unsigned long start, unsigned long end) `L4_NOTHROW`  
Make memory coherent between I-cache and D-cache; writes back to PoU.
- `int l4_cache_dma_coherent` (unsigned long start, unsigned long end) `L4_NOTHROW`  
Make memory coherent for use with external memory; writes back to PoC.
- `int l4_cache_dma_coherent_full` (void) `L4_NOTHROW`  
Make memory coherent for use with external memory; writes back to PoC.

## 17.456.1 Detailed Description

Cache functions.

Date

2007-11

Author

Adam Lackorzynski [adam@os.inf.tu-dresden.de](mailto:adam@os.inf.tu-dresden.de)

Definition in file [cache.h](#).

## 17.457 cache.h

[Go to the documentation of this file.](#)

```

00001
00009 /*
00010 * (c) 2007-2009 Author(s)
00011 * economic rights: Technische Universität Dresden (Germany)
00012 *
00013 * License: see LICENSE.spdx (in this directory or the directories above)
00014 */
00015 #ifndef __L4SYS__INCLUDE__ARCH_ARM__CACHE_H__
00016 #define __L4SYS__INCLUDE__ARCH_ARM__CACHE_H__
00017
00018 #include <l4/sys/compiler.h>
00019 #include <l4/sys/syscall_defs.h>
00020
00021 #include_next <l4/sys/cache.h>
00022
00026 L4_INLINE void
00027 l4_cache_op_arm_call(unsigned long op,
00028 unsigned long start,
00029 unsigned long end);
00030
00031 L4_INLINE void
00032 l4_cache_op_arm_call(unsigned long op,
00033 unsigned long start,
00034 unsigned long end)
00035 {
00036 register unsigned long _op __asm__ ("r0") = op;
00037 register unsigned long _start __asm__ ("r1") = start;
00038 register unsigned long _end __asm__ ("r2") = end;
00039
00040 __asm__ __volatile__
00041 ("@ l4_cache_op_arm_call(start) \n\t"
00042 "mov r5, %[sc] \n\t"
00043 "blx __l4_sys_syscall \n\t"
00044 "@ l4_cache_op_arm_call(end) \n\t"
00045 :
00046 "=r" (_op),
00047 "=r" (_start),
00048 "=r" (_end)
00049 :
00050 [sc] "i" (L4_SYSCALL_MEM_OP),
00051 "0" (_op),
00052 "1" (_start),
00053 "2" (_end)
00054 :
00055 "cc", "memory", "r5", "ip", "lr"
00056);
00057 }
00058
00059 enum L4_mem_cache_ops
00060 {
00061 L4_MEM_CACHE_OP_CLEAN_DATA = 0,
00062 L4_MEM_CACHE_OP_FLUSH_DATA = 1,
00063 L4_MEM_CACHE_OP_INV_DATA = 2,
00064 L4_MEM_CACHE_OP_COHERENT = 3,
00065 L4_MEM_CACHE_OP_DMA_COHERENT = 4,

```

```

00066 L4_MEM_CACHE_OP_DMA_COHERENT_FULL = 5,
00067 };
00068
00069 L4_INLINE int
00070 l4_cache_clean_data(unsigned long start,
00071 unsigned long end) L4_NOTHROW
00072 {
00073 l4_cache_op_arm_call(L4_MEM_CACHE_OP_CLEAN_DATA, start, end);
00074 return 0;
00075 }
00076
00077 L4_INLINE int
00078 l4_cache_flush_data(unsigned long start,
00079 unsigned long end) L4_NOTHROW
00080 {
00081 l4_cache_op_arm_call(L4_MEM_CACHE_OP_FLUSH_DATA, start, end);
00082 return 0;
00083 }
00084
00085 L4_INLINE int
00086 l4_cache_inv_data(unsigned long start,
00087 unsigned long end) L4_NOTHROW
00088 {
00089 l4_cache_op_arm_call(L4_MEM_CACHE_OP_INV_DATA, start, end);
00090 return 0;
00091 }
00092
00093 L4_INLINE int
00094 l4_cache_coherent(unsigned long start,
00095 unsigned long end) L4_NOTHROW
00096 {
00097 l4_cache_op_arm_call(L4_MEM_CACHE_OP_COHERENT, start, end);
00098 return 0;
00099 }
00100
00101 L4_INLINE int
00102 l4_cache_dma_coherent(unsigned long start,
00103 unsigned long end) L4_NOTHROW
00104 {
00105 l4_cache_op_arm_call(L4_MEM_CACHE_OP_DMA_COHERENT, start, end);
00106 return 0;
00107 }
00108
00109 L4_INLINE int
00110 l4_cache_dma_coherent_full(void) L4_NOTHROW
00111 {
00112 l4_cache_op_arm_call(L4_MEM_CACHE_OP_DMA_COHERENT_FULL, 0, 0);
00113 return 0;
00114 }
00115
00116 #endif /* ! __L4SYS__INCLUDE__ARCH_ARM__CACHE_H__ */

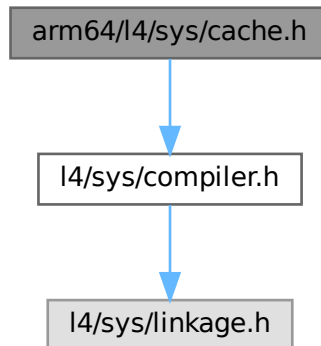
```

## 17.458 arm64/l4/sys/cache.h File Reference

Cache functions.

```
#include <l4/sys/compiler.h>
```

Include dependency graph for cache.h:



## Functions

- int [l4\\_cache\\_clean\\_data](#) (unsigned long start, unsigned long end) [L4\\_NOTHROW](#)  
*Cache clean a range in D-cache; writes back to PoC.*
- int [l4\\_cache\\_flush\\_data](#) (unsigned long start, unsigned long end) [L4\\_NOTHROW](#)  
*Cache flush a range; writes back to PoC.*
- int [l4\\_cache\\_inv\\_data](#) (unsigned long start, unsigned long end) [L4\\_NOTHROW](#)  
*Cache invalidate a range; might write back to PoC.*
- int [l4\\_cache\\_coherent](#) (unsigned long start, unsigned long end) [L4\\_NOTHROW](#)  
*Make memory coherent between I-cache and D-cache; writes back to PoU.*
- int [l4\\_cache\\_dma\\_coherent](#) (unsigned long start, unsigned long end) [L4\\_NOTHROW](#)  
*Make memory coherent for use with external memory; writes back to PoC.*

### 17.458.1 Detailed Description

Cache functions.

Date

2007-11

Author

Adam Lackorzynski [adam@os.inf.tu-dresden.de](mailto:adam@os.inf.tu-dresden.de)

Definition in file [cache.h](#).

## 17.459 cache.h

[Go to the documentation of this file.](#)

```

00001
00009 /*
00010 * (c) 2007-2009 Author(s)
00011 * economic rights: Technische Universität Dresden (Germany)
00012 *
00013 * License: see LICENSE.spdx (in this directory or the directories above)
00014 */
00015 #ifndef __L4SYS__INCLUDE__ARCH_ARM__CACHE_H__
00016 #define __L4SYS__INCLUDE__ARCH_ARM__CACHE_H__
00017
00018 #include <l4/sys/compiler.h>
00019
00020 #include_next <l4/sys/cache.h>
00021
00022 L4_INLINE unsigned long __attribute__((pure, always_inline))
00023 l4_cache_arm_ctr(void);
00024
00025 L4_INLINE unsigned long __attribute__((pure, always_inline))
00026 l4_cache_arm_ctr(void)
00027 {
00028 unsigned long v;
00029 asm ("mrs %0, CTR_EL0" : "=r"(v));
00030 return v;
00031 }
00032
00033 L4_INLINE unsigned __attribute__((pure, always_inline))
00034 l4_cache_dmin_line(void);
00035
00036 L4_INLINE unsigned __attribute__((pure, always_inline))
00037 l4_cache_dmin_line(void)
00038 {
00039 return 4U < ((l4_cache_arm_ctr() > 16) & 0xf);
00040 }
00041
00042 #define L4_ARM_CACHE_LOOP(op)
00043 unsigned long step;
00044
00045 if (start > end)
00046 __builtin_unreachable();
00047
00048 step = l4_cache_dmin_line();
00049 start &= ~(step - 1);
00050 end = (end + step - 1) & ~(step - 1);
00051 for (; start != end; start += step)
00052 asm volatile (op " ", %0" : : "r"(start) : "memory");
00053 asm volatile ("dsb ish");
00054
00055
00056 L4_INLINE int
00057 l4_cache_clean_data(unsigned long start,
00058 unsigned long end) L4_NOTHROW
00059 {
00060 L4_ARM_CACHE_LOOP("dc cvac");
00061 return 0;
00062 }
00063
00064 L4_INLINE int
00065 l4_cache_flush_data(unsigned long start,
00066 unsigned long end) L4_NOTHROW
00067 {
00068 L4_ARM_CACHE_LOOP("dc civac");
00069 return 0;
00070 }
00071
00072 L4_INLINE int
00073 l4_cache_inv_data(unsigned long start,
00074 unsigned long end) L4_NOTHROW
00075 {
00076 // DC IVAC is always privileged, use DC CIVAC instead
00077 L4_ARM_CACHE_LOOP("dc civac");
00078 return 0;
00079 }
00080
00081 L4_INLINE int
00082 l4_cache_coherent(unsigned long start,
00083 unsigned long end) L4_NOTHROW
00084 {
00085 L4_ARM_CACHE_LOOP("dc cvau, %0; ic ivau");
00086 asm volatile ("isb");
00087 return 0;
00088 }
00089

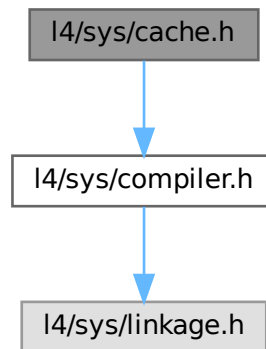
```

```
00090 L4_INLINE int
00091 l4_cache_dma_coherent(unsigned long start,
00092 unsigned long end) L4_NOTHROW
00093 {
00094 L4_ARM_CACHE_LOOP("dc civac");
00095 return 0;
00096 }
00097
00098 #undef L4_ARM_CACHE_LOOP
00099
00100 #endif /* ! __L4SYS__INCLUDE__ARCH_ARM__CACHE_H__ */
```

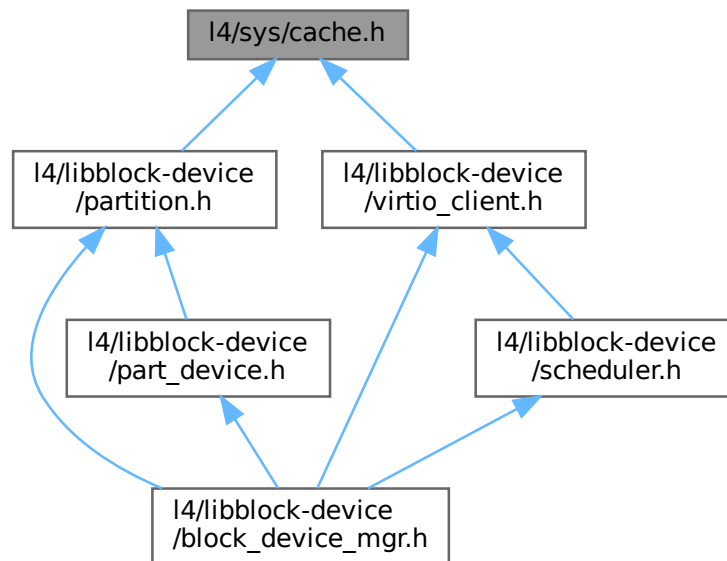
## 17.460 l4/sys/cache.h File Reference

Cache-consistency functions.

```
#include <l4/sys/compiler.h>
Include dependency graph for cache.h:
```



This graph shows which files directly or indirectly include this file:



## Functions

- [L4\\_BEGIN\\_DECLS](#) [int l4\\_cache\\_clean\\_data](#) (unsigned long start, unsigned long end) [L4\\_NOTHROW](#)  
*Cache clean a range in D-cache; writes back to PoC.*
- [int l4\\_cache\\_flush\\_data](#) (unsigned long start, unsigned long end) [L4\\_NOTHROW](#)  
*Cache flush a range; writes back to PoC.*
- [int l4\\_cache\\_inv\\_data](#) (unsigned long start, unsigned long end) [L4\\_NOTHROW](#)  
*Cache invalidate a range; might write back to PoC.*
- [int l4\\_cache\\_coherent](#) (unsigned long start, unsigned long end) [L4\\_NOTHROW](#)  
*Make memory coherent between I-cache and D-cache; writes back to PoU.*
- [int l4\\_cache\\_dma\\_coherent](#) (unsigned long start, unsigned long end) [L4\\_NOTHROW](#)  
*Make memory coherent for use with external memory; writes back to PoC.*
- [int l4\\_cache\\_dma\\_coherent\\_full](#) (void) [L4\\_NOTHROW](#)  
*Make memory coherent for use with external memory; writes back to PoC.*

## 17.460.1 Detailed Description

Cache-consistency functions.

Date

2007-11

Author

Adam Lackorzynski [adam@os.inf.tu-dresden.de](mailto:adam@os.inf.tu-dresden.de)

Definition in file [cache.h](#).



## 17.461 cache.h

[Go to the documentation of this file.](#)

```

00001
00010 /*
00011 * (c) 2007-2009 Author(s)
00012 * economic rights: Technische Universität Dresden (Germany)
00013 *
00014 * License: see LICENSE.spdx (in this directory or the directories above)
00015 */
00016
00017 #ifndef __L4SYS__INCLUDE__CACHE_H__
00018 #define __L4SYS__INCLUDE__CACHE_H__
00019
00020 #include <14/sys/compiler.h>
00021
00036
00037 L4_BEGIN_DECLS
00038
00053 L4_INLINE int
00054 l4_cache_clean_data(unsigned long start,
00055 unsigned long end) L4_NOTHROW;
00056
00071 L4_INLINE int
00072 l4_cache_flush_data(unsigned long start,
00073 unsigned long end) L4_NOTHROW;
00074
00093 L4_INLINE int
00094 l4_cache_inv_data(unsigned long start,
00095 unsigned long end) L4_NOTHROW;
00096
00108 L4_INLINE int
00109 l4_cache_coherent(unsigned long start,
00110 unsigned long end) L4_NOTHROW;
00111
00123 L4_INLINE int
00124 l4_cache_dma_coherent(unsigned long start,
00125 unsigned long end) L4_NOTHROW;
00126
00127 #if !defined(ARCH_arm64)
00132 L4_INLINE int
00133 l4_cache_dma_coherent_full(void) L4_NOTHROW;
00134 #endif
00135
00136 L4_END_DECLS
00137
00138 #endif /* ! __L4SYS__INCLUDE__CACHE_H__ */

```

## 17.462 x86/I4/sys/cache.h File Reference

Cache functions.

### Functions

- int [l4\\_cache\\_clean\\_data](#) (unsigned long start, unsigned long end) [L4\\_NOTHROW](#)  
*Cache clean a range in D-cache; writes back to PoC.*
- int [l4\\_cache\\_flush\\_data](#) (unsigned long start, unsigned long end) [L4\\_NOTHROW](#)  
*Cache flush a range; writes back to PoC.*
- int [l4\\_cache\\_inv\\_data](#) (unsigned long start, unsigned long end) [L4\\_NOTHROW](#)  
*Cache invalidate a range; might write back to PoC.*
- int [l4\\_cache\\_coherent](#) (unsigned long start, unsigned long end) [L4\\_NOTHROW](#)  
*Make memory coherent between I-cache and D-cache; writes back to PoU.*
- int [l4\\_cache\\_dma\\_coherent](#) (unsigned long start, unsigned long end) [L4\\_NOTHROW](#)  
*Make memory coherent for use with external memory; writes back to PoC.*
- int [l4\\_cache\\_dma\\_coherent\\_full](#) (void) [L4\\_NOTHROW](#)  
*Make memory coherent for use with external memory; writes back to PoC.*

## 17.462.1 Detailed Description

Cache functions.

Definition in file [cache.h](#).

## 17.463 cache.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #ifndef __L4SYS__INCLUDE__ARCH_X86__CACHE_H__
00012 #define __L4SYS__INCLUDE__ARCH_X86__CACHE_H__
00013
00014 #include_next <l4/sys/cache.h>
00015
00016 L4_INLINE int
00017 l4_cache_clean_data(unsigned long start,
00018 unsigned long end) L4_NOTHROW
00019 {
00020 (void)start; (void)end;
00021 return 0;
00022 }
00023
00024 L4_INLINE int
00025 l4_cache_flush_data(unsigned long start,
00026 unsigned long end) L4_NOTHROW
00027 {
00028 (void)start; (void)end;
00029 return 0;
00030 }
00031
00032 L4_INLINE int
00033 l4_cache_inv_data(unsigned long start,
00034 unsigned long end) L4_NOTHROW
00035 {
00036 (void)start; (void)end;
00037 return 0;
00038 }
00039
00040 L4_INLINE int
00041 l4_cache_coherent(unsigned long start,
00042 unsigned long end) L4_NOTHROW
00043 {
00044 (void)start; (void)end;
00045 return 0;
00046 }
00047
00048 L4_INLINE int
00049 l4_cache_dma_coherent(unsigned long start,
00050 unsigned long end) L4_NOTHROW
00051 {
00052 (void)start; (void)end;
00053 return 0;
00054 }
00055
00056 L4_INLINE int
00057 l4_cache_dma_coherent_full(void) L4_NOTHROW
00058 {
00059 return 0;
00060 }
00061
00062 #endif /* ! __L4SYS__INCLUDE__ARCH_X86__CACHE_H__ */

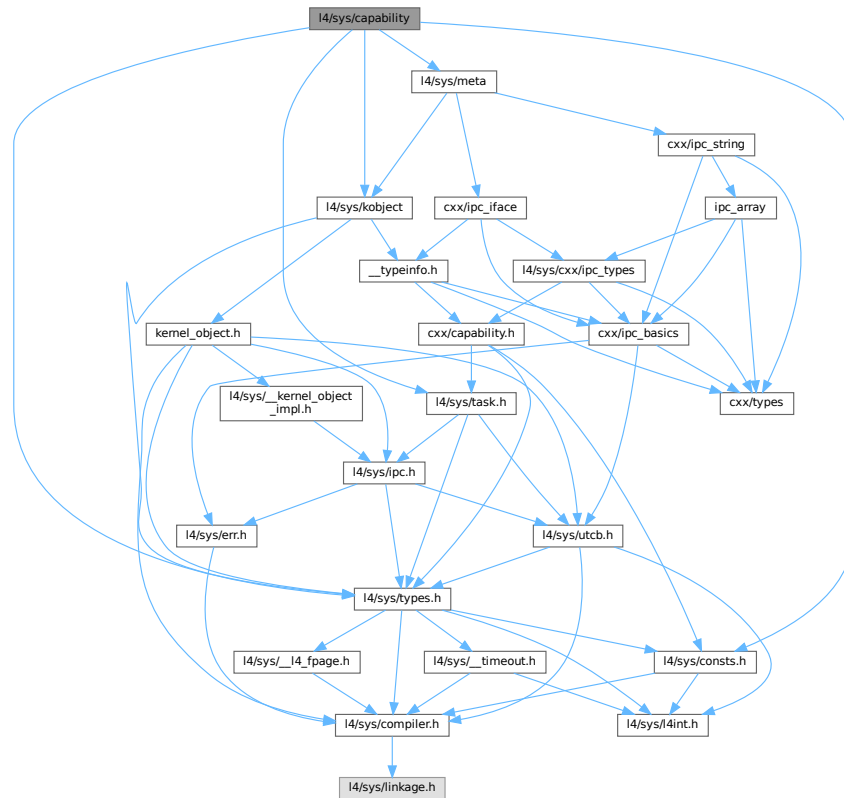
```

## 17.464 l4/sys/capability File Reference

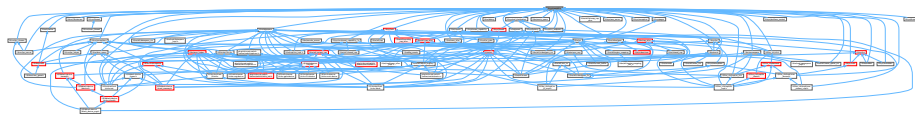
[L4::Cap](#) related definitions.

```
#include <l4/sys/consts.h>
#include <l4/sys/types.h>
#include <l4/sys/kobject>
#include <l4/sys/task.h>
#include <l4/sys/meta>
```

Include dependency graph for capability:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [L4](#)  
*L4 low-level kernel interface.*

## Macros

- #define [L4\\_DISABLE\\_COPY](#)(\_class)  
*Disable copy of a class.*

## Functions

- `template<typename T, typename F>`  
`Cap< T > L4::cap_dynamic_cast (Cap< F > const &c) noexcept`  
*dynamic\_cast for capabilities.*

### 17.464.1 Detailed Description

[L4::Cap](#) related definitions.

#### Author

Alexander Warg [alexander.warg@os.inf.tu-dresden.de](mailto:alexander.warg@os.inf.tu-dresden.de)

Definition in file [capability](#).

### 17.464.2 Macro Definition Documentation

#### 17.464.2.1 L4\_DISABLE\_COPY

```
#define L4_DISABLE_COPY(
 _class)
```

#### Value:

```
public:
 _class(_class const &) = delete; \
 _class operator = (_class const &) = delete; \
private:
```

Disable copy of a class.

#### Parameters

|                     |                                                             |
|---------------------|-------------------------------------------------------------|
| <code>_class</code> | Name of the class that shall not have value copy semantics. |
|---------------------|-------------------------------------------------------------|

The typical use of this is:

```
class Non_value
{
 L4_DISABLE_COPY(Non_value)

 ...
}
```

Definition at line 52 of file [capability](#).

## 17.465 capability

[Go to the documentation of this file.](#)

```

00001 // vim:set ft=cpp: -*- Mode: C++ -*-
00009 /*
00010 * (c) 2008-2009,2015 Author(s)
00011 * economic rights: Technische Universität Dresden (Germany)
00012 *
00013 * License: see LICENSE.spdx (in this directory or the directories above)
00014 */
00015 #pragma once
00016
00017 #include <l4/sys/consts.h>
00018 #include <l4/sys/types.h>
00019 #include <l4/sys/kobject>
00020 #include <l4/sys/task.h>
00021
00022 namespace L4
00023 {
00024
00025 /* Forward declarations for our kernel object classes. */
00026 class Task;
00027 class Thread;
00028 class Thread_group;
00029 class Factory;
00030 class Irq;
00031 class Log;
00032 class Vm;
00033 class Vcpu_context;
00034 class Kobject;
00035
00051 #if __cplusplus >= 201103L
00052 # define L4_DISABLE_COPY(_class) \
00053 public: \
00054 _class(_class const &) = delete; \
00055 _class operator = (_class const &) = delete; \
00056 private:
00057 #else
00058 # define L4_DISABLE_COPY(_class) \
00059 private: \
00060 _class(_class const &); \
00061 _class operator = (_class const &);
00062 #endif
00063
00064
00065 #define L4_KOBJECT_DISABLE_COPY(_class) \
00066 protected: \
00067 _class(); \
00068 L4_DISABLE_COPY(_class)
00069
00070
00071 #define L4_KOBJECT(_class) L4_KOBJECT_DISABLE_COPY(_class)
00072
00073 inline l4_msgtag_t
00074 Cap_base::validate(Cap<Task> task, l4_utcb_t *u) const noexcept
00075 {
00076 return is_valid() ? l4_task_cap_valid_u(task.cap(), _c, u)
00077 : l4_msgtag(0, 0, 0, 0);
00078 }
00079
00080 inline l4_msgtag_t
00081 Cap_base::validate(l4_utcb_t *u) const noexcept
00082 {
00083 return is_valid() ? l4_task_cap_valid_u(L4_BASE_TASK_CAP, _c, u)
00084 : l4_msgtag(0, 0, 0, 0);
00085 }
00086
00087 }; // namespace L4
00088
00089 #include <l4/sys/meta>
00090
00091 namespace L4 {
00092
00114 template< typename T, typename F >
00115 inline
00116 Cap<T> cap_dynamic_cast(Cap<F> const &c) noexcept
00117 {
00118 if (!c.is_valid())
00119 return Cap<T>::Invalid;
00120
00121 Cap<Meta> mc = cap_reinterpret_cast<Meta>(c);
00122 Type_info const *m = kobject_typeid<T>();
00123 if (m->proto() && l4_error(mc->supports(m->proto())) > 0)
00124 return Cap<T>(c.cap());
00125

```

```

00126 // FIXME: use generic checker
00127 #if 0
00128 if (l4_error(mc->supports(T::kobject_proto())) > 0)
00129 return Cap<T>(c.cap());
00130 #endif
00131
00132 return Cap<T>::Invalid;
00133 }
00134
00135 }

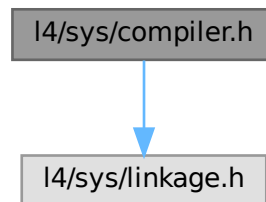
```

## 17.466 l4/sys/compiler.h File Reference

L4 compiler related defines.

```
#include <l4/sys/linkage.h>
```

Include dependency graph for compiler.h:



This graph shows which files directly or indirectly include this file:



### Macros

- **#define L4\_INLINE**  
*L4 Inline function attribute.*
- **#define L4\_ALWAYS\_INLINE**  
*Always inline a function.*
- **#define L4\_NOTHROW**  
*Mark a function declaration and definition as never throwing an exception.*
- **#define L4\_BEGIN\_DECLS**  
*Start section with C types and functions.*
- **#define L4\_END\_DECLS**  
*End section with C types and functions.*
- **#define L4\_CONSTEXPR**  
*Constexpr function attribute.*
- **#define L4\_NORETURN**

- Noreturn function attribute.*
- **#define L4\_NOINSTRUMENT**  
*No instrumentation function attribute.*
- **#define L4\_HIDDEN**  
*Attribute to mark functions, variables, and data types as being explicitly hidden from users of a library.*
- **#define L4\_EXPORT**  
*Attribute to mark functions, variables, and data types as being exported from a library.*
- **#define L4\_LIKELY(x)**  
*Expression is likely to execute.*
- **#define L4\_UNLIKELY(x)**  
*Expression is unlikely to execute.*
- **#define L4\_STICKY(x)**  
*Mark symbol sticky (even not there).*
- **#define L4\_DEPRECATED(s)**  
*Mark symbol deprecated.*
- **#define L4\_stringify\_helper(x)**  
*stringify helper.*
- **#define L4\_stringify(x)**  
*stringify.*

## Functions

- unsigned long **[l4\\_align\\_stack\\_for\\_direct\\_fncall](#)** (unsigned long stack)  
*Specify the desired alignment of the stack pointer.*
- void **[l4\\_barrier](#)** (void)  
*Memory barrier.*
- void **[l4\\_mb](#)** (void)  
*Memory barrier.*
- void **[l4\\_wmb](#)** (void)  
*Write memory barrier.*
- **[L4\\_NORETURN](#)** void **[l4\\_infinite\\_loop](#)** (void)  
*Infinite loop.*

## 17.466.1 Detailed Description

[L4](#) compiler related defines.

Definition in file [compiler.h](#).

## 17.467 compiler.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002 */
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00005 * Frank Mehnert <fm3@os.inf.tu-dresden.de>,
00006 * Jork Löser <jork@os.inf.tu-dresden.de>,
00007 * Ronald Aigner <ra3@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 /*****
00013 */
00014 #ifndef __L4_COMPILER_H__
00015 #define __L4_COMPILER_H__
00016
00017 #if !defined(__ASSEMBLY__) && !defined(__ASSEMBLER__)
00018
00019 #ifndef L4_INLINE
00020 #ifndef __cplusplus
00021 # ifdef __OPTIMIZE__
00022 # define L4_INLINE_STATIC static __inline__
00023 # define L4_INLINE_EXTERN extern __inline__
00024 # ifdef __GNUC_STDC_INLINE__
00025 # define L4_INLINE L4_INLINE_STATIC
00026 # else
00027 # define L4_INLINE L4_INLINE_EXTERN
00028 # endif
00029 # else /* ! __OPTIMIZE__ */
00030 # define L4_INLINE static
00031 # endif /* ! __OPTIMIZE__ */
00032 #else /* __cplusplus */
00033 #define L4_INLINE inline
00034 #endif /* __cplusplus */
00035 #elif defined DOXYGEN
00036 #define L4_INLINE inline
00037 #endif /* L4_INLINE */
00038 #define L4_ALWAYS_INLINE L4_INLINE __attribute__((__always_inline__))
00039
00040 #define L4_DECLARE_CONSTRUCTOR(func, prio) \
00041 static inline __attribute__((constructor(prio))) void func ## _ctor_func(void) { func(); }
00042
00043 #ifndef __cplusplus
00044 #define L4_NOTHROW_A __attribute__((nothrow))
00045 #define L4_NOTHROW
00046 #ifndef __BEGIN_DECLS
00047 #define __BEGIN_DECLS
00048 #endif
00049 #ifndef __END_DECLS
00050 #define __END_DECLS
00051 #endif
00052 #define L4_BEGIN_DECLS
00053 #define L4_END_DECLS
00054 #define L4_DEFAULT_PARAM(x)
00055 #else /* __cplusplus */
00056 #if __cplusplus >= 201103L
00057 #define L4_NOTHROW noexcept
00058 #else /* C++ < 11 */
00059 #define L4_NOTHROW throw()
00060 #endif
00061 #define L4_BEGIN_DECLS extern "C" {
00062 #define L4_END_DECLS }
00063 #if !defined __BEGIN_DECLS || defined DOXYGEN
00064 #define __BEGIN_DECLS extern "C" {
00065 #endif
00066 #if !defined __END_DECLS || defined DOXYGEN
00067 #define __END_DECLS }
00068 #endif
00069 #define L4_DEFAULT_PARAM(x) = x
00070 #endif /* __cplusplus */
00071
00072 /* Deprecation hints during compile -- remove later (2025+) */
00073 #ifndef EXTERN_C
00074 #define EXTERN_C DO_NOT_USE_EXTERN_C_ANY_MORE
00075 #endif
00076 #ifndef EXTERN_C_BEGIN

```



```

00181 #define EXTERN_C_BEGIN DO_NOT_USE_EXTERN_C_BEGIN_ANY_MORE__USE_L4_BEGIN_DECLS
00182 #endif
00183 #ifndef EXTERN_C_END
00184 #define EXTERN_C_END DO_NOT_USE_EXTERN_C_END_ANY_MORE__USE_L4_END_DECLS
00185 #endif
00186
00191 #if defined __cplusplus && __cplusplus >= 201402L
00192 # define L4_CONSTEXPR constexpr
00193 #else
00194 # define L4_CONSTEXPR
00195 #endif
00196
00201 #define L4_NORETURN __attribute__((noreturn))
00202
00203 #define L4_PURE __attribute__((pure))
00204
00209 #define L4_NOINSTRUMENT __attribute__((no_instrument_function))
00210 #ifndef L4_HIDDEN
00211 # define L4_HIDDEN __attribute__((visibility("hidden")))
00212 #endif
00213 #if !defined L4_EXPORT || defined DOXYGEN
00214 # define L4_EXPORT __attribute__((visibility("default")))
00215 #endif
00216 #ifndef L4_EXPORT_TYPE
00217 # ifdef __cplusplus
00218 # define L4_EXPORT_TYPE __attribute__((visibility("default")))
00219 # else
00220 # define L4_EXPORT_TYPE
00221 # endif
00222 #endif
00223 #define L4_STRONG_ALIAS(name, aliasname) L4__STRONG_ALIAS(name, aliasname)
00224 #ifdef __clang__
00225 #define L4__STRONG_ALIAS(name, aliasname) \
00226 extern __typeof (name) aliasname __attribute__((alias (#name)));
00227 #else
00228 #define L4__STRONG_ALIAS(name, aliasname) \
00229 extern __typeof (name) aliasname __attribute__((alias (#name), copy(name)));
00230 #endif
00231
00239 #if defined(__i386__) || defined(__amd64__) || \
00240 defined(__arm__) || defined(__aarch64__) || \
00241 defined(__mips__) || defined(__riscv__) || \
00242 defined(__powerpc__) || defined(__sparc__)
00243 # define L4_STACK_ALIGN __BIGGEST_ALIGNMENT__
00244 #else
00245 # error Define L4_STACK_ALIGN for this target!
00246 #endif
00247
00265 #if defined(__i386__) || defined(__amd64__)
00266 L4_INLINE unsigned long l4_align_stack_for_direct_fncall(unsigned long stack)
00267 {
00268 if ((stack & (L4_STACK_ALIGN - 1)) == (L4_STACK_ALIGN - sizeof(unsigned long)))
00269 return stack;
00270 return (stack & ~(L4_STACK_ALIGN)) - sizeof(unsigned long);
00271 }
00272 #else
00273 L4_INLINE unsigned long l4_align_stack_for_direct_fncall(unsigned long stack)
00274 {
00275 return stack & ~(L4_STACK_ALIGN);
00276 }
00277 #endif
00278
00279 #endif /* !__ASSEMBLY__ */
00280
00281 #include <l4/sys/linkage.h>
00282
00283 #define L4_LIKELY(x) __builtin_expect((x),1)
00284 #define L4_UNLIKELY(x) __builtin_expect((x),0)
00285
00286 /* Make sure that the function is not removed by optimization. Without the
00287 * "used" attribute, unreferenced static functions are removed. */
00288 #define L4_STICKY(x) __attribute__((used)) x
00289 #define L4_DEPRECATED(s) __attribute__((deprecated(s)))
00290
00291 #ifndef static_assert
00292 # if !defined(__cplusplus)
00293 # define static_assert(x, y) _Static_assert(x, y)
00294 # elif __cplusplus < 201103L
00295 # define static_assert(x, y) \
00296 extern int l4_static_assert[-(!x)] __attribute__((unused))
00297 # endif
00298 #endif
00299
00300 #define L4_stringify_helper(x) #x
00301 #define L4_stringify(x) L4_stringify_helper(x)
00302
00303 #ifdef __has_builtin

```

```

00304 #define L4_HAS_BUILTIN(def) __has_builtin(def)
00305 #else
00306 #define L4_HAS_BUILTIN(def) 0
00307 #endif
00308
00309 #ifndef __ASSEMBLER__
00313 L4_INLINE void l4_barrier(void);
00314
00318 L4_INLINE void l4_mb(void);
00319
00323 L4_INLINE void l4_wmb(void);
00324
00328 L4_INLINE L4_NORETURN void l4_infinite_loop(void);
00329
00330
00331 /* Implementations */
00332 L4_INLINE void l4_barrier(void)
00333 {
00334 __asm__ __volatile__ ("": : : "memory");
00335 }
00336
00337 L4_INLINE void l4_mb(void)
00338 {
00339 __asm__ __volatile__ ("": : : "memory");
00340 }
00341
00342 L4_INLINE void l4_wmb(void)
00343 {
00344 __asm__ __volatile__ ("": : : "memory");
00345 }
00346
00347 L4_INLINE L4_NORETURN void l4_infinite_loop(void)
00348 {
00349 while (1)
00350 l4_barrier();
00351 }
00352 #endif
00353
00355
00356 #endif /* !__L4_COMPILER_H__ */

```

## 17.468 amd64/l4/sys/consts.h File Reference

Common [L4](#) constants, AMD64 version.

### Macros

- `#define L4_PAGESHIFT 12`  
*Size of a page, log2-based.*
- `#define L4_SUPERPAGESHIFT 21`  
*Size of a large page, log2-based.*

### 17.468.1 Detailed Description

Common [L4](#) constants, AMD64 version.

Definition in file [consts.h](#).

## 17.469 consts.h

[Go to the documentation of this file.](#)

```

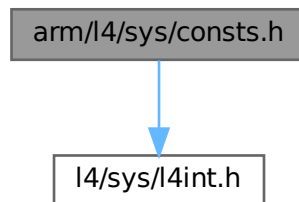
00001 /*****
00007 */
00008 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00009 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00010 * Björn Döbel <doebel@os.inf.tu-dresden.de>,
00011 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00012 * economic rights: Technische Universität Dresden (Germany)
00013 *
00014 * License: see LICENSE.spdx (in this directory or the directories above)
00015 */
00016 /*****
00017 #ifndef __L4SYS__INCLUDE__ARCH_AMD64__CONSTS_H__
00018 #define __L4SYS__INCLUDE__ARCH_AMD64__CONSTS_H__
00019
00024 #define L4_PAGESHIFT 12
00025
00030 #define L4_SUPERPAGESHIFT 21
00031
00032 #include_next <l4/sys/consts.h>
00033
00034 #endif /* ! __L4SYS__INCLUDE__ARCH_AMD64__CONSTS_H__ */

```

## 17.470 arm/l4/sys/consts.h File Reference

Common [L4](#) constants, arm version.

#include <l4/sys/l4int.h>  
 Include dependency graph for consts.h:



### Macros

- #define [L4\\_PAGESHIFT](#) 12  
*Size of a page, log2-based.*
- #define [L4\\_SUPERPAGESHIFT](#) 21  
*Size of a large page, log2-based.*

### 17.470.1 Detailed Description

Common [L4](#) constants, arm version.

Definition in file [consts.h](#).

## 17.471 consts.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00009 * Björn Döbel <doebel@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #ifndef _L4_SYS_CONSTS_H
00015 #define _L4_SYS_CONSTS_H
00016
00017 /* L4 includes */
00018 #include <l4/sys/l4int.h>
00026 #define L4_PAGESHIFT 12
00027
00031 #define L4_SUPERPAGESHIFT 21
00032
00034
00035 #include_next <l4/sys/consts.h>
00036
00037 #endif /* !_L4_SYS_CONSTS_H */

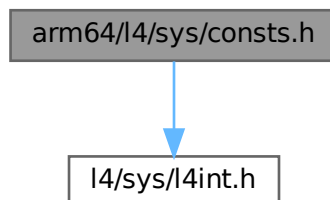
```

## 17.472 arm64/l4/sys/consts.h File Reference

Common [L4](#) constants, arm version.

`#include <l4/sys/l4int.h>`

Include dependency graph for consts.h:



### Macros

- `#define L4_PAGESHIFT 12`  
*Size of a page, log2-based.*
- `#define L4_SUPERPAGESHIFT 21`  
*Size of a large page, log2-based.*

### 17.472.1 Detailed Description

Common [L4](#) constants, arm version.

Definition in file [consts.h](#).

## 17.473 consts.h

[Go to the documentation of this file.](#)

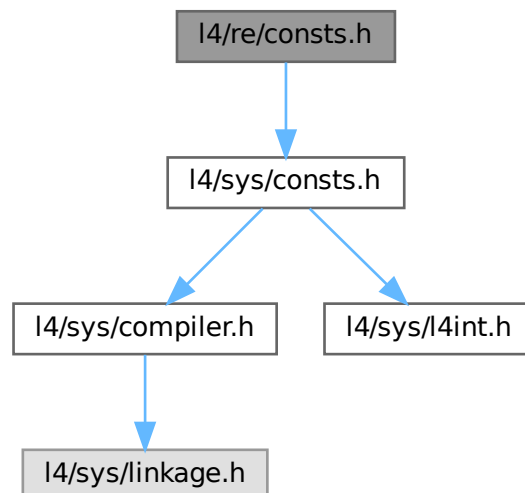
```
00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00009 * Björn Döbel <doebel@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #ifndef _L4_SYS_CONSTS_H
00015 #define _L4_SYS_CONSTS_H
00016
00017 /* L4 includes */
00018 #include <l4/sys/l4int.h>
00026 #define L4_PAGESHIFT 12
00027
00031 #define L4_SUPERPAGESHIFT 21
00032
00034
00035 #include_next <l4/sys/consts.h>
00036
00037 #endif /* !_L4_SYS_CONSTS_H */
```

## 17.474 l4/re/consts.h File Reference

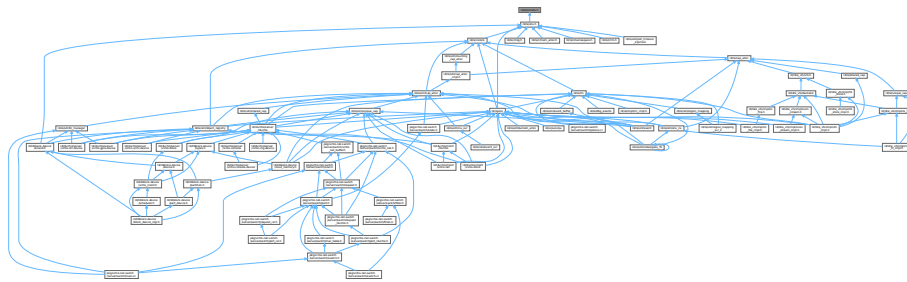
Constants.

```
#include <l4/sys/consts.h>
```

Include dependency graph for consts.h:



This graph shows which files directly or indirectly include this file:



## Enumerations

- `enum`  
*Defaults for local thread priorities.*

### 17.474.1 Detailed Description

Constants.

Definition in file [consts.h](#).

### 17.474.2 Enumeration Type Documentation

#### 17.474.2.1 anonymous enum

anonymous enum

Defaults for local thread priorities.

Priorities are to be seen as local. These are used by the loader and libpthread. They are to be understood as 'local', which means the actual priority of the thread (as seen by the kernel) is the base priority as defined by the scheduler plus the local priority.

Definition at line 28 of file [consts.h](#).

## 17.475 consts.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #pragma once
00012
00013 #include <l4/sys/consts.h>
00014
00015 enum
00016 {
00017 L4RE_THIS_TASK_CAP = 1UL << L4_CAP_SHIFT,
00018 };
00019
00028 enum
00029 {
00030 L4RE_MAIN_THREAD_PRIO = 2, /* Priority of the main thread */
00031 };
00032

```

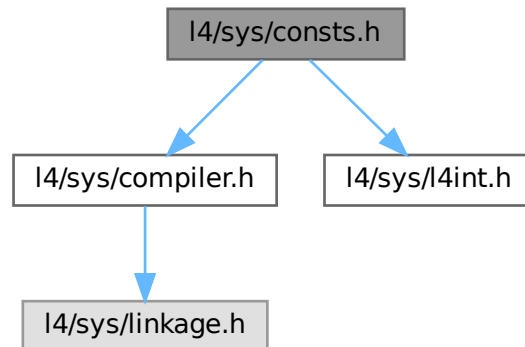
## 17.476 l4/sys/consts.h File Reference

Common constants.

```
#include <l4/sys/compiler.h>
```

```
#include <l4/sys/l4int.h>
```

Include dependency graph for consts.h:



This graph shows which files directly or indirectly include this file:



### Macros

- **#define L4\_PAGESIZE**  
*Minimal page size (in bytes).*
- **#define L4\_PAGEMASK**  
*Mask for the page number.*
- **#define L4\_LOG2\_PAGESIZE**  
*Number of bits used for page offset.*
- **#define L4\_SUPERPAGE\_SIZE**  
*Size of a large page.*
- **#define L4\_SUPERPAGEMASK**  
*Mask for the number of a large page.*
- **#define L4\_LOG2\_SUPERPAGE\_SIZE**  
*Number of bits used as offset for a large page.*
- **#define L4\_INVALID\_PTR** `((void *)L4_INVALID_ADDR)`  
*Invalid address as pointer type.*

## Enumerations

- enum [l4\\_syscall\\_flags\\_t](#) {  
[L4\\_SYSF\\_NONE](#) , [L4\\_SYSF\\_SEND](#) , [L4\\_SYSF\\_RECV](#) , [L4\\_SYSF\\_OPEN\\_WAIT](#) ,  
[L4\\_SYSF\\_REPLY](#) , [L4\\_SYSF\\_CALL](#) , [L4\\_SYSF\\_WAIT](#) , [L4\\_SYSF\\_SEND\\_AND\\_WAIT](#) ,  
[L4\\_SYSF\\_REPLY\\_AND\\_WAIT](#) }  
*Capability selector flags.*
- enum [l4\\_cap\\_consts\\_t](#) {  
[L4\\_CAP\\_SHIFT](#) , [L4\\_CAP\\_SIZE](#) = 1UL << [L4\\_CAP\\_SHIFT](#) , [L4\\_CAP\\_OFFSET](#) , [L4\\_CAP\\_MASK](#) ,  
[L4\\_INVALID\\_CAP](#) , [L4\\_INVALID\\_CAP\\_BIT](#) = 1UL << ([L4\\_CAP\\_SHIFT](#) - 1) }  
*Constants related to capability selectors.*
- enum [l4\\_unmap\\_flags\\_t](#) { [L4\\_FP\\_ALL\\_SPACES](#) , [L4\\_FP\\_DELETE\\_OBJ](#) , [L4\\_FP\\_OTHER\\_SPACES](#) }  
*Flags for the unmap operation.*
- enum [l4\\_msg\\_item\\_consts\\_t](#) {  
[L4\\_ITEM\\_MAP](#) = 8 , [L4\\_ITEM\\_CONT](#) = 1 , [L4\\_MAP\\_ITEM\\_GRANT](#) = 2 , [L4\\_MAP\\_ITEM\\_MAP](#) = 0 ,  
[L4\\_RCV\\_ITEM\\_FORWARD\\_MAPPINGS](#) = 1 , [L4\\_RCV\\_ITEM\\_SINGLE\\_CAP](#) = [L4\\_ITEM\\_MAP](#) | 2 ,  
[L4\\_RCV\\_ITEM\\_LOCAL\\_ID](#) = 4 }  
*Constants for message items.*
- enum [l4\\_buffer\\_desc\\_consts\\_t](#) { [L4\\_BDR\\_MEM\\_SHIFT](#) = 0 , [L4\\_BDR\\_IO\\_SHIFT](#) = 5 , [L4\\_BDR\\_OBJ\\_SHIFT](#)  
= 10 , [L4\\_BDR\\_OFFSET\\_MASK](#) = (1UL << 20) - 1 }  
*Constants for buffer descriptors.*
- enum [l4\\_default\\_caps\\_t](#) {  
[L4\\_BASE\\_TASK\\_CAP](#) , [L4\\_BASE\\_FACTORY\\_CAP](#) , [L4\\_BASE\\_THREAD\\_CAP](#) , [L4\\_BASE\\_PAGER\\_CAP](#) ,  
[L4\\_BASE\\_LOG\\_CAP](#) , [L4\\_BASE\\_ICU\\_CAP](#) , [L4\\_BASE\\_SCHEDULER\\_CAP](#) , [L4\\_BASE\\_IOMMU\\_CAP](#) ,  
[L4\\_BASE\\_DEBUGGER\\_CAP](#) , [L4\\_BASE\\_ARM\\_SMCCC\\_CAP](#) , [L4\\_BASE\\_CAPS\\_LAST\\_P1](#) , [L4\\_BASE\\_CAPS\\_LAST](#)  
= [L4\\_BASE\\_CAPS\\_LAST\\_P1](#) - 1 }  
*Default capabilities setup for the initial tasks.*
- enum [l4\\_addr\\_consts\\_t](#) { [L4\\_INVALID\\_ADDR](#) = ~0UL }  
*Address related constants.*

## Functions

- [l4\\_addr\\_t l4\\_trunc\\_page](#) ([l4\\_addr\\_t](#) address) [L4\\_NOTHROW](#)  
*Round an address down to the next lower page boundary.*
- [l4\\_addr\\_t l4\\_trunc\\_size](#) ([l4\\_addr\\_t](#) address, unsigned char bits) [L4\\_NOTHROW](#)  
*Round an address down to the next lower flexpage with size bits.*
- [l4\\_addr\\_t l4\\_round\\_page](#) ([l4\\_addr\\_t](#) address) [L4\\_NOTHROW](#)  
*Round address up to the next page.*
- [l4\\_addr\\_t l4\\_round\\_size](#) ([l4\\_addr\\_t](#) value, unsigned char bits) [L4\\_NOTHROW](#)  
*Round value up to the next alignment with bits size.*
- unsigned [l4\\_bytes\\_to\\_mwords](#) (unsigned size) [L4\\_NOTHROW](#)  
*Determine how many machine words ([l4\\_umword\\_t](#)) are required to store a buffer of 'size' bytes.*

## 17.476.1 Detailed Description

Common constants.

Definition in file [consts.h](#).



## 17.477 consts.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00009 * Björn Döbel <doebel@os.inf.tu-dresden.de>,
00010 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00011 * economic rights: Technische Universität Dresden (Germany)
00012 *
00013 * License: see LICENSE.spdx (in this directory or the directories above)
00014 */
00015 #ifndef __L4_SYS__INCLUDE__CONSTS_H__
00016 #define __L4_SYS__INCLUDE__CONSTS_H__
00017
00018 #include <l4/sys/compiler.h>
00019 #include <l4/sys/l4int.h>
00020
00050 enum l4_syscall_flags_t
00051 {
00056 L4_SYSF_NONE = 0x00,
00057
00069 L4_SYSF_SEND = 0x01,
00070
00080 L4_SYSF_RECV = 0x02,
00081
00091 L4_SYSF_OPEN_WAIT = 0x04,
00092
00100 L4_SYSF_REPLY = 0x08,
00101
00108 L4_SYSF_CALL = L4_SYSF_SEND | L4_SYSF_RECV,
00109
00116 L4_SYSF_WAIT = L4_SYSF_OPEN_WAIT | L4_SYSF_RECV,
00117
00124 L4_SYSF_SEND_AND_WAIT = L4_SYSF_OPEN_WAIT | L4_SYSF_CALL,
00125
00132 L4_SYSF_REPLY_AND_WAIT = L4_SYSF_WAIT | L4_SYSF_SEND | L4_SYSF_REPLY
00133 };
00134
00139 enum l4_cap_consts_t
00140 {
00142 L4_CAP_SHIFT = 12UL,
00144 L4_CAP_SIZE = 1UL << L4_CAP_SHIFT,
00146 L4_CAP_OFFSET = 1UL << L4_CAP_SHIFT,
00151 L4_CAP_MASK = ~0UL << (L4_CAP_SHIFT - 1),
00153 L4_INVALID_CAP = ~0UL << (L4_CAP_SHIFT - 1),
00154
00155 L4_INVALID_CAP_BIT = 1UL << (L4_CAP_SHIFT - 1),
00156 };
00157
00158 enum l4_sched_consts_t
00159 {
00160 L4_SCHED_MIN_PRIO = 1,
00161 L4_SCHED_MAX_PRIO = 255,
00162 };
00163
00169 enum l4_unmap_flags_t
00170 {
00183 L4_FP_ALL_SPACES = 0x80000000UL,
00184
00197 L4_FP_DELETE_OBJ = 0xc0000000UL,
00198
00205 L4_FP_OTHER_SPACES = 0x00UL
00206 };
00207
00212 enum l4_msg_item_consts_t
00213 {
00214 L4_ITEM_MAP = 8,
00215
00220 L4_ITEM_CONT = 1,
00221
00222 // send
00245 L4_MAP_ITEM_GRANT = 2,
00246
00247 L4_MAP_ITEM_MAP = 0,
00248
00249 // receive
00260 L4_RCV_ITEM_FORWARD_MAPPINGS = 1,
00261
00275 L4_RCV_ITEM_SINGLE_CAP = L4_ITEM_MAP | 2,
00276
00296 L4_RCV_ITEM_LOCAL_ID = 4,
00297 };
00298

```

```

00303 enum l4_buffer_desc_consts_t
00304 {
00305 L4_BDR_MEM_SHIFT = 0,
00306 L4_BDR_IO_SHIFT = 5,
00307 L4_BDR_OBJ_SHIFT = 10,
00308 L4_BDR_OFFSET_MASK = (1UL << 20) - 1,
00309 };
00310
00324 enum l4_default_caps_t
00325 {
00327 L4_BASE_TASK_CAP = 1UL << L4_CAP_SHIFT,
00329 L4_BASE_FACTORY_CAP = 2UL << L4_CAP_SHIFT,
00331 L4_BASE_THREAD_CAP = 3UL << L4_CAP_SHIFT,
00333 L4_BASE_PAGER_CAP = 4UL << L4_CAP_SHIFT,
00347 L4_BASE_LOG_CAP = 5UL << L4_CAP_SHIFT,
00349 L4_BASE_ICU_CAP = 6UL << L4_CAP_SHIFT,
00351 L4_BASE_SCHEDULER_CAP = 7UL << L4_CAP_SHIFT,
00358 L4_BASE_IOMMU_CAP = 8UL << L4_CAP_SHIFT,
00366 L4_BASE_DEBUGGER_CAP = 10UL << L4_CAP_SHIFT,
00373 L4_BASE_ARM_SMCCC_CAP = 11UL << L4_CAP_SHIFT,
00374
00376 L4_BASE_CAPS_LAST_P1,
00378 L4_BASE_CAPS_LAST = L4_BASE_CAPS_LAST_P1 - 1
00379 };
00380
00391 #define L4_PAGESIZE (1UL << L4_PAGESHIFT)
00392
00400 #define L4_PAGEMASK (~ (L4_PAGESIZE - 1))
00401
00409 #define L4_LOG2_PAGESIZE L4_PAGESHIFT
00410
00418 #define L4_SUPERPAGESIZE (1UL << L4_SUPERPAGESHIFT)
00419
00427 #define L4_SUPERPAGEMASK (~ (L4_SUPERPAGESIZE - 1))
00428
00435 #define L4_LOG2_SUPERPAGESIZE L4_SUPERPAGESHIFT
00436
00447 L4_INLINE l4_addr_t l4_trunc_page(l4_addr_t address) L4_NOTHROW;
00448 L4_INLINE l4_addr_t l4_trunc_page(l4_addr_t address) L4_NOTHROW
00449 { return address & L4_PAGEMASK; }
00450
00458 L4_INLINE l4_addr_t l4_trunc_size(l4_addr_t address, unsigned char bits) L4_NOTHROW;
00459 L4_INLINE l4_addr_t l4_trunc_size(l4_addr_t address, unsigned char bits) L4_NOTHROW
00460 { return address & (~0UL << bits); }
00461
00472 L4_INLINE l4_addr_t l4_round_page(l4_addr_t address) L4_NOTHROW;
00473 L4_INLINE l4_addr_t l4_round_page(l4_addr_t address) L4_NOTHROW
00474 { return (address + L4_PAGESIZE - 1) & L4_PAGEMASK; }
00475
00483 L4_INLINE l4_addr_t l4_round_size(l4_addr_t value, unsigned char bits) L4_NOTHROW;
00484 L4_INLINE l4_addr_t l4_round_size(l4_addr_t value, unsigned char bits) L4_NOTHROW
00485 { return (value + (1UL << bits) - 1) & (~0UL << bits); }
00486
00495 L4_INLINE unsigned l4_bytes_to_mwords(unsigned size) L4_NOTHROW;
00496 L4_INLINE unsigned l4_bytes_to_mwords(unsigned size) L4_NOTHROW
00497 { return (size + sizeof(l4_umword_t) - 1) / sizeof(l4_umword_t); }
00498
00503 enum l4_addr_consts_t {
00505 L4_INVALID_ADDR = ~0UL
00506 };
00507
00512 #define L4_INVALID_PTR ((void *)L4_INVALID_ADDR)
00513
00514 #ifndef NULL
00515 #ifndef __cplusplus
00516 # define NULL ((void *)0)
00520 #elif __cplusplus >= 201103L
00521 # define NULL nullptr
00522 #else
00523 # define NULL 0
00524 #endif
00525 #endif
00526
00527 #endif /* ! __L4_SYS__INCLUDE__CONSTS_H__ */

```

## 17.478 x86/I4/sys/consts.h File Reference

Common [L4](#) constants, x86 version.

## Macros

- `#define L4_PAGESHIFT 12`  
*Size of a page log2-based.*
- `#define L4_SUPERPAGESHIFT 22`  
*Size of a large page log2-based.*

### 17.478.1 Detailed Description

Common [L4](#) constants, x86 version.

Definition in file [consts.h](#).

## 17.479 consts.h

[Go to the documentation of this file.](#)

```

00001 /*****
00007 */
00008 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00009 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00010 * Björn Döbel <doebel@os.inf.tu-dresden.de>,
00011 * Lars Reuther <reuther@os.inf.tu-dresden.de>
00012 * economic rights: Technische Universität Dresden (Germany)
00013 *
00014 * License: see LICENSE.spdx (in this directory or the directories above)
00015 */
00016 /*****
00017 #ifndef __L4SYS__INCLUDE__ARCH_X86__CONSTS_H__
00018 #define __L4SYS__INCLUDE__ARCH_X86__CONSTS_H__
00019
00024 #define L4_PAGESHIFT 12
00025
00030 #define L4_SUPERPAGESHIFT 22
00031
00032 #include_next <l4/sys/consts.h>
00033
00034 #endif /* ! __L4SYS__INCLUDE__ARCH_X86__CONSTS_H__ */

```

## 17.480 capability.h

```

00001
00002 #pragma once
00003
00004 #include <l4/sys/consts.h>
00005 #include <l4/sys/types.h>
00006 #include <l4/sys/task.h>
00007
00008 namespace L4 {
00009
00010 class Task;
00011 class Kobject;
00012
00013 template< typename T > class L4_EXPORT Cap;
00014
00025 class L4_EXPORT Cap_base
00026 {
00027 public:
00029 enum No_init_type
00030 {
00034 No_init
00035 };
00036
00040 enum Cap_type
00041 {
00042 Invalid = L4_INVALID_CAP
00043 };
00044

```

```

00049 l4_cap_idx_t cap() const noexcept { return _c; }
00050
00057 bool is_valid() const noexcept { return !(_c & L4_INVALID_CAP_BIT); }
00058
00062 int invalid_cap_error() const noexcept { return _c & ~L4_INVALID_CAP_BIT; }
00063
00064 explicit operator bool () const noexcept
00065 { return !(_c & L4_INVALID_CAP_BIT); }
00066
00074 l4_fpage_t fpage(unsigned rights = L4_CAP_FPAGE_RWS) const noexcept
00075 { return l4_obj_fpage(_c, 0, rights); }
00076
00086 l4_umword_t snd_base(unsigned grant = L4_MAP_ITEM_MAP,
00087 l4_cap_idx_t base = L4_INVALID_CAP) const noexcept
00088 {
00089 if (base == L4_INVALID_CAP)
00090 base = _c;
00091 return l4_map_obj_control(base, grant);
00092 }
00093
00094
00098 bool operator == (Cap_base const &o) const noexcept
00099 { return _c == o._c; }
00100
00104 bool operator != (Cap_base const &o) const noexcept
00105 { return _c != o._c; }
00106
00120 inline l4_msgtag_t validate(l4_utcb_t *u = l4_utcb()) const noexcept;
00121
00136 inline l4_msgtag_t validate(Cap<Task> task,
00137 l4_utcb_t *u = l4_utcb()) const noexcept;
00138
00142 void invalidate() noexcept { _c = L4_INVALID_CAP; }
00143 protected:
00149 explicit Cap_base(l4_cap_idx_t c) noexcept : _c(c) {}
00153 explicit Cap_base(Cap_type cap) noexcept : _c(cap) {}
00154
00160 explicit Cap_base(l4_default_caps_t cap) noexcept : _c(cap) {}
00161
00165 explicit Cap_base() noexcept {}
00166
00176 void move(Cap_base const &src) const
00177 {
00178 if (!is_valid() || !src.is_valid())
00179 return;
00180
00181 l4_task_map(L4_BASE_TASK_CAP, L4_BASE_TASK_CAP, src.fpage(L4_CAP_FPAGE_RWSD),
00182 snd_base(L4_MAP_ITEM_GRANT) | L4_FPAGE_C_OBJ_RIGHTS);
00183 }
00184
00192 void copy(Cap_base const &src) const
00193 {
00194 if (!is_valid() || !src.is_valid())
00195 return;
00196
00197 l4_task_map(L4_BASE_TASK_CAP, L4_BASE_TASK_CAP, src.fpage(L4_CAP_FPAGE_RWSD),
00198 snd_base() | L4_FPAGE_C_OBJ_RIGHTS);
00199 }
00200
00203 l4_cap_idx_t _c;
00204 };
00205
00206
00222 template< typename T >
00223 class L4_EXPORT Cap : public Cap_base
00224 {
00225 private:
00226 friend class L4::Kobject;
00227
00239 explicit Cap(T const *p) noexcept
00240 : Cap_base(reinterpret_cast<l4_cap_idx_t>(p)) {}
00241
00242 public:
00243
00250 template< typename From >
00251 static void check_convertible_from() noexcept
00252 {
00253 using To = T;
00254 [[maybe_unused]] To* t = static_cast<From*>(nullptr);
00255 }
00256
00263 template< typename From >
00264 static void check_castable_from() noexcept
00265 {
00266 using To = T;
00267 [[maybe_unused]] To *t = static_cast<To *>(static_cast<From *>(nullptr));
00268 }

```

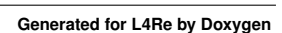
```

00269
00274 template< typename O >
00275 Cap(Cap<O> const &o) noexcept : Cap_base(o.cap())
00276 { check_convertible_from<O>(); }
00277
00282 Cap(Cap_type cap) noexcept : Cap_base(cap) {}
00283
00288 Cap(l4_default_caps_t cap) noexcept : Cap_base(cap) {}
00289
00294 explicit Cap(l4_cap_idx_t idx = L4_INVALID_CAP) noexcept : Cap_base(idx) {}
00295
00299 explicit Cap(No_init_type) noexcept {}
00300
00307 Cap move(Cap const &src) const
00308 {
00309 Cap_base::move(src);
00310 return *this;
00311 }
00312
00317 Cap copy(Cap const &src) const
00318 {
00319 Cap_base::copy(src);
00320 return *this;
00321 }
00322
00326 T *operator -> () const noexcept { return reinterpret_cast<T*>(_c); }
00327 };
00328
00329
00340 template<
00341 class L4_EXPORT Cap<void> : public Cap_base
00342 {
00343 public:
00344
00345 explicit Cap(void const *p) noexcept
00346 : Cap_base(reinterpret_cast<l4_cap_idx_t>(p)) {}
00347
00351 Cap(Cap_type cap) noexcept : Cap_base(cap) {}
00352
00357 Cap(l4_default_caps_t cap) noexcept : Cap_base(cap) {}
00358
00363 explicit Cap(l4_cap_idx_t idx = L4_INVALID_CAP) noexcept : Cap_base(idx) {}
00364 explicit Cap(No_init_type) noexcept {}
00365
00366 template< typename From >
00367 static void check_convertible_from() noexcept {}
00368
00369 template< typename From >
00370 static void check_castable_from() noexcept {}
00371
00378 Cap move(Cap const &src) const
00379 {
00380 Cap_base::move(src);
00381 return *this;
00382 }
00383
00388 Cap copy(Cap const &src) const
00389 {
00390 Cap_base::copy(src);
00391 return *this;
00392 }
00393
00394 template< typename T >
00395 Cap(Cap<T> const &o) noexcept : Cap_base(o.cap()) {}
00396 };
00397
00414 template< typename T, typename F >
00415 inline
00416 Cap<T> cap_cast(Cap<F> const &c) noexcept
00417 {
00418 Cap<T>::template check_castable_from<F>();
00419 return Cap<T>(c.cap());
00420 }
00421
00422 // gracefully deal with L4::Kobject ambiguity
00423 template< typename T >
00424 inline
00425 Cap<T> cap_cast(Cap<L4::Kobject> const &c) noexcept
00426 {
00427 return Cap<T>(c.cap());
00428 }
00429
00445 template< typename T, typename F >
00446 inline
00447 Cap<T> cap_reinterpret_cast(Cap<F> const &c) noexcept
00448 {
00449 return Cap<T>(c.cap());

```

## 17.481 I4/re/consts File Reference

Include dependency graph for consts:



## 17.481.1 Detailed Description

Constants.

Definition in file [consts](#).

## 17.482 consts

[Go to the documentation of this file.](#)

```
00001 // -*- Mode: C++ -*-
00002 // vim:ft=cpp
00007 /*
00008 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #pragma once
00014
00015 #include <l4/sys/capability>
00016 #include <l4/sys/consts.h>
00017 #include <l4/re/env.h>
00018
00019 namespace L4Re {
00020 static L4::Cap<L4::Task>::Cap_type const This_task
00021 = static_cast<L4::Cap<L4::Task>::Cap_type>(L4RE_THIS_TASK_CAP);
00022 }
```

## 17.483 consts

```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002
00003 #pragma once
00004
00005 #include <l4/sys/consts.h>
00006
00007 namespace L4 {
00008
00017 template<typename T>
00018 constexpr T trunc_order(T val, unsigned char order)
00019 {
00020 return val & ((~T(0)) << order);
00021 }
00022
00031 template<typename T>
00032 constexpr T round_order(T val, unsigned char order)
00033 {
00034 return (val + (T(1) << order) - T(1)) & ((~T(0)) << order);
00035 }
00036
00037 template<typename T>
00038 constexpr T trunc_page(T val)
00039 {
00040 return trunc_order(val, L4_PAGESHIFT);
00041 }
00042
00043 template<typename T>
00044 constexpr T round_page(T val)
00045 {
00046 return round_order(val, L4_PAGESHIFT);
00047 }
00048
00049 template<typename T>
00050 inline unsigned char
00051 max_order(unsigned char order, T addr,
00052 T min_addr, T max_addr,
00053 T hotspot = T(0))
00054 {
00055 while (order < 30 /* limit to 1GB flexpages */)
00056 {
00057 T mask;
00058 T base = trunc_order(addr, order + 1);
00059 if (base < min_addr)
```

```

00060 return order;
00061
00062 if (base + (T(1) < (order + 1)) - T(1) > max_addr - T(1))
00063 return order;
00064
00065 mask = ~((~T(0)) < (order + 1));
00066 if (hotspot == ~T(0) || ((addr ^ hotspot) & mask))
00067 break;
00068
00069 ++order;
00070 }
00071
00072 return order;
00073 }
00074
00075 }

```

## 17.484 ipc\_array

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2014 Alexander Warg <alexander.warg@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include "types"
00010 #include "ipc_basics"
00011 #include "ipc_types"
00012
00013 namespace L4 { namespace Ipc L4_EXPORT {
00014
00016 typedef unsigned short Array_len_default;
00017
00027 template<typename ELEM_TYPE, typename LEN_TYPE = Array_len_default >
00028 struct Array_ref
00029 {
00030 typedef ELEM_TYPE *ptr_type;
00031 typedef LEN_TYPE len_type;
00032
00033 len_type length;
00034 ptr_type data;
00035 Array_ref() = default;
00036 Array_ref(len_type length, ptr_type data)
00037 : length(length), data(data)
00038 {}
00039
00040 template<typename X> struct Non_const
00041 { typedef Array_ref<X, LEN_TYPE> type; };
00042
00043 template<typename X> struct Non_const<X const>
00044 { typedef Array_ref<X, LEN_TYPE> type; };
00045
00046 Array_ref(typename Non_const<ELEM_TYPE>::type const &other)
00047 : length(other.length), data(other.data)
00048 {}
00049
00050 Array_ref &operator = (typename Non_const<ELEM_TYPE>::type const &other)
00051 {
00052 this->length = other.length;
00053 this->data = other.data;
00054 return *this;
00055 }
00056 };
00057
00080 template<typename ELEM_TYPE, typename LEN_TYPE = Array_len_default>
00081 struct Array : Array_ref<ELEM_TYPE, LEN_TYPE>
00082 {
00084 Array() {}
00086 Array(LEN_TYPE length, ELEM_TYPE *data)
00087 : Array_ref<ELEM_TYPE, LEN_TYPE>(length, data)
00088 {}
00089
00090 template<typename X> struct Non_const
00091 { typedef Array<X, LEN_TYPE> type; };
00092
00093 template<typename X> struct Non_const<X const>
00094 { typedef Array<X, LEN_TYPE> type; };
00095
00097 Array(typename Non_const<ELEM_TYPE>::type const &other)
00098 : Array_ref<ELEM_TYPE, LEN_TYPE>(other.length, other.data)
00099 {}

```



```

00100
00101 Array &operator = (typename Non_const<ELEM_TYPE>::type const &other)
00102 {
00103 this->length = other.length;
00104 this->data = other.data;
00105 return *this;
00106 }
00107 };
00108
00122 template< typename ELEM_TYPE,
00123 typename LEN_TYPE = Array_len_default,
00124 LEN_TYPE MAX = (L4_UTCB_GENERIC_DATA_SIZE *
00125 sizeof(l4_umword_t)) / sizeof(ELEM_TYPE) >
00126 struct Array_in_buf
00127 {
00128 typedef Array_ref<ELEM_TYPE, LEN_TYPE> array;
00129 typedef Array_ref<ELEM_TYPE const, LEN_TYPE> const_array;
00130
00132 ELEM_TYPE data[MAX];
00134 LEN_TYPE length;
00135
00137 void copy_in(const_array a)
00138 {
00139 length = a.length;
00140 if (length > MAX)
00141 length = MAX;
00142
00143 for (LEN_TYPE i = 0; i < length; ++i)
00144 data[i] = a.data[i];
00145 }
00146
00148 Array_in_buf(const_array a) { copy_in(a); }
00150 Array_in_buf(array a) { copy_in(a); }
00151 };
00152
00153 // implementation details for transmission
00154 namespace Msg {
00155
00157 template<typename A, typename LEN>
00158 struct Elem< Array<A, LEN> >
00159 {
00161 typedef Array<A, LEN> arg_type;
00163 typedef Array_ref<A, LEN> svr_type;
00164 typedef svr_type svr_arg_type;
00165 enum { Is_optional = false };
00166 };
00167
00169 template<typename A, typename LEN>
00170 struct Elem< Array<A, LEN> & >
00171 {
00173 typedef Array<A, LEN> &arg_type;
00175 typedef Array_ref<A, LEN> svr_type;
00177 typedef svr_type &svr_arg_type;
00178 enum { Is_optional = false };
00179 };
00180
00182 template<typename A, typename LEN>
00183 struct Elem< Array_ref<A, LEN> & >
00184 {
00186 typedef Array_ref<A, LEN> &arg_type;
00188 typedef Array_ref<typename L4::Types::Remove_const<A>::type, LEN> svr_type;
00190 typedef svr_type &svr_arg_type;
00191 enum { Is_optional = false };
00192 };
00193
00194 template<typename A> struct Class<Array<A> > : Class<A>::type {};
00195 template<typename A> struct Class<Array_ref<A> > : Class<A>::type {};
00196
00197 namespace Detail {
00198
00199 template<typename A, typename LEN, typename ARRAY, bool REF>
00200 struct Clnt_val_ops_d_in : Clnt_noops<ARRAY>
00201 {
00202 using Clnt_noops<ARRAY>::to_msg;
00203 static int to_msg(char *msg, unsigned offset, unsigned limit,
00204 ARRAY a, Dir_in, Cls_data)
00205 {
00206 offset = align_to<LEN>(offset);
00207 if (L4_UNLIKELY(!check_size<LEN>(offset, limit)))
00208 return -L4_MSGTOOLONG;
00209 *reinterpret_cast<LEN *>(msg + offset) = a.length;
00210 offset = align_to<A>(offset + sizeof(LEN));
00211 if (L4_UNLIKELY(!check_size<A>(offset, limit, a.length)))
00212 return -L4_MSGTOOLONG;
00213 typedef typename L4::Types::Remove_const<A>::type elem_type;
00214 elem_type *data = reinterpret_cast<elem_type*>(msg + offset);
00215

```

```

00216 // we do not correctly handle overlaps
00217 if (!REF || data != a.data)
00218 {
00219 for (LEN i = 0; i < a.length; ++i)
00220 data[i] = a.data[i];
00221 }
00222
00223 return offset + a.length * sizeof(A);
00224 }
00225 };
00226 } // namespace Detail
00227
00228 template<typename A, typename LEN>
00229 struct Clnt_val_ops<Array<A, LEN>, Dir_in, Cls_data> :
00230 Detail::Clnt_val_ops_d_in<A, LEN, Array<A, LEN>, false> {};
00231
00232 template<typename A, typename LEN>
00233 struct Clnt_val_ops<Array_ref<A, LEN>, Dir_in, Cls_data> :
00234 Detail::Clnt_val_ops_d_in<A, LEN, Array_ref<A, LEN>, true> {};
00235
00236 template<typename A, typename LEN, typename CLASS>
00237 struct Svr_val_ops<Array_ref<A, LEN>, Dir_in, CLASS> :
00238 Svr_noops<Array_ref<A, LEN>>
00239 {
00240 typedef Array_ref<A, LEN> svr_type;
00241
00242 using Svr_noops<svr_type>::to_svr;
00243 static int to_svr(char *msg, unsigned offset, unsigned limit,
00244 svr_type &a, Dir_in, Cls_data)
00245 {
00246 offset = align_to<LEN>(offset);
00247 if (L4_UNLIKELY(!check_size<LEN>(offset, limit)))
00248 return -L4_MSGTOOSHORT;
00249 a.length = *reinterpret_cast<LEN*>(msg + offset);
00250 offset = align_to<A>(offset + sizeof(LEN));
00251 if (L4_UNLIKELY(!check_size<A>(offset, limit, a.length)))
00252 return -L4_MSGTOOSHORT;
00253 a.data = reinterpret_cast<A*>(msg + offset);
00254 return offset + a.length * sizeof(A);
00255 }
00256 };
00257
00258 template<typename A, typename LEN>
00259 struct Svr_xmit<Array<A, LEN>> : Svr_xmit<Array_ref<A, LEN>> {};
00260
00261 template<typename A, typename LEN>
00262 struct Clnt_val_ops<Array<A, LEN>, Dir_out, Cls_data> : Clnt_noops<Array<A, LEN>>
00263 {
00264 typedef Array<A, LEN> type;
00265
00266 using Clnt_noops<type>::from_msg;
00267 static int from_msg(char *msg, unsigned offset, unsigned limit, long,
00268 type &a, Dir_out, Cls_data)
00269 {
00270 offset = align_to<LEN>(offset);
00271 if (L4_UNLIKELY(!check_size<LEN>(offset, limit)))
00272 return -L4_MSGTOOSHORT;
00273
00274 LEN l = *reinterpret_cast<LEN*>(msg + offset);
00275
00276 offset = align_to<A>(offset + sizeof(LEN));
00277 if (L4_UNLIKELY(!check_size<A>(offset, limit, l)))
00278 return -L4_MSGTOOSHORT;
00279
00280 A *data = reinterpret_cast<A*>(msg + offset);
00281
00282 if (l > a.length)
00283 l = a.length;
00284 else
00285 a.length = l;
00286
00287 for (unsigned i = 0; i < l; ++i)
00288 a.data[i] = data[i];
00289
00290 return offset + l * sizeof(A);
00291 };
00292 };
00293
00294 template<typename A, typename LEN>
00295 struct Clnt_val_ops<Array_ref<A, LEN>, Dir_out, Cls_data> :
00296 Clnt_noops<Array_ref<A, LEN>>
00297 {
00298 typedef Array_ref<A, LEN> type;
00299
00300 using Clnt_noops<type>::from_msg;
00301 static int from_msg(char *msg, unsigned offset, unsigned limit, long,
00302 type &a, Dir_out, Cls_data)

```

```

00303 {
00304 offset = align_to<LEN>(offset);
00305 if (L4_UNLIKELY(!check_size<LEN>(offset, limit)))
00306 return -L4_EMMSGTOOSHORT;
00307
00308 LEN l = *reinterpret_cast<LEN *>(msg + offset);
00309
00310 offset = align_to<A>(offset + sizeof(LEN));
00311 if (L4_UNLIKELY(!check_size<A>(offset, limit, 1)))
00312 return -L4_EMMSGTOOSHORT;
00313
00314 a.data = reinterpret_cast<A*>(msg + offset);
00315 a.length = l;
00316 return offset + 1 * sizeof(A);
00317 };
00318 };
00319
00320 template<typename A, typename LEN, typename CLASS>
00321 struct Svr_val_ops<Array_ref<A, LEN>, Dir_out, CLASS> :
00322 Svr_noops<Array_ref<typename L4::Types::Remove_const<A>::type, LEN> &>
00323 {
00324 typedef typename L4::Types::Remove_const<A>::type elem_type;
00325 typedef Array_ref<elem_type, LEN> &svr_type;
00326
00327 using Svr_noops<svr_type>::to_svr;
00328 static int to_svr(char *msg, unsigned offset, unsigned limit,
00329 svr_type a, Dir_out, Cls_data)
00330 {
00331 offset = align_to<LEN>(offset);
00332 if (L4_UNLIKELY(!check_size<LEN>(offset, limit)))
00333 return -L4_EMMSGTOOLONG;
00334
00335 offset = align_to<A>(offset + sizeof(LEN));
00336 a.data = reinterpret_cast<elem_type *>(msg + offset);
00337 a.length = (limit - offset) / sizeof(A);
00338 return offset;
00339 }
00340
00341 using Svr_noops<svr_type>::from_svr;
00342 static int from_svr(char *msg, unsigned offset, unsigned limit, long,
00343 svr_type a, Dir_out, Cls_data)
00344 {
00345 offset = align_to<LEN>(offset);
00346 if (L4_UNLIKELY(!check_size<LEN>(offset, limit)))
00347 return -L4_EMMSGTOOLONG;
00348
00349 *reinterpret_cast<LEN *>(msg + offset) = a.length;
00350
00351 offset = align_to<A>(offset + sizeof(LEN));
00352 if (L4_UNLIKELY(!check_size<A>(offset, limit, a.length)))
00353 return -L4_EMMSGTOOLONG;
00354
00355 return offset + a.length * sizeof(A);
00356 }
00357 };
00358
00359 template<typename A, typename LEN>
00360 struct Svr_xmit<Array<A, LEN> &> : Svr_xmit<Array_ref<A, LEN> &> {};
00361
00362 // Pointer to array is not implemented.
00363 template<typename A, typename LEN>
00364 struct Is_valid_rpc_type<Array_ref<A, LEN> *> : L4::Types::False {};
00365
00366 // Optional input arrays are not implemented.
00367 template<typename A, typename LEN>
00368 struct Is_valid_rpc_type<Opt<Array_ref<A, LEN> > > : L4::Types::False {};
00369
00370 } // namespace Msg
00371
00372 }

```

## 17.485 ipc\_basics

```

00001 // vi:set ft=c++ -- Mode: C++ --
00002 /*
00003 * (c) 2014 Alexander Warg <alexander.warg@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include "types"
00010 #include <l4/sys/utcb.h>

```

```

00011 #include <l4/sys/err.h>
00012
00013 namespace L4 {
00014
00016 namespace Ipc {
00017
00019 namespace Msg {
00020
00021 using L4::Types::True;
00022 using L4::Types::False;
00023
00030 constexpr unsigned long align_to(unsigned long bytes, unsigned long align) noexcept
00031 { return (bytes + align - 1) & ~(align - 1); }
00032
00039 template<typename T>
00040 constexpr unsigned long align_to(unsigned long bytes) noexcept
00041 { return align_to(bytes, __alignof(T)); }
00042
00052 template<typename T>
00053 constexpr bool check_size(unsigned offset, unsigned limit) noexcept
00054 {
00055 return offset + sizeof(T) <= limit;
00056 }
00057
00070 template<typename T, typename CTYPE>
00071 inline bool check_size(unsigned offset, unsigned limit, CTYPE cnt) noexcept
00072 {
00073 if (L4_UNLIKELY(sizeof(CTYPE) <= sizeof(unsigned) &&
00074 ~0U / sizeof(T) <= static_cast<unsigned>(cnt)))
00075 return false;
00076
00077 if (L4_UNLIKELY(sizeof(CTYPE) > sizeof(unsigned) &&
00078 static_cast<CTYPE>(~0U / sizeof(T)) <= cnt))
00079 return false;
00080
00081 return sizeof(T) * cnt <= limit - offset;
00082 }
00083
00084
00085 enum
00086 {
00088 Word_bytes = sizeof(l4_umword_t),
00090 Item_words = 2,
00092 Item_bytes = Word_bytes * Item_words,
00094 Mr_words = L4_UTCB_GENERIC_DATA_SIZE,
00096 Mr_bytes = Word_bytes * Mr_words,
00098 Br_bytes = Word_bytes * L4_UTCB_GENERIC_BUFFERS_SIZE,
00099 };
00100
00101
00113 template<typename T>
00114 inline int msg_add(char *msg, unsigned offs, unsigned limit, T v) noexcept
00115 {
00116 offs = align_to<T>(offs);
00117 if (L4_UNLIKELY(!check_size<T>(offs, limit)))
00118 return -L4_EMMSGTOOLONG;
00119 *reinterpret_cast<typename L4::Types::Remove_const<T>::type *>(msg + offs) = v;
00120 return offs + sizeof(T);
00121 }
00122
00134 template<typename T>
00135 inline int msg_get(char *msg, unsigned offs, unsigned limit, T &v) noexcept
00136 {
00137 offs = align_to<T>(offs);
00138 if (L4_UNLIKELY(!check_size<T>(offs, limit)))
00139 return -L4_EMMSGTOOSHORT;
00140 v = *reinterpret_cast<T *>(msg + offs);
00141 return offs + sizeof(T);
00142 }
00143
00145 struct Dir_in { typedef Dir_in type; typedef Dir_in dir; };
00147 struct Dir_out { typedef Dir_out type; typedef Dir_out dir; };
00148
00150 struct Cls_data { typedef Cls_data type; typedef Cls_data cls; };
00152 struct Cls_item { typedef Cls_item type; typedef Cls_item cls; };
00154 struct Cls_buffer { typedef Cls_buffer type; typedef Cls_buffer cls; };
00155
00156 // Typical combinations
00158 struct Do_in_data : Dir_in, Cls_data {};
00160 struct Do_out_data : Dir_out, Cls_data {};
00162 struct Do_in_items : Dir_in, Cls_item {};
00164 struct Do_out_items : Dir_out, Cls_item {};
00166 struct Do_rcv_buffers : Dir_in, Cls_buffer {};
00167
00168 // implementation details
00169 namespace Detail {
00170

```

```

00171 template<typename T> struct _Plain
00172 {
00173 typedef T type;
00174 static T deref(T v) noexcept { return v; }
00175 };
00176
00177 template<typename T> struct _Plain<T *>
00178 {
00179 typedef T type;
00180 static T &deref(T *v) noexcept { return *v; }
00181 };
00182
00183 template<typename T> struct _Plain<T &>
00184 {
00185 typedef T type;
00186 static T &deref(T &v) noexcept { return v; }
00187 };
00188
00189
00190 template<typename T> struct _Plain<T const *>
00191 {
00192 typedef T type;
00193 static T const &deref(T const *v) noexcept { return *v; }
00194 };
00195
00196 template<typename T> struct _Plain<T const &>
00197 {
00198 typedef T type;
00199 static T const &deref(T const &v) noexcept { return v; }
00200 };
00201
00202
00210 template<typename MTYPE, typename DIR, typename CLASS> struct Clnt_val_ops;
00211
00212 template<typename T> struct Clnt_noops
00213 {
00214 template<typename A, typename B>
00215 static constexpr int to_msg(char *, unsigned offset, unsigned, T, A, B) noexcept
00216 { return offset; }
00217
00218 template<typename A, typename B>
00219 static constexpr int from_msg(char *, unsigned offset, unsigned, long, T const &, A, B) noexcept
00220 { return offset; }
00221 };
00222
00223
00224 template<typename T> struct Svr_noops
00225 {
00226 template<typename A, typename B>
00227 static constexpr int from_svr(char *, unsigned offset, unsigned, long, T, A, B) noexcept
00228 { return offset; }
00229
00230 template<typename A, typename B>
00231 static constexpr int to_svr(char *, unsigned offset, unsigned, T, A, B) noexcept
00232 { return offset; }
00233 };
00234
00235
00236 template<typename MTYPE, typename CLASS>
00237 struct Clnt_val_ops<MTYPE, Dir_in, CLASS> : Clnt_noops<MTYPE>
00238 {
00239 using Clnt_noops<MTYPE>::to_msg;
00240 static int to_msg(char *msg, unsigned offset, unsigned limit,
00241 MTYPE arg, Dir_in, CLASS) noexcept
00242 { return msg_add<MTYPE>(msg, offset, limit, arg); }
00243 };
00244
00245
00246
00247 template<typename MTYPE, typename CLASS>
00248 struct Clnt_val_ops<MTYPE, Dir_out, CLASS> : Clnt_noops<MTYPE>
00249 {
00250 using Clnt_noops<MTYPE>::from_msg;
00251 static int from_msg(char *msg, unsigned offset, unsigned limit, long,
00252 MTYPE &arg, Dir_out, CLASS) noexcept
00253 { return msg_get<MTYPE>(msg, offset, limit, arg); }
00254 };
00255
00256
00264 template<typename MTYPE, typename DIR, typename CLASS> struct Svr_val_ops;
00265
00266 template<typename MTYPE, typename CLASS>
00267 struct Svr_val_ops<MTYPE, Dir_in, CLASS> : Svr_noops<MTYPE>
00268 {
00269 using Svr_noops<MTYPE>::to_svr;
00270 static int to_svr(char *msg, unsigned offset, unsigned limit,
00271 MTYPE &arg, Dir_in, CLASS) noexcept
00272 { return msg_get<MTYPE>(msg, offset, limit, arg); }
00273 };
00274
00275
00276 template<typename MTYPE, typename CLASS>

```

```

00277 struct Svr_val_ops<MTYPE, Dir_out, CLASS> : Svr_noops<MTYPE>
00278 {
00279 using Svr_noops<MTYPE>::to_svr;
00280 static int to_svr(char *, unsigned offs, unsigned limit,
00281 MTYPE &, Dir_out, CLASS) noexcept
00282 {
00283 offs = align_to<MTYPE>(offs);
00284 if (L4_UNLIKELY(!check_size<MTYPE>(offs, limit)))
00285 return -L4_MSGTOOLONG;
00286 return offs + sizeof(MTYPE);
00287 }
00288
00289 using Svr_noops<MTYPE>::from_svr;
00290 static int from_svr(char *msg, unsigned offset, unsigned limit, long,
00291 MTYPE arg, Dir_out, CLASS) noexcept
00292 { return msg_add<MTYPE>(msg, offset, limit, arg); }
00293 };
00294
00295 template<typename T> struct Elem
00296 {
00297 typedef T arg_type;
00298 typedef T svr_type;
00300 typedef T svr_arg_type; // might by const & (depending on the size)
00301
00302 enum { Is_optional = false };
00303 };
00304
00305 template<typename T> struct Elem<T &>
00306 {
00307 typedef T &arg_type;
00308 typedef T svr_type;
00309 typedef T &svr_arg_type;
00310 enum { Is_optional = false };
00311 };
00312
00313 template<typename T> struct Elem<T const &>
00314 {
00315 typedef T const &arg_type;
00316 typedef T svr_type;
00317 // as the RPC uses a const reference we use it here too,
00318 // we could also use pass by value depending on the size
00319 typedef T const &svr_arg_type;
00320 enum { Is_optional = false };
00321 };
00322
00323 template<typename T> struct Elem<T *> : Elem<T &>
00324 {
00325 typedef T *arg_type;
00326 };
00327
00328 template<typename T> struct Elem<T const *> : Elem<T const &>
00329 {
00330 typedef T const *arg_type;
00331 };
00332
00333 template<typename T> struct Is_valid_rpc_type : L4::Types::True {};
00334
00335 // Static assertions outside functions work only properly from C++11
00336 // onwards. On earlier version make sure the compiler fails on an ugly
00337 // undefined struct instead.
00338 template<typename T, bool B> struct Error_invalid_rpc_parameter_used;
00339 template<typename T> struct Error_invalid_rpc_parameter_used<T, true> {};
00340
00341 #if __cplusplus >= 201103L
00342 template<typename T>
00343 struct _Elem : Elem<T>
00344 {
00345 static_assert(Is_valid_rpc_type<T>::value,
00346 "L4::Rpc::Msg::_Elem<T>: type T is not a valid RPC parameter type.");
00347 };
00348 #else
00349 template<typename T>
00350 struct _Elem : Elem<T>, Error_invalid_rpc_parameter_used<T, Is_valid_rpc_type<T>::value>
00351 {};
00352 #endif
00353
00354 template<typename T> struct Class : Cls_data {};
00355 template<typename T> struct Direction : Dir_in {};
00356 template<typename T> struct Direction<T const &> : Dir_in {};
00357 template<typename T> struct Direction<T const *> : Dir_in {};
00358 template<typename T> struct Direction<T &> : Dir_out {};
00359 template<typename T> struct Direction<T *> : Dir_out {};
00360
00361 template<typename T> struct _Clnt_noops :

```

```

00370 Clnt_noops<typename Detail::_Plain<typename _Elem<T>::arg_type>::type>
00371 {};
00372
00373 namespace Detail {
00374
00375 template<typename T, typename DIR, typename CLASS>
00376 struct _Clnt_val_ops :
00377 Clnt_val_ops<typename Detail::_Plain<T>::type, DIR, CLASS> {};
00378
00379 template<typename T,
00380 typename ELEM = _Elem<T>,
00381 typename CLNT_OPS = _Clnt_val_ops<typename ELEM::arg_type,
00382 typename Direction<T>::type,
00383 typename Class<typename Detail::_Plain<T>::type>::type>
00384 >
00385 struct _Clnt_xmit : CLNT_OPS {};
00386
00387 template<typename T,
00388 typename ELEM = _Elem<T>,
00389 typename SVR_OPS = Svr_val_ops<typename ELEM::svr_type,
00390 typename Direction<T>::type,
00391 typename Class<typename Detail::_Plain<T>::type>::type>
00392 >
00393 struct _Svr_xmit : SVR_OPS {};
00394
00395 } //namespace Detail
00396 template<typename T> struct Clnt_xmit : Detail::_Clnt_xmit<T> {};
00397 template<typename T> struct Svr_xmit : Detail::_Svr_xmit<T> {};
00398
00399 }}} // namespace Msg, Ipc, L4
00400
00401

```

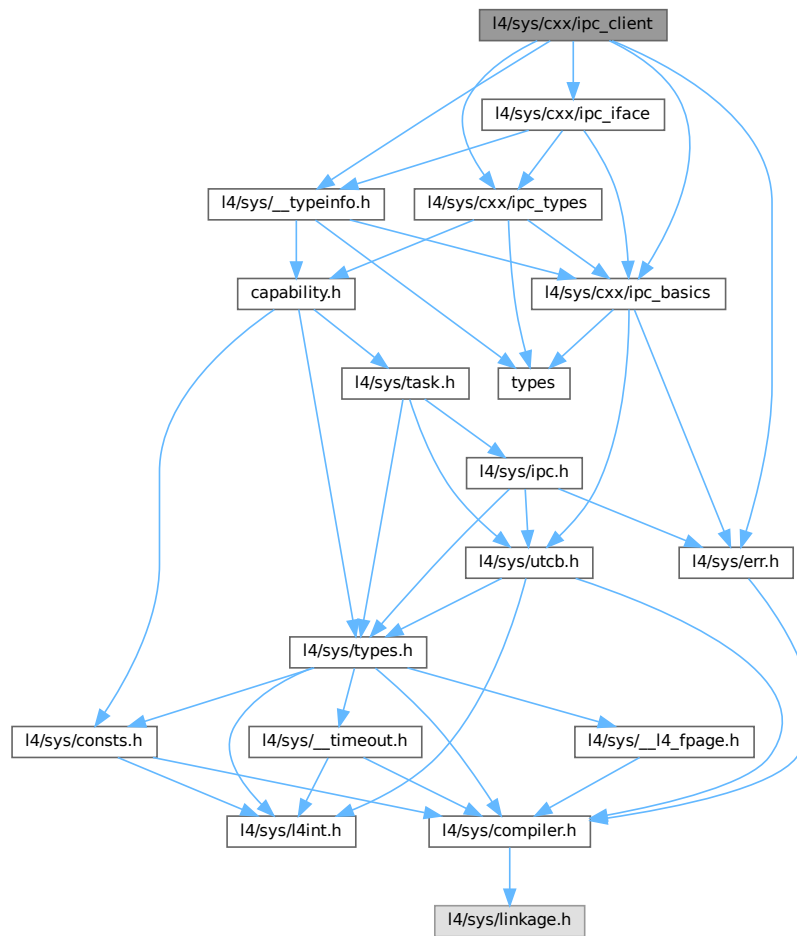
## 17.486 l4/sys/cxx/ipc\_client File Reference

```

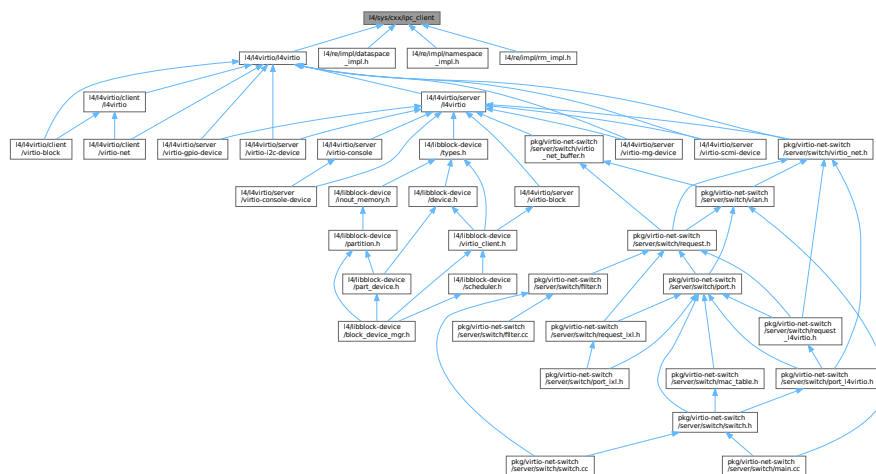
#include <l4/sys/cxx/ipc_basics>
#include <l4/sys/cxx/ipc_types>
#include <l4/sys/cxx/ipc_iface>
#include <l4/sys/__typeinfo.h>
#include <l4/sys/err.h>

```

Include dependency graph for ipc\_client:



This graph shows which files directly or indirectly include this file:





## Namespaces

- namespace [L4](#)  
*L4 low-level kernel interface.*
- namespace [L4::ipc](#)  
*IPC related functionality.*
- namespace [L4::ipc::Msg](#)  
*IPC Message related functionality.*

## Macros

- `#define L4_RPC_DEF(name)`  
*Generate the definition of an RPC stub.*

## 17.486.1 Macro Definition Documentation

### 17.486.1.1 L4\_RPC\_DEF

```
#define L4_RPC_DEF(
 name)
```

#### Value:

```
template struct L4::Ipcc::Msg::Rpc_call \
 <name##_t, name##_t::class_type, name##_t::ipc_type, name##_t::flags_type>
```

Generate the definition of an RPC stub.

#### Parameters

|             |                                                                                            |
|-------------|--------------------------------------------------------------------------------------------|
| <i>name</i> | The fully qualified method name to be implemented, this means <code>class::method</code> . |
|-------------|--------------------------------------------------------------------------------------------|

This macro generates the definition (implementation) for the given RPC interface method.

Definition at line 32 of file [ipc\\_client](#).

## 17.487 ipc\_client

[Go to the documentation of this file.](#)

```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2014 Alexander Warg <alexander.warg@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008 #pragma GCC system_header
00009
00010 #include <l4/sys/cxx/ipc_basics>
00011 #include <l4/sys/cxx/ipc_types>
00012 #include <l4/sys/cxx/ipc_iface>
00013 #include <l4/sys/__typeinfo.h>
00014 #include <l4/sys/err.h>
00015
00019
```

```

00020 namespace L4 { namespace Ipc { namespace Msg {
00021 //-----
00022
00032 #define L4_RPC_DEF(name) \
00033 template struct L4::Ipc::Msg::Rpc_call \
00034 <name##_t, name##_t::class_type, name##_t::ipc_type, name##_t::flags_type>
00035
00036
00038 //-----
00039 //Implementation of the RPC call
00040 template<typename OP, typename C, typename FLAGS, typename R, typename ...ARGS>
00041 R L4_EXPORT
00042 Rpc_call<OP, C, R (ARGS...), FLAGS>::
00043 call(L4::Cap<C> cap, typename _Elem<ARGS>::arg_type ...a, l4_utcb_t *utcb) noexcept
00044 {
00045 return Rpc_inline_call<OP, C, R (ARGS...), FLAGS>::call(cap, a..., utcb);
00046 }
00048
00049 } // namespace Msg
00050 } // namespace Ipc
00051 } // namespace L4
00052

```

## 17.488 ipc\_epiface

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2014-2015 Alexander Warg <alexander.warg@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008 #pragma GCC system_header
00009
00010 #include "capability.h"
00011 #include "ipc_server"
00012 #include "ipc_string"
00013 #include <l4/sys/types.h>
00014 #include <l4/sys/utcb.h>
00015 #include <l4/sys/__typeinfo.h>
00016 #include <l4/sys/meta>
00017 #include <l4/cxx/type_traits>
00018
00019 namespace L4 {
00020
00021 // forward for Irqep_t
00022 class Irq;
00023 class Rcv_endpoint;
00024
00025 namespace Ipc_svr {
00026
00027 class Timeout;
00028
00036 class Server_iface
00037 {
00038 private:
00039 Server_iface(Server_iface const &);
00040 Server_iface const &operator = (Server_iface const &);
00041
00042 public:
00044 typedef L4::Type_info::Demand Demand;
00045
00046 Server_iface(Server_iface &&) = delete;
00047 Server_iface &operator = (Server_iface &&) = delete;
00048
00050 Server_iface() {}
00051
00052 // Destroy the server interface
00053 virtual ~Server_iface() = 0;
00054
00064 virtual int alloc_buffer_demand(Demand const &demand) = 0;
00065
00074 virtual L4::Cap<void> get_rcv_cap(int index) const = 0;
00075
00084 virtual int realloc_rcv_cap(int index) = 0;
00085
00093 virtual int add_timeout(Timeout *timeout, l4_kernel_clock_t time) = 0;
00094
00100 virtual int remove_timeout(Timeout *timeout) = 0;
00101
00113 template<typename T>
00114 L4::Cap<T> rcv_cap(int index) const
00115 { return L4::cap_cast<T>(get_rcv_cap(index)); }

```

```

00116
00126 L4::Cap<void> rcv_cap(int index) const
00127 { return get_rcv_cap(index); }
00128 };
00129
00130 inline Server_iface::~Server_iface() {}
00131
00132 } // namespace Ipc_svr
00133
00145 struct Epiface
00146 {
00147 Epiface(Epiface const &) = delete;
00148 Epiface &operator = (Epiface const &) = delete;
00149
00151 typedef Ipc_svr::Server_iface Server_iface;
00153 typedef Ipc_svr::Server_iface::Demand Demand;
00154
00155 class Stored_cap : public Cap<void>
00156 {
00157 private:
00158 enum { Managed = 0x10 };
00159
00160 public:
00161 Stored_cap() = default;
00162 Stored_cap(Cap<void> const &c, bool managed = false)
00163 : Cap<void>((c.cap() & L4_CAP_MASK) | (managed ? Managed : 0))
00164 {
00165 static_assert (!(L4_CAP_MASK & Managed), "conflicting bits used...");
00166 }
00167
00168 bool managed() const { return cap() & Managed; }
00169 };
00170
00172 Epiface() : _data(0) {}
00173
00186 virtual l4_msgtag_t dispatch(l4_msgtag_t tag, unsigned rights,
00187 l4_utcb_t *utcb) = 0;
00188
00195 virtual Demand get_buffer_demand() const = 0; //{ return Demand(0); }
00196
00198 virtual ~Epiface() = 0;
00199
00206 Stored_cap obj_cap() const { return _cap; }
00207
00213 Server_iface *server_iface() const { return _data; }
00214
00224 int set_server(Server_iface *srv, Cap<void> cap, bool managed = false)
00225 {
00226 if ((srv && cap) || (!srv && !cap))
00227 {
00228 _data = srv;
00229 _cap = Stored_cap(cap, managed);
00230 return 0;
00231 }
00232
00233 return -L4_EINVAL;
00234 }
00235
00239 void set_obj_cap(Cap<void> const &cap) { _cap = cap; }
00240
00241 private:
00242 Server_iface *_data;
00243 Stored_cap _cap;
00244 };
00245
00246 inline Epiface::~Epiface() {}
00247
00255 template<typename RPC_IFACE, typename BASE = Epiface>
00256 struct Epiface_t0 : BASE
00257 {
00259 typedef RPC_IFACE Interface;
00260
00262 typename Type_info::Demand get_buffer_demand() const
00263 { return typename Kobject_typeid<RPC_IFACE>::Demand(); }
00264
00269 Cap<RPC_IFACE> obj_cap() const
00270 { return L4::cap_cast<RPC_IFACE>(BASE::obj_cap()); }
00271 };
00272
00280 template<typename Derived, typename BASE = Epiface,
00281 bool = cxx::is_polymorphic<BASE>::value>
00282 struct Irqep_t : Epiface_t0<void, BASE>
00283 {
00284 l4_msgtag_t dispatch(l4_msgtag_t, unsigned, l4_utcb_t *) final
00285 {
00286 static_cast<Derived*>(this)->handle_irq();
00287 return l4_msgtag(-L4_ENOREPLY, 0, 0, 0);
00288 }
00289 };

```

```

00288 }
00289
00294 Cap<L4::Irq> obj_cap() const
00295 { return L4::cap_cast<L4::Irq>(BASE::obj_cap()); }
00296 };
00297
00298 template<typename Derived, typename BASE>
00299 struct Irqep_t<Derived, BASE, false> : Epiface_t0<void, BASE>
00300 {
00301 l4_msgtag_t dispatch(l4_msgtag_t, unsigned, l4_utcb_t *)
00302 {
00303 static_cast<Derived*>(this)->handle_irq();
00304 return l4_msgtag(~L4_ENOREPLY, 0, 0, 0);
00305 }
00306
00311 Cap<L4::Irq> obj_cap() const
00312 { return L4::cap_cast<L4::Irq>(BASE::obj_cap()); }
00313 };
00314
00322 class Registry_iface
00323 {
00324 public:
00325 virtual ~Registry_iface() = 0;
00326
00339 virtual L4::Cap<void>
00340 register_obj(L4::Epiface *o, char const *service) = 0;
00341
00356 virtual L4::Cap<void>
00357 register_obj(L4::Epiface *o) = 0;
00358
00372 virtual L4::Cap<L4::Irq> register_irq_obj(L4::Epiface *o) = 0;
00373
00386 virtual L4::Cap<L4::Rcv_endpoint>
00387 register_obj(L4::Epiface *o, L4::Cap<L4::Rcv_endpoint> ep) = 0;
00388
00401 virtual void
00402 unregister_obj(L4::Epiface *o, bool unmap = true) = 0;
00403 };
00404
00405 inline Registry_iface::~Registry_iface() {}
00406
00407 namespace Ipc {
00408 namespace Detail {
00409 using namespace L4::Typeid;
00410
00411 template<typename IFACE>
00412 struct Meta_svr
00413 {
00414 {
00415 long op_num_interfaces(L4::Meta::Rights)
00416 { return 1; }
00417
00418 long op_interface(L4::Meta::Rights, l4_umword_t ifx, long &proto, L4::Ipc::String<char> &name)
00419 {
00420 if (ifx > 0)
00421 return -L4_ERANGE;
00422 proto = L4::kobject_typeid<IFACE>()->proto();
00423 if (auto *n = L4::kobject_typeid<IFACE>()->name())
00424 name.copy_in(n);
00425
00426 return 0;
00427 }
00428
00429 long op_supports(L4::Meta::Rights, l4_mword_t proto)
00430 { return L4::kobject_typeid<IFACE>()->has_proto(proto); }
00431 };
00432
00433 template<typename IFACE, typename LIST>
00434 struct _Dispatch;
00435
00436 // No match dispatcher found
00437 template<typename IFACE>
00438 struct _Dispatch<IFACE, Iface_list_end>
00439 {
00440 template< typename THIS, typename A1, typename A2 >
00441 static l4_msgtag_t f(THIS *, l4_msgtag_t, A1, A2 &)
00442 { return l4_msgtag(~L4_EBADPROTO, 0, 0, 0); }
00443 };
00444
00445 // call matching p_dispatch() function
00446 template<typename IFACE, typename I, typename LIST >
00447 struct _Dispatch<IFACE, Iface_list<I, LIST> >
00448 {
00449 // special handling for the meta protocol, to avoid 'using' murx
00450 template< typename THIS >
00451 static l4_msgtag_t _f(THIS *, l4_msgtag_t tag, unsigned r,
00452 l4_utcb_t *utcb, True::type)

```

```

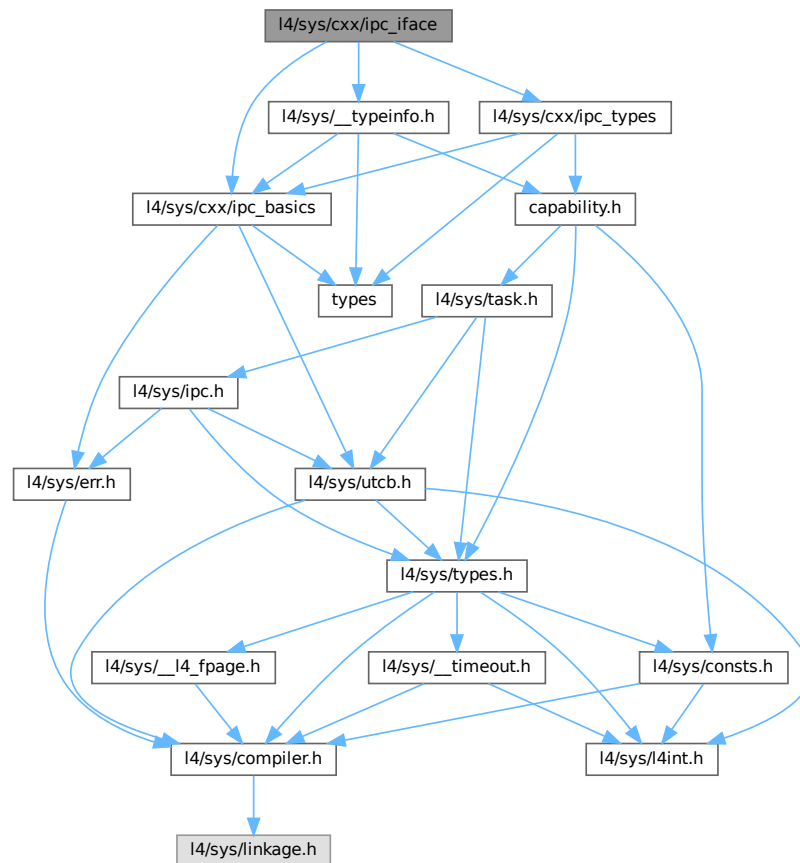
00453 {
00454 using L4::IpC::Msg::dispatch_call;
00455 typedef L4::Meta::Rpc Meta;
00456 typedef Meta_svr<IFACE> Msvr;
00457 return dispatch_call<Meta>(static_cast<Msvr*>(nullptr), utcb, tag, r);
00458 }
00459
00460 // normal dispatch to the op_<func> methods of \a self.
00461 template< typename THIS >
00462 static l4_msgtag_t _f(THIS *self, l4_msgtag_t t, unsigned r,
00463 l4_utcb_t *utcb, False::type)
00464 {
00465 using L4::IpC::Msg::dispatch_call;
00466 return dispatch_call<typename I::iface_type::Rpc>(self, utcb, t, r);
00467 }
00468
00469 // dispatch function with switch for meta protocol
00470 template< typename THIS >
00471 static l4_msgtag_t f(THIS *self, l4_msgtag_t tag, unsigned r,
00472 l4_utcb_t *utcb)
00473 {
00474 if (I::Proto == tag.label())
00475 return _f(self, tag, r, utcb,
00476 Bool<I::Proto == static_cast<long>(L4_PROTO_META)>());
00477
00478 return _Dispatch<IFACE, typename LIST::type>::f(self, tag, r, utcb);
00479 }
00480 };
00481
00482 template<typename IFACE>
00483 struct Dispatch :
00484 _Dispatch<IFACE, typename L4::Kobject_typeid<IFACE>::Iface_list::type>
00485 {};
00486
00487 } // namespace Detail
00488
00489 template<typename EPIFACE>
00490 struct Dispatch : Detail::Dispatch<typename EPIFACE::Interface>
00491 {};
00492
00493 } // namespace IpC
00494
00501 template<typename Derived, typename IFACE, typename BASE = L4::Epiface,
00502 bool = cxx::is_polymorphic<BASE>::value>
00503 struct Epiface_t : Epiface_t0<IFACE, BASE>
00504 {
00505 l4_msgtag_t
00506 dispatch(l4_msgtag_t tag, unsigned rights, l4_utcb_t *utcb) final
00507 {
00508 typedef IpC::Dispatch<Derived> Dispatch;
00509 return Dispatch::f(static_cast<Derived*>(this), tag, rights, utcb);
00510 }
00511 };
00512
00513 template<typename Derived, typename IFACE, typename BASE>
00514 struct Epiface_t<Derived, IFACE, BASE, false> : Epiface_t0<IFACE, BASE>
00515 {
00516 l4_msgtag_t
00517 dispatch(l4_msgtag_t tag, unsigned rights, l4_utcb_t *utcb)
00518 {
00519 typedef IpC::Dispatch<Derived> Dispatch;
00520 return Dispatch::f(static_cast<Derived*>(this), tag, rights, utcb);
00521 }
00522 };
00523
00529 class Basic_registry
00530 {
00531 public:
00532 typedef Epiface Value;
00533 static Value *find(l4_umword_t label)
00534 { return reinterpret_cast<Value*>(label & ~3UL); }
00540
00554 static l4_msgtag_t dispatch(l4_msgtag_t tag, l4_umword_t label,
00555 l4_utcb_t *utcb)
00556 {
00557 return find(label)->dispatch(tag, label, utcb);
00558 }
00559 };
00560
00561
00562 } // namespace L4

```

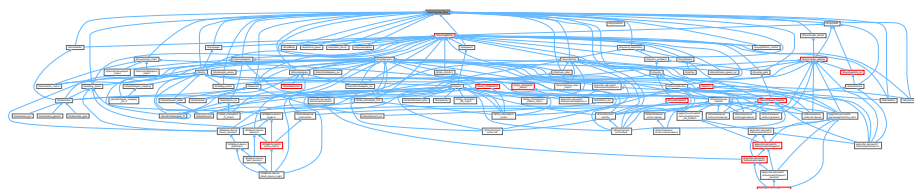
## 17.489 l4/sys/cxx/ipc\_iface File Reference

Interface Definition Language.

```
#include <l4/sys/cxx/ipc_basics>
#include <l4/sys/cxx/ipc_types>
#include <l4/sys/__typeinfo.h>
Include dependency graph for ipc_iface:
```



This graph shows which files directly or indirectly include this file:



### Data Structures

- struct [L4::ipc::Call](#)

- RPC attribute for a standard RPC call.*
  - struct [L4::ipc::Call\\_zero\\_send\\_timeout](#)
*RPC attribute for an RPC call, with zero send timeout.*
  - struct [L4::ipc::Call\\_t< RIGHTS >](#)
*RPC attribute for an RPC call with required rights.*
  - struct [L4::ipc::Send\\_only](#)
*RPC attribute for a send-only RPC.*

## Namespaces

- namespace [L4](#)
  - [L4](#) low-level kernel interface.
- namespace [L4::ipc](#)
  - IPC related functionality.
- namespace [L4::ipc::Msg](#)
  - IPC Message related functionality.

## Macros

- #define [L4\\_INLINE\\_RPC\\_NF](#)(res, name, args...)
  - Define an inline RPC call type (the type only, no callable).
- #define [L4\\_INLINE\\_RPC\\_NF\\_OP](#)(op, res, name, args...)
  - Define an inline RPC call type with specific opcode (the type only, no callable).
- #define [L4\\_INLINE\\_RPC](#)(res, name, args, attr...)
  - Define an inline RPC call (type and callable).
- #define [L4\\_INLINE\\_RPC\\_OP](#)(op, res, name, args, attr...)
  - Define an inline RPC call with specific opcode (type and callable).
- #define [L4\\_RPC\\_NF](#)(res, name, args...)
  - Define an RPC call type (the type only, no callable).
- #define [L4\\_RPC\\_NF\\_OP](#)(op, res, name, args...)
  - Define an RPC call type with specific opcode (the type only, no callable).
- #define [L4\\_RPC](#)(res, name, args, attr...)
  - Define an RPC call (type and callable).
- #define [L4\\_RPC\\_OP](#)(op, res, name, args, attr...)
  - Define an RPC call with specific opcode (type and callable).

## 17.489.1 Detailed Description

Interface Definition Language.

See also

[L4\\_RPC](#), [L4\\_INLINE\\_RPC](#), [L4::ipc::Call](#) [L4::ipc::Send\\_only](#), [L4::ipc::Msg::Rpc\\_call](#), [L4::ipc::Msg::Rpc\\_call](#)  
[inline\\_call](#)

Definition in file [ipc\\_iface](#).

## 17.489.2 Macro Definition Documentation

### 17.489.2.1 L4\_INLINE\_RPC

```
#define L4_INLINE_RPC(
 res,
 name,
 args,
 attr...)
```

#### Value:

`res name args`

Define an inline RPC call (type and callable).

#### Parameters

|             |                                                                                                  |
|-------------|--------------------------------------------------------------------------------------------------|
| <i>res</i>  | The result type of the RPC call                                                                  |
| <i>name</i> | The name of the function ( <code>name_t</code> is used for the type.)                            |
| <i>args</i> | The argument list of the RPC function.                                                           |
| <i>attr</i> | Optional RPC attributes ( <a href="#">L4::lpc::Call</a> , <a href="#">L4::lpc::Call_t</a> etc.). |

#### Examples

[examples/clntsrv/src/shared.h](#).

Definition at line 482 of file [ipc\\_iface](#).

Referenced by [L4Re::Mem\\_alloc::info\(\)](#).

### 17.489.2.2 L4\_INLINE\_RPC\_NF

```
#define L4_INLINE_RPC_NF(
 res,
 name,
 args...)
```

#### Value:

```
struct name##_t : L4::Ipc::Msg::Rpc_inline_call<name##_t, Class, res args> \
{ \
 typedef L4::Ipc::Msg::Rpc_inline_call<name##_t, Class, res args> type; \
 L4_INLINE_RPC_SRV_FORWARD(name); \
}
```

Define an inline RPC call type (the type only, no callable).

#### Parameters

|             |                                                                                                                                    |
|-------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>res</i>  | The result type of the RPC call                                                                                                    |
| <i>name</i> | The name of the function ( <code>name_t</code> is used for the type.)                                                              |
| <i>args</i> | The argument list of the RPC function, and RPC attributes ( <a href="#">L4::lpc::Call</a> , <a href="#">L4::lpc::Call_t</a> etc.). |

Stubs generated by this macro can be used explicitly in custom wrapper methods that need to use the underlying RPC code and provide some higher level abstraction, for example with default arguments or extra argument conversion.

Definition at line 453 of file [ipc\\_iface](#).



**17.489.2.3 L4\_INLINE\_RPC\_NF\_OP**

```
#define L4_INLINE_RPC_NF_OP (
 op,
 res,
 name,
 args...)

```

**Value:**

```
struct name##_t : L4::Ipc::Msg::Rpc_inline_call<name##_t, Class, res args> \
{ \
 typedef L4::Ipc::Msg::Rpc_inline_call<name##_t, Class, res args> type; \
 enum { Opcode = (op) }; \
 L4_INLINE_RPC_SRV_FORWARD(name); \
}

```

Define an inline RPC call type with specific opcode (the type only, no callable).

**Parameters**

|             |                                                                                                                                    |
|-------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>op</i>   | The opcode number for this function                                                                                                |
| <i>res</i>  | The result type of the RPC call                                                                                                    |
| <i>name</i> | The name of the function ( <i>name_t</i> is used for the type.)                                                                    |
| <i>args</i> | The argument list of the RPC function, and RPC attributes ( <a href="#">L4::Ipc::Call</a> , <a href="#">L4::Ipc::Call_t</a> etc.). |

Stubs generated by this macro can be used explicitly in custom wrapper methods that need to use the underlying RPC code and provide some higher level abstraction, for example with default arguments or extra argument conversion.

Definition at line 466 of file [ipc\\_iface](#).

**17.489.2.4 L4\_INLINE\_RPC\_OP**

```
#define L4_INLINE_RPC_OP (
 op,
 res,
 name,
 args,
 attr...)

```

**Value:**

```
res name args
```

Define an inline RPC call with specific opcode (type and callable).

**Parameters**

|             |                                                                                                  |
|-------------|--------------------------------------------------------------------------------------------------|
| <i>op</i>   | The opcode number for this function                                                              |
| <i>res</i>  | The result type of the RPC call                                                                  |
| <i>name</i> | The name of the function ( <i>name_t</i> is used for the type.)                                  |
| <i>args</i> | The argument list of the RPC function.                                                           |
| <i>attr</i> | Optional RPC attributes ( <a href="#">L4::Ipc::Call</a> , <a href="#">L4::Ipc::Call_t</a> etc.). |

Definition at line 497 of file [ipc\\_iface](#).

### 17.489.2.5 L4\_RPC

```
#define L4_RPC(
 res,
 name,
 args,
 attr...)

```

#### Value:

res name args

Define an RPC call (type and callable).

#### Parameters

|             |                                                                                                  |
|-------------|--------------------------------------------------------------------------------------------------|
| <i>res</i>  | The result type of the RPC call                                                                  |
| <i>name</i> | The name of the function ( <code>name_t</code> is used for the type.)                            |
| <i>args</i> | The argument list of the RPC function.                                                           |
| <i>attr</i> | Optional RPC attributes ( <a href="#">L4::Ipc::Call</a> , <a href="#">L4::Ipc::Call_t</a> etc.). |

Definition at line 541 of file [ipc\\_iface](#).

Referenced by [L4Re::Dataspace::info\(\)](#).

### 17.489.2.6 L4\_RPC\_NF

```
#define L4_RPC_NF(
 res,
 name,
 args...)

```

#### Value:

```
struct name##_t : L4::Ipc::Msg::Rpc_call<name##_t, Class, res args>
{
 typedef L4::Ipc::Msg::Rpc_call<name##_t, Class, res args> type;
 L4_INLINE_RPC_SRV_FORWARD(name);
}

```

Define an RPC call type (the type only, no callable).

#### Parameters

|             |                                                                                                                                    |
|-------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>res</i>  | The result type of the RPC call                                                                                                    |
| <i>name</i> | The name of the function ( <code>name_t</code> is used for the type.)                                                              |
| <i>args</i> | The argument list of the RPC function, and RPC attributes ( <a href="#">L4::Ipc::Call</a> , <a href="#">L4::Ipc::Call_t</a> etc.). |

Definition at line 510 of file [ipc\\_iface](#).

Referenced by [L4Re::Dataspace::info\(\)](#).

**17.489.2.7 L4\_RPC\_NF\_OP**

```
#define L4_RPC_NF_OP (
 op,
 res,
 name,
 args...)

```

**Value:**

```
struct name##_t : L4::Ipc::Msg::Rpc_call<name##_t, Class, res args>
{
 typedef L4::Ipc::Msg::Rpc_call<name##_t, Class, res args> type;
 enum { Opcode = (op) };
 L4_INLINE_RPC_SRV_FORWARD(name);
}

```

```
\\
\\
\\
\\

```

Define an RPC call type with specific opcode (the type only, no callable).

**Parameters**

|             |                                                                                                                                    |
|-------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>op</i>   | The opcode number for this function                                                                                                |
| <i>res</i>  | The result type of the RPC call                                                                                                    |
| <i>name</i> | The name of the function ( <i>name_t</i> is used for the type.)                                                                    |
| <i>args</i> | The argument list of the RPC function, and RPC attributes ( <a href="#">L4::Ipc::Call</a> , <a href="#">L4::Ipc::Call_t</a> etc.). |

Definition at line 525 of file [ipc\\_iface](#).

**17.489.2.8 L4\_RPC\_OP**

```
#define L4_RPC_OP (
 op,
 res,
 name,
 args,
 attr...)

```

**Value:**

```
res name args
```

Define an RPC call with specific opcode (type and callable).

**Parameters**

|             |                                                                                                  |
|-------------|--------------------------------------------------------------------------------------------------|
| <i>op</i>   | The opcode number for this function                                                              |
| <i>res</i>  | The result type of the RPC call                                                                  |
| <i>name</i> | The name of the function ( <i>name_t</i> is used for the type.)                                  |
| <i>args</i> | The argument list of the RPC function.                                                           |
| <i>attr</i> | Optional RPC attributes ( <a href="#">L4::Ipc::Call</a> , <a href="#">L4::Ipc::Call_t</a> etc.). |

Definition at line 556 of file [ipc\\_iface](#).

## 17.490 ipc\_iface

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2014 Alexander Warg <alexander.warg@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008 #pragma GCC system_header
00009
00010 #include <l4/sys/cxx/ipc_basics>
00011 #include <l4/sys/cxx/ipc_types>
00012 #include <l4/sys/__typeinfo.h>
00013
00022
00219
00220 // TODO: add some more documentation
00221 namespace L4 { namespace Ipc {
00222
00239 struct L4_EXPORT Call
00240 {
00241 enum { Is_call = true };
00242 enum { Rights = 0 };
00243 static l4_timeout_t timeout() { return L4_IPC_NEVER; }
00244 };
00245
00249 struct L4_EXPORT Call_zero_send_timeout : Call
00250 {
00251 static l4_timeout_t timeout() { return L4_IPC_SEND_TIMEOUT_0; }
00252 };
00253
00269 template<unsigned RIGHTS>
00270 struct L4_EXPORT Call_t : Call
00271 {
00272 enum { Rights = RIGHTS };
00273 };
00274
00287 struct L4_EXPORT Send_only
00288 {
00289 enum { Is_call = false };
00290 enum { Rights = 0 };
00291 static l4_timeout_t timeout() { return L4_IPC_NEVER; }
00292 };
00293
00294 namespace Msg {
00295
00306 template<typename OP, typename CLASS, typename SIG, typename FLAGS = Call>
00307 struct L4_EXPORT Rpc_inline_call;
00308
00313 template<typename OP, typename CLASS, typename FLAGS, typename R,
00314 typename ...ARGS>
00315 struct L4_EXPORT Rpc_inline_call<OP, CLASS, R (ARGS...), FLAGS>
00316 {
00317 template<typename T> struct Result { typedef T result_type; };
00318 enum
00319 {
00320 Return_tag = L4::Types::Same<R, l4_msgtag_t>::value
00321 };
00322
00324 typedef Rpc_inline_call type;
00326 typedef OP op_type;
00328 typedef CLASS class_type;
00330 typedef typename Result<R>::result_type result_type;
00332 typedef R ipc_type (ARGS...);
00334 typedef result_type func_type (typename _Elem<ARGS>::arg_type...);
00335
00337 typedef FLAGS flags_type;
00338
00339 template<typename RES>
00340 static typename L4::Types::Enable_if< Return_tag, RES >::type
00341 return_err(long err) noexcept { return l4_msgtag(err, 0, 0, 0); }
00342
00343 template<typename RES>
00344 static typename L4::Types::Enable_if< Return_tag, RES >::type
00345 return_ipc_err(l4_msgtag_t tag, l4_utcb_t const *) noexcept { return tag; }
00346
00347 template<typename RES>
00348 static typename L4::Types::Enable_if< Return_tag, RES >::type
00349 return_code(l4_msgtag_t tag) noexcept { return tag; }
00350
00351 template<typename RES>
00352 static typename L4::Types::Enable_if< !Return_tag, RES >::type
00353 return_err(long err) noexcept { return err; }

```

```

00354
00355 template<typename RES>
00356 static typename L4::Types::Enable_if< !Return_tag, RES >::type
00357 return_ipc_err(l4_msgtag_t, l4_utcb_t *utcb) noexcept
00358 { return l4_ipc_to_errno(l4_ipc_error_code(utcb)); }
00359
00360 template<typename RES>
00361 static typename L4::Types::Enable_if< !Return_tag, RES >::type
00362 return_code(l4_msgtag_t tag) noexcept { return tag.label(); }
00363
00364 static R call(L4::Cap<class_type> cap,
00365 typename _Elem<ARGS>::arg_type ...a,
00366 l4_utcb_t *utcb = l4_utcb()) noexcept;
00367 };
00368
00373 template<typename OP, typename CLASS, typename SIG, typename FLAGS = Call>
00374 struct L4_EXPORT Rpc_call;
00375
00383 template<typename IPC, typename SIG> struct _Call;
00384
00386 template<typename IPC, typename R, typename ...ARGS>
00387 struct _Call<IPC, R (ARGS...)>
00388 {
00389 public:
00390 typedef typename IPC::class_type class_type;
00391 typedef typename IPC::result_type result_type;
00392
00393 private:
00394 L4::Cap<class_type> cap() const noexcept
00395 {
00396 return L4::Cap<class_type>(reinterpret_cast<l4_cap_idx_t>(this)
00397 & L4_CAP_MASK);
00398 }
00399
00400 public:
00402 result_type operator () (ARGS ...a, l4_utcb_t *utcb = l4_utcb()) const noexcept
00403 { return IPC::call(cap(), a..., utcb); }
00404 };
00405
00412 template<typename IPC> struct Call : _Call<IPC, typename IPC::func_type> {};
00413
00418 template<typename OP,
00419 typename CLASS,
00420 typename FLAGS,
00421 typename R,
00422 typename ...ARGS>
00423 struct L4_EXPORT Rpc_call<OP, CLASS, R (ARGS...), FLAGS> :
00424 Rpc_inline_call<OP, CLASS, R (ARGS...), FLAGS>
00425 {
00426 static R call(L4::Cap<CLASS> cap,
00427 typename _Elem<ARGS>::arg_type ...a,
00428 l4_utcb_t *utcb = l4_utcb()) noexcept;
00429 };
00430
00431 #define L4_INLINE_RPC_SRV_FORWARD(name)
00432 template<typename OBJ> struct fwd
00433 {
00434 OBJ *o;
00435 fwd(OBJ *o) noexcept : o(o) {}
00436 template<typename ...ARGS> long call(ARGS ...a) noexcept(noexcept(o->op_##name(a...)))
00437 { return o->op_##name(a...); }
00438 }
00439
00440
00453 #define L4_INLINE_RPC_NF(res, name, args...)
00454 struct name##_t : L4::IpC::Msg::Rpc_inline_call<name##_t, Class, res args>
00455 {
00456 typedef L4::IpC::Msg::Rpc_inline_call<name##_t, Class, res args> type;
00457 L4_INLINE_RPC_SRV_FORWARD(name);
00458 }
00459
00466 #define L4_INLINE_RPC_NF_OP(op, res, name, args...)
00467 struct name##_t : L4::IpC::Msg::Rpc_inline_call<name##_t, Class, res args>
00468 {
00469 typedef L4::IpC::Msg::Rpc_inline_call<name##_t, Class, res args> type;
00470 enum { Opcode = (op) };
00471 L4_INLINE_RPC_SRV_FORWARD(name);
00472 }
00473
00474 #ifndef DOXYGEN
00482 #define L4_INLINE_RPC(res, name, args, attr...) res name args
00483 #else
00484 #define L4_INLINE_RPC(res, name, args...)
00485 L4_INLINE_RPC_NF(res, name, args); L4::IpC::Msg::Call<name##_t> name
00486 #endif
00487
00488 #ifndef DOXYGEN

```

```

00497 #define L4_INLINE_RPC_OP(op, res, name, args, attr...) res name args
00498 #else
00499 #define L4_INLINE_RPC_OP(op, res, name, args...) \
00500 L4_INLINE_RPC_NF_OP(op, res, name, args); L4::Rpc::Msg::Call<name##_t> name
00501 #endif
00502
00510 #define L4_RPC_NF(res, name, args...) \
00511 struct name##_t : L4::Rpc::Msg::Rpc_call<name##_t, Class, res args> \
00512 { \
00513 typedef L4::Rpc::Msg::Rpc_call<name##_t, Class, res args> type; \
00514 L4_INLINE_RPC_SRV_FORWARD(name); \
00515 }
00516
00525 #define L4_RPC_NF_OP(op, res, name, args...) \
00526 struct name##_t : L4::Rpc::Msg::Rpc_call<name##_t, Class, res args> \
00527 { \
00528 typedef L4::Rpc::Msg::Rpc_call<name##_t, Class, res args> type; \
00529 enum { Opcode = (op) }; \
00530 L4_INLINE_RPC_SRV_FORWARD(name); \
00531 }
00532
00533 #ifdef DOXYGEN
00541 #define L4_RPC(res, name, args, attr...) res name args
00542 #else
00543 #define L4_RPC(res, name, args...) \
00544 L4_RPC_NF(res, name, args); L4::Rpc::Msg::Call<name##_t> name
00545 #endif
00546
00547 #ifdef DOXYGEN
00556 #define L4_RPC_OP(op, res, name, args, attr...) res name args
00557 #else
00558 #define L4_RPC_OP(op, res, name, args...) \
00559 L4_RPC_NF_OP(op, res, name, args); L4::Rpc::Msg::Call<name##_t> name
00560 #endif
00561
00562 namespace Detail {
00563
00572 template<typename ...ARGS>
00573 struct Buf
00574 {
00575 public:
00576 template<typename DIR>
00577 static constexpr int write(char *, int offset, int) noexcept
00578 { return offset; }
00579
00580 template<typename DIR>
00581 static constexpr int read(char *, int offset, int, long) noexcept
00582 { return offset; }
00583
00584 typedef void Base;
00585 };
00586
00587 template<typename A, typename ...M>
00588 struct Buf<A, M...> : Buf<M...>
00589 {
00590 typedef Buf<M...> Base;
00591
00592 typedef Clnt_xmit<A> xmit;
00593 typedef typename _Elem<A>::arg_type arg_type;
00594 typedef Detail::_Plain<arg_type> plain;
00595
00596 template<typename DIR>
00597 static int
00598 write(char *base, int offset, int limit,
00599 arg_type a, typename _Elem<M>::arg_type ...m) noexcept
00600 {
00601 offset = xmit::to_msg(base, offset, limit, plain::deref(a),
00602 typename DIR::dir(), typename DIR::cls());
00603 return Base::template write<DIR>(base, offset, limit, m...);
00604 }
00605
00606 template<typename DIR>
00607 static int
00608 read(char *base, int offset, int limit, long ret,
00609 arg_type a, typename _Elem<M>::arg_type ...m) noexcept
00610 {
00611 int r = xmit::from_msg(base, offset, limit, ret, plain::deref(a),
00612 typename DIR::dir(), typename DIR::cls());
00613 if (L4_LIKELY(r >= 0))
00614 return Base::template read<DIR>(base, r, limit, ret, m...);
00615
00616 if (_Elem<A>::Is_optional)
00617 return Base::template read<DIR>(base, offset, limit, ret, m...);
00618
00619 return r;
00620 }
00621 }

```

```

00621 };
00622
00623 template <typename ...ARGS> struct _Part
00624 {
00625 typedef Buf<ARGS...> Data;
00626
00627 template<typename DIR>
00628 static int write(void *b, int offset, int limit,
00629 typename _Elem<ARGS>::arg_type ...m) noexcept
00630 {
00631 char *buf = static_cast<char *>(b);
00632 int r = Data::template write<DIR>(buf, offset, limit, m...);
00633 if (L4_LIKELY(r >= offset))
00634 return r - offset;
00635 return r;
00636 }
00637
00638 template<typename DIR>
00639 static int read(void *b, int offset, int limit, long ret,
00640 typename _Elem<ARGS>::arg_type ...m) noexcept
00641 {
00642 char *buf = static_cast<char *>(b);
00643 int r = Data::template read<DIR>(buf, offset, limit, ret, m...);
00644 if (L4_LIKELY(r >= offset))
00645 return r - offset;
00646 return r;
00647 }
00648 };
00649
00650 template<typename IPC_TYPE, typename OPCODE = void>
00651 struct Part;
00652
00653 // The version without an op-code
00654 template<typename R, typename ...ARGS>
00655 struct Part<R (ARGS...), void> : _Part<ARGS...>
00656 {
00657 typedef Buf<ARGS...> Data;
00658
00659 // write arguments, skipping the dummy opcode
00660 template<typename DIR>
00661 static int write_op(void *b, int offset, int limit,
00662 int /*placeholder for op*/,
00663 typename _Elem<ARGS>::arg_type ...m) noexcept
00664 {
00665 char *buf = static_cast<char *>(b);
00666 int r = Data::template write<DIR>(buf, offset, limit, m...);
00667 if (L4_LIKELY(r >= offset))
00668 return r - offset;
00669 return r;
00670 }
00671 };
00672
00673 // Message part with additional opcode
00674 template<typename OPCODE, typename R, typename ...ARGS>
00675 struct Part<R (ARGS...), OPCODE> : _Part<ARGS...>
00676 {
00677 typedef OPCODE opcode_type;
00678 typedef Buf<opcode_type, ARGS...> Data;
00679
00680 // write arguments, including the opcode
00681 template<typename DIR>
00682 static int write_op(void *b, int offset, int limit,
00683 opcode_type op, typename _Elem<ARGS>::arg_type ...m) noexcept
00684 {
00685 char *buf = static_cast<char *>(b);
00686 int r = Data::template write<DIR>(buf, offset, limit, op, m...);
00687 if (L4_LIKELY(r >= offset))
00688 return r - offset;
00689 return r;
00690 }
00691 };
00692
00693 } // namespace Detail
00694
00695 //-----
00696 // Implementation of the RPC call
00697 // TODO: Add support for timeout via special RPC argument
00698 // TODO: Add support for passing the UTCB pointer as argument
00699 //
00700 template<typename OP, typename CLASS, typename FLAGS, typename R,
00701 typename ...ARGS>
00702 inline R
00703 Rpc_inline_call<OP, CLASS, R (ARGS...), FLAGS>::
00704 call(L4::Cap<CLASS> cap,
00705 typename _Elem<ARGS>::arg_type ...a,
00706 l4_utcb_t *utcb) noexcept

```

```

00717 {
00718 using namespace Ipc::Msg;
00719
00720 typedef typename Kobject_typeid<CLASS>::Iface::Rpc Rpcs;
00721 typedef typename Rpcs::template Rpc<OP> Opt;
00722 typedef Detail::Part<ipc_type, typename Rpcs::opcode_type> Args;
00723
00724 l4_msg_regs_t *mrs = l4_utcb_mr_u(utcb);
00725
00726 // handle in-data part of the arguments
00727 int send_bytes =
00728 Args::template write_op<Do_in_data>(mrs->mr, 0, Mr_bytes,
00729 Opt::Opcode, a...);
00730
00731 if (L4_UNLIKELY(send_bytes < 0))
00732 return return_err<R>(send_bytes);
00733
00734 send_bytes = align_to<l4_umword_t>(send_bytes);
00735 int const send_words = send_bytes / Word_bytes;
00736 // write the in-items part of the message if there is one
00737 int item_bytes =
00738 Args::template write<Do_in_items>(&mrs->mr[send_words], 0,
00739 Mr_bytes - send_bytes, a...);
00740
00741 if (L4_UNLIKELY(item_bytes < 0))
00742 return return_err<R>(item_bytes);
00743
00744 int send_items = item_bytes / Item_bytes;
00745
00746 {
00747 // setup the receive buffers for the RPC call
00748 l4_buf_regs_t *brs = l4_utcb_br_u(utcb);
00749 // XXX: we currently support only one type of receive buffers per call
00750 brs->bdr = 0; // we always start at br[0]
00751
00752 // the limit leaves us at least one register for the zero terminator
00753 // add the buffers given as arguments to the buffer registers
00754 int bytes =
00755 Args::template write<Do_rcv_buffers>(brs->br, 0, Br_bytes - Word_bytes,
00756 a...);
00757
00758 if (L4_UNLIKELY(bytes < 0))
00759 return return_err<R>(bytes);
00760
00761 brs->br[bytes / Word_bytes] = 0;
00762 }
00763
00764 // here we do the actual IPC -----
00765 l4_msgtag_t t;
00766 t = l4_msgtag(CLASS::Protocol, send_words, send_items, 0);
00767 // do the call (Q: do we need support for timeouts?)
00768 if (flags_type::Is_call)
00769 t = l4_ipc_call(cap.cap(), utcb, t, flags_type::timeout());
00770 else
00771 {
00772 t = l4_ipc_send(cap.cap(), utcb, t, flags_type::timeout());
00773 if (L4_UNLIKELY(t.has_error()))
00774 return return_ipc_err<R>(t, utcb);
00775
00776 return return_code<R>(l4_msgtag(0, 0, 0, t.flags()));
00777 }
00778
00779 // unmarshalling starts here -----
00780
00781 // bail out early in the case of an IPC error
00782 if (L4_UNLIKELY(t.has_error()))
00783 return return_ipc_err<R>(t, utcb);
00784
00785 // take the label as return value
00786 long r = t.label();
00787
00788 // bail out on negative error codes too
00789 if (L4_UNLIKELY(r < 0))
00790 return return_err<R>(r);
00791
00792 int const rcv_bytes = t.words() * Word_bytes;
00793
00794 // read the static out-data values to the arguments
00795 int err = Args::template read<Do_out_data>(mrs->mr, 0, rcv_bytes, r, a...);
00796
00797 int const item_limit = t.items() * Item_bytes;
00798
00799 if (L4_UNLIKELY(err < 0 || item_limit > Mr_bytes))
00800 return return_err<R>(-L4_EMMSGTOOSHORT);
00801
00802 // read the static out-items to the arguments

```



```

00804 err = Args::template read<Do_out_items>(&mrs->mr[t.words()], 0, item_limit,
00805 r, a...);
00806
00807 if (L4_UNLIKELY(err < 0))
00808 return return_err<R>(-L4_EMSTOOSHORT);
00809
00810 return return_code<R>(t);
00811 }
00812
00813 } // namespace Msg
00814 } // namespace Ipc
00815 } // namespace L4

```

## 17.491 ipc\_legacy

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * Copyright (C) 2015, 2017, 2024 Kernkonzept GmbH.
00004 * Author(s): Alexander Warg <alexander.warg@kernkonzept.com>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include <l4/sys/cxx/ipc_epiface>
00011 #ifndef L4_RPC_DISABLE_LEGACY_DISPATCH
00012 #define L4_RPC_LEGACY_DISPATCH(IFACE)
00013 template<typename IOS>
00014 int dispatch(unsigned rights, IOS &ios)
00015 {
00016 typedef ::L4::Ipc::Detail::Dispatch<IFACE> Dispatch;
00017 l4_msgtag_t r = Dispatch::f(this, ios.tag(), rights, ios.utcb());
00018 ios.set_ipc_params(r);
00019 return r.label();
00020 }
00021
00022 template<typename IOS>
00023 int p_dispatch(IFACE *, unsigned rights, IOS &ios)
00024 {
00025 using ::L4::Ipc::Msg::dispatch_call;
00026 l4_msgtag_t r;
00027 r = dispatch_call<typename IFACE::Rpcs>(this, ios.utcb(),
00028 ios.tag(), rights);
00029 ios.set_ipc_params(r);
00030 return r.label();
00031 }
00032
00033 #define L4_RPC_LEGACY_USING(IFACE) \
00034 using IFACE::p_dispatch
00035
00036 #else
00037 #define L4_RPC_LEGACY_DISPATCH(IFACE)
00038 #define L4_RPC_LEGACY_USING(IFACE)
00039 #endif

```

## 17.492 ipc\_ret\_array

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2014 Alexander Warg <alexander.warg@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include "types"
00010 #include "ipc_basics"
00011
00012 namespace L4 { namespace Ipc L4_EXPORT {
00013
00014 // -----
00023 template<typename T> struct L4_EXPORT Ret_array
00024 {
00025 typedef T const **ptr_type;
00026
00027 T *value = nullptr;
00028 unsigned max = 0;
00029 Ret_array() {}

```

```

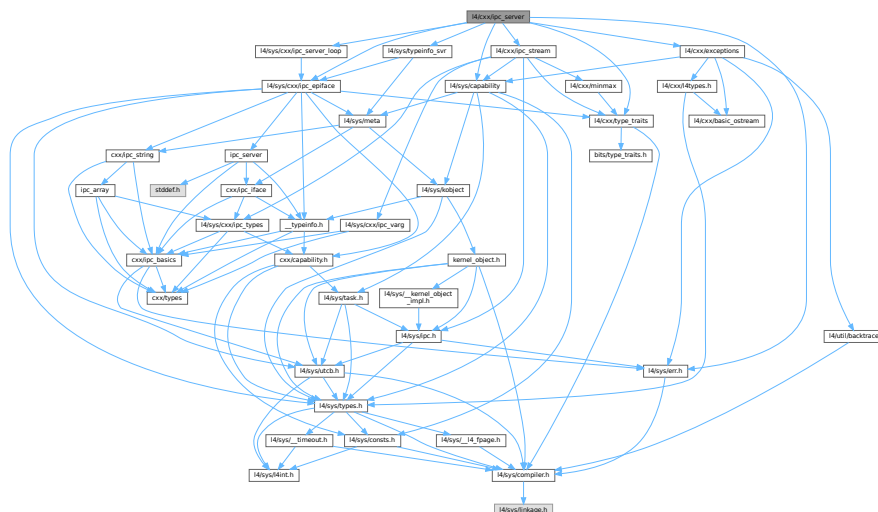
00030 Ret_array(T *v, unsigned max) : value(v), max(max) {}
00031 };
00032
00033 namespace Msg {
00034
00035 template<typename A> struct Elem< Ret_array<A> >
00036 {
00037 enum { Is_optional = false };
00038 typedef Ret_array<A> type;
00039 typedef typename type::ptr_type arg_type;
00040 typedef type svr_type;
00041 typedef type svr_arg_type;
00042 };
00043
00044 template<typename A>
00045 struct Is_valid_rpc_type<Ret_array<A> *> : L4::Types::False {};
00046 template<typename A>
00047 struct Is_valid_rpc_type<Ret_array<A> &> : L4::Types::False {};
00048 template<typename A>
00049 struct Is_valid_rpc_type<Ret_array<A> const &> : L4::Types::False {};
00050 template<typename A>
00051 struct Is_valid_rpc_type<Ret_array<A> const *> : L4::Types::False {};
00052
00053 template<typename A> struct Class< Ret_array<A> > : Class<A>::type {};
00054 template<typename A> struct Direction< Ret_array<A> > : Dir_out {};
00055
00056 template<typename A, typename CLASS>
00057 struct Clnt_val_ops<A const *, Dir_out, CLASS> : Clnt_noops<A const *>
00058 {
00059 using Clnt_noops<A const *>::from_msg;
00060 static int from_msg(char *msg, unsigned offset, unsigned limit, long ret,
00061 A const *arg, Dir_out, Cls_data)
00062 {
00063 offset = align_to<A>(offset);
00064 arg = reinterpret_cast<A const *>(msg + offset);
00065 if (L4_UNLIKELY(!check_size<A>(offset, limit, ret)))
00066 return -1;
00067 return offset + ret * sizeof(A);
00068 }
00069 };
00070
00071
00072 template<typename A, typename CLASS>
00073 struct Svr_val_ops<Ret_array<A>, Dir_out, CLASS> :
00074 Svr_noops<Ret_array<A> >
00075 {
00076 typedef Ret_array<A> ret_array;
00077 using Svr_noops<ret_array>::from_svr;
00078 static int from_svr(char *, unsigned offset, unsigned limit, long ret,
00079 ret_array const &, Dir_out, CLASS)
00080 {
00081 offset = align_to<A>(offset);
00082 if (L4_UNLIKELY(!check_size<A>(offset, limit, ret)))
00083 return -1;
00084 offset += sizeof(A) * ret;
00085 return offset;
00086 }
00087
00088 using Svr_noops<ret_array>::to_svr;
00089 static int to_svr(char *msg, unsigned offset, unsigned limit,
00090 ret_array &arg, Dir_out, CLASS)
00091 {
00092 // there can be actually no limit check here, as this
00093 // is variably sized output array
00094 // FIXME: we could somehow makesure that this is the last
00095 // output value...
00096 offset = align_to<A>(offset);
00097 arg = ret_array(reinterpret_cast<A*>(msg + offset),
00098 (limit - offset) / sizeof(A));
00099 // FIXME: we dont know the length of the array here so, cheat
00100 return offset;
00101 }
00102 };
00103 } // namespace Msg
00104
00105 }

```

## 17.493 I4/cxx/ipc\_server File Reference

IPC server loop.

Include dependency graph for ipc\_server:



- class `L4::Server_object`  
*Abstract server object to be used with `L4::Server` and `L4::Basic_registry`.*
- struct `L4::Server_object_t< IFACE, BASE >`  
*Base class (template) for server implementing server objects.*
- struct `L4::Server_object_x< Derived, IFACE, BASE >`  
*Helper class to implement `p_dispatch` based server objects.*
- struct `L4::Irq_handler_object`  
*`Server` object base class for handling IRQ messages.*

- namespace L4  
*L4 low-level kernel interface.*

Definition in file [ipc\\_server](#).

## 17.494 ipc\_server

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00009 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #pragma once
00015
00016 #include <l4/sys/capability>
00017 #include <l4/sys/typeinfo_svr>
00018 #include <l4/sys/err.h>
00019 #include <l4/cxx/ipc_stream>
00020 #include <l4/sys/cxx/ipc_epiface>
00021 #include <l4/sys/cxx/ipc_server_loop>
00022 #include <l4/cxx/type_traits>
00023 #include <l4/cxx/exceptions>
00024
00025 namespace L4 {
00026
00038 class Server_object : public Epiface
00039 {
00040 public:
00058 virtual int dispatch(unsigned long rights, Ipc::Iostream &ios) = 0;
00059
00060 l4_msgtag_t dispatch(l4_msgtag_t tag, unsigned rights, l4_utcb_t *utcb) override
00061 {
00062 L4::Ipc::Iostream ios(utcb);
00063 ios.tag() = tag;
00064 int r = dispatch(rights, ios);
00065 return ios.prepare_ipc(r);
00066 }
00067
00068 Cap<Kobject> obj_cap() const
00069 { return cap_cast<Kobject>(Epiface::obj_cap()); }
00070 };
00071
00079 template<typename IFACE, typename BASE = L4::Server_object>
00080 struct Server_object_t : BASE
00081 {
00083 typedef IFACE Interface;
00084
00086 typename BASE::Demand get_buffer_demand() const override
00087 { return typename L4::Kobject_typeid<IFACE>::Demand(); }
00088
00099 int dispatch_meta_request(L4::Ipc::Iostream &ios)
00100 { return L4::Util::handle_meta_request<IFACE>(ios); }
00101
00113 template<typename THIS>
00114 static int proto_dispatch(THIS *self, l4_umword_t rights, L4::Ipc::Iostream &ios)
00115 {
00116 l4_msgtag_t t;
00117 ios » t;
00118 return Kobject_typeid<IFACE>::proto_dispatch(self, t.label(), rights, ios);
00119 }
00120 };
00121
00142 template<typename Derived, typename IFACE, typename BASE = L4::Server_object>
00143 struct Server_object_x : Server_object_t<IFACE, BASE>
00144 {
00146 int dispatch(l4_umword_t r, L4::Ipc::Iostream &ios)
00147 {
00148 return Server_object_t<IFACE, BASE>::proto_dispatch(static_cast<Derived *>(this),
00149 r, ios);
00150 }
00151 };
00152
00161 struct Irq_handler_object : Server_object_t<Kobject> {};
00162
00163 }

```

## 17.495 ipc\_server

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*

```

```

00003 * (c) 2014 Alexander Warg <alexander.warg@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008 #pragma GCC system_header
00009
00010 #include <l4/sys/cxx/ipc_basics>
00011 #include <l4/sys/cxx/ipc_iface>
00012 #include <l4/sys/__typeinfo.h>
00013 #include <stddef.h>
00014
00015 namespace L4 {
00016 namespace Ipc {
00017 namespace Msg {
00018 namespace Detail {
00019
00020 template<typename T> struct Sizeof { enum { size = sizeof(T) }; };
00021 template<> struct Sizeof<void> { enum { size = 0 }; };
00022
00026 template<typename ...> struct Arg_pack
00027 {
00028 template<typename DIR>
00029 unsigned get(char *, unsigned offset, unsigned)
00030 { return offset; }
00031
00032 template<typename DIR>
00033 unsigned set(char *, unsigned offset, unsigned, long)
00034 { return offset; }
00035
00036 template<typename F, typename ...ARGS>
00037 long call(F f, ARGS ...args)
00038 { return f(args...); }
00039
00040 template<typename O, typename FUNC, typename ...ARGS>
00041 long obj_call(O *o, ARGS ...args)
00042 {
00043 typedef typename FUNC::template fwd<O> Fwd;
00044 return Fwd(o).template call<ARGS...>(args...);
00045 //return o->op_dispatch(args...);
00046 }
00047 };
00048
00052 template<typename T, typename SVR_TYPE, typename ...M>
00053 struct Svr_arg : Svr_xmit<T>, Arg_pack<M...>
00054 {
00055 typedef Arg_pack<M...> Base;
00056
00057 typedef SVR_TYPE svr_type;
00058 typedef typename _Elem<T>::svr_arg_type svr_arg_type;
00059
00060 svr_type v;
00061
00062 template<typename DIR>
00063 int get(char *msg, unsigned offset, unsigned limit)
00064 {
00065 typedef Svr_xmit<T> ct;
00066 int r = ct::to_svr(msg, offset, limit, this->v,
00067 typename DIR::dir(), typename DIR::cls());
00068 if (L4_LIKELY(r >= 0))
00069 return Base::template get<DIR>(msg, r, limit);
00070
00071 if (_Elem<T>::Is_optional)
00072 {
00073 v = svr_type();
00074 return Base::template get<DIR>(msg, offset, limit);
00075 }
00076 return r;
00077 }
00078
00079 template<typename DIR>
00080 int set(char *msg, unsigned offset, unsigned limit, long ret)
00081 {
00082 typedef Svr_xmit<T> ct;
00083 int r = ct::from_svr(msg, offset, limit, ret, this->v,
00084 typename DIR::dir(), typename DIR::cls());
00085 if (L4_UNLIKELY(r < 0))
00086 return r;
00087 return Base::template set<DIR>(msg, r, limit, ret);
00088 }
00089
00090 template<typename F, typename ...ARGS>
00091 long call(F f, ARGS ...args)
00092 {
00093 //As_arg<value_type> check;
00094 return Base::template
00095 call<F, ARGS..., svr_arg_type>(f, args..., this->v);

```

```

00096 }
00097
00098 template<typename O, typename FUNC, typename ...ARGS>
00099 long obj_call(O *o, ARGS ...args)
00100 {
00101 //As_arg<value_type> check;
00102 return Base::template
00103 obj_call<O,FUNC, ARGS..., svr_arg_type>(o, args..., this->v);
00104 }
00105 };
00106
00107 template<typename T, typename ...M>
00108 struct Svr_arg<T, void, M...> : Arg_pack<M...>
00109 {
00110 typedef Arg_pack<M...> Base;
00111
00112 template<typename DIR>
00113 int get(char *msg, unsigned offset, unsigned limit)
00114 { return Base::template get<DIR>(msg, offset, limit); }
00115
00116 template<typename DIR>
00117 int set(char *msg, unsigned offset, unsigned limit, long ret)
00118 { return Base::template set<DIR>(msg, offset, limit, ret); }
00119
00120 template<typename F, typename ...ARGS>
00121 long call(F f, ARGS ...args)
00122 {
00123 return Base::template call<F, ARGS...>(f, args...);
00124 }
00125
00126 template<typename O, typename FUNC, typename ...ARGS>
00127 long obj_call(O *o, ARGS ...args)
00128 {
00129 return Base::template obj_call<O, FUNC, ARGS...>(o, args...);
00130 }
00131 };
00132
00133 template<typename A, typename ...M>
00134 struct Arg_pack<A, M...> : Svr_arg<A, typename _Elem<A>::svr_type, M...>
00135 {};
00136
00137 } // namespace Detail
00138
00139 //-----
00144 template<typename IPC_TYPE> struct Svr_arg_pack;
00145
00146 template<typename R, typename ...ARGS>
00147 struct Svr_arg_pack<R (ARGS...)> : Detail::Arg_pack<ARGS...>
00148 {
00149 typedef Detail::Arg_pack<ARGS...> Base;
00150 template<typename DIR>
00151 int get(void *msg, unsigned offset, unsigned limit)
00152 {
00153 char *buf = static_cast<char *>(msg);
00154 return Base::template get<DIR>(buf, offset, limit);
00155 }
00156
00157 template<typename DIR>
00158 int set(void *msg, unsigned offset, unsigned limit, long ret)
00159 {
00160 char *buf = static_cast<char *>(msg);
00161 return Base::template set<DIR>(buf, offset, limit, ret);
00162 }
00163 };
00164
00168 template<typename IPC_TYPE, typename O, typename ...ARGS>
00169 static l4_msgtag_t
00170 handle_svr_obj_call(O *o, l4_utcb_t *utcb, l4_msgtag_t tag, ARGS ...args)
00171 {
00172 typedef Svr_arg_pack<typename IPC_TYPE::rpc::ipc_type> Pack;
00173 enum
00174 {
00175 Do_reply = IPC_TYPE::rpc::flags_type::Is_call,
00176 Short_err = Do_reply ? -L4_EMMSGTOOSHORT : -L4_ENOREPLY,
00177 };
00178
00179 // XXX: send a reply or just do not reply in case of a cheating client
00180 if (L4_UNLIKELY(tag.words() + tag.items() * Item_words > Mr_words))
00181 return l4_msgtag(Short_err, 0, 0, 0);
00182
00183 // our whole arguments data structure
00184 Pack pack;
00185 l4_msg_regs_t *mrs = l4_utcb_mr_u(utcb);
00186
00187 int in_pos = Detail::Sizeof<typename IPC_TYPE::opcode_type>::size;
00188
00189 unsigned const in_bytes = tag.words() * Word_bytes;

```

```

00190
00191 in_pos = pack.template get<Do_in_data>(&mrs->mr[0], in_pos, in_bytes);
00192
00193 if (L4_UNLIKELY(in_pos < 0))
00194 return l4_msgtag(Short_err, 0, 0, 0);
00195
00196 if (L4_UNLIKELY(pack.template get<Do_out_data>(mrs->mr, 0, Mr_bytes) < 0))
00197 return l4_msgtag(Short_err, 0, 0, 0);
00198
00199
00200 in_pos = pack.template get<Do_in_items>(&mrs->mr[tag.words()], 0,
00201 tag.items() * Item_bytes);
00202
00203 if (L4_UNLIKELY(in_pos < 0))
00204 return l4_msgtag(Short_err, 0, 0, 0);
00205
00206 asm volatile ("": "=m" (mrs->mr));
00207
00208 // call the server function
00209 long ret = pack.template obj_call<O, typename IPC_TYPE::rpc, ARGS...>(o, args...);
00210
00211 if (!Do_reply)
00212 return l4_msgtag(-L4_ENOREPLY, 0, 0, 0);
00213
00214 // our convention says that negative return value means no
00215 // reply data
00216 if (L4_UNLIKELY(ret < 0))
00217 return l4_msgtag(ret, 0, 0, 0);
00218
00219 // reply with the reply data from the server function
00220 int bytes = pack.template set<Do_out_data>(mrs->mr, 0, Mr_bytes, ret);
00221 if (L4_UNLIKELY(bytes < 0))
00222 return l4_msgtag(-L4_MSGTOOLONG, 0, 0, 0);
00223
00224 unsigned words = (bytes + Word_bytes - 1) / Word_bytes;
00225 bytes = pack.template set<Do_out_items>(&mrs->mr[words], 0,
00226 Mr_bytes - words * Word_bytes,
00227 ret);
00228 if (L4_UNLIKELY(bytes < 0))
00229 return l4_msgtag(-L4_MSGTOOLONG, 0, 0, 0);
00230
00231 unsigned const items = bytes / Item_bytes;
00232 return l4_msgtag(ret, words, items, 0);
00233 }
00234
00235 //-----
00236
00237 template<typename RPCS, typename OPCODE_TYPE>
00238 struct Dispatch_call;
00239
00240 template<typename CLASS>
00241 struct Dispatch_call<L4::Typeid::Raw_ipc<CLASS>, void>
00242 {
00243 template<typename OBJ, typename ...ARGS>
00244 static l4_msgtag_t
00245 call(OBJ *o, l4_utcb_t *utcb, l4_msgtag_t tag, ARGS ...a)
00246 {
00247 return o->op_dispatch(utcb, tag, a...);
00248 }
00249 };
00250
00251 template<typename RPCS>
00252 struct Dispatch_call<RPCS, void>
00253 {
00254 constexpr static unsigned rmask()
00255 { return RPCS::rpc::flags_type::Rights & 3UL; }
00256
00257 template<typename OBJ, typename ...ARGS>
00258 static l4_msgtag_t
00259 call(OBJ *o, l4_utcb_t *utcb, l4_msgtag_t tag, unsigned rights, ARGS ...a)
00260 {
00261 if ((rights & rmask()) != rmask())
00262 return l4_msgtag(-L4_EPERM, 0, 0, 0);
00263
00264 typedef L4::Typeid::Rights<typename RPCS::rpc::class_type> Rights;
00265 return handle_svr_obj_call<RPCS>(o, utcb, tag,
00266 Rights(rights), a...);
00267 }
00268 };
00269
00270
00271 template<typename RPCS, typename OPCODE_TYPE>
00272 struct Dispatch_call
00273 {
00274 constexpr static unsigned rmask()
00275 { return RPCS::rpc::flags_type::Rights & 3UL; }
00276

```

```

00277 template<typename OBJ, typename ...ARGS>
00278 static l4_msgtag_t
00279 _call(OBJ *o, l4_utcb_t *utcb, l4_msgtag_t tag, unsigned rights, OPCODE_TYPE op, ARGS ...)
00280 {
00281 if (L4::Types::Same<typename RPCS::opcode_type, void>::value
00282 || RPCS::Opcode == op)
00283 {
00284 if ((rights & rmask()) != rmask())
00285 return l4_msgtag(-L4_EPERM, 0, 0, 0);
00286
00287 typedef L4::Typeid::Rights<typename RPCS::rpc::class_type> Rights;
00288 return handle_svr_obj_call<RPCS>(o, utcb, tag,
00289 Rights(rights), a...);
00290 }
00291 return Dispatch_call<typename RPCS::next, OPCODE_TYPE>::template
00292 _call<OBJ, ARGS...>(o, utcb, tag, rights, op, a...);
00293 }
00294
00295 template<typename OBJ, typename ...ARGS>
00296 static l4_msgtag_t
00297 call(OBJ *o, l4_utcb_t *utcb, l4_msgtag_t tag, unsigned rights, ARGS ...)
00298 {
00299 OPCODE_TYPE op;
00300 unsigned limit = tag.words() * Word_bytes;
00301 typedef Svr_xmit<OPCODE_TYPE> S;
00302 int err = S::to_svr(reinterpret_cast<char *>(l4_utcb_mr_u(utcb)->mr), 0,
00303 limit, op, Dir_in(), Cls_data());
00304 if (L4_UNLIKELY(err < 0))
00305 return l4_msgtag(-L4_EMMSGTOOSHORT, 0, 0, 0);
00306
00307 return _call<OBJ, ARGS...>(o, utcb, tag, rights, op, a...);
00308 }
00309 };
00310
00311 template<>
00312 struct Dispatch_call<Typeid::Detail::Rpc_end, void>
00313 {
00314 template<typename OBJ, typename ...ARGS>
00315 static l4_msgtag_t
00316 _call(OBJ *, l4_utcb_t *, l4_msgtag_t, unsigned, int, ARGS ...)
00317 { return l4_msgtag(-L4_ENOSYS, 0, 0, 0); }
00318
00319 template<typename OBJ, typename ...ARGS>
00320 static l4_msgtag_t
00321 call(OBJ *, l4_utcb_t *, l4_msgtag_t, unsigned, ARGS ...)
00322 { return l4_msgtag(-L4_ENOSYS, 0, 0, 0); }
00323 };
00324
00325 template<typename OPCODE_TYPE>
00326 struct Dispatch_call<Typeid::Detail::Rpc_end, OPCODE_TYPE> :
00327 Dispatch_call<Typeid::Detail::Rpc_end, void> {};
00328
00329 template<typename RPCS, typename OBJ, typename ...ARGS>
00330 static l4_msgtag_t
00331 dispatch_call(OBJ *o, l4_utcb_t *utcb, l4_msgtag_t tag, unsigned rights, ARGS ...)
00332 {
00333 return Dispatch_call<typename RPCS::type, typename RPCS::opcode_type>::template
00334 call<OBJ, ARGS...>(o, utcb, tag, rights, a...);
00335 }
00336
00337 } // namespace Msg
00338 } // namespace Ipc
00339 } // namespace L4

```

## 17.496 ipc\_server\_loop

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * Copyright (C) 2015, 2017, 2019, 2021-2024 Kernkonzept GmbH.
00004 * Author(s): Alexander Warg <alexander.warg@kernkonzept.com>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include "ipc_epiface"
00011
00012 namespace L4 {
00013
00019
00031 namespace Ipc_svr {
00032
00046 enum Reply_mode

```



```

00047 {
00048 Reply_compound,
00049 Reply_separate
00050 };
00051
00052 struct Ignore_errors
00053 { static void error(l4_msgtag_t, l4_utcb_t *) {} };
00054
00055 struct Default_timeout
00056 { static l4_timeout_t timeout() { return L4_IPC_SEND_TIMEOUT_0; } };
00057
00058 struct Compound_reply
00059 {
00060 static Reply_mode before_reply(l4_msgtag_t, l4_utcb_t *)
00061 { return Reply_compound; }
00062 };
00063
00064 struct Default_setup_wait
00065 { static void setup_wait(l4_utcb_t *, Reply_mode) {} };
00066
00067 template< typename R >
00068 struct Direct_dispatch
00069 {
00070 R &r;
00071
00072 Direct_dispatch(R &r) : r(r) {}
00073
00074 l4_msgtag_t operator () (l4_msgtag_t tag, l4_umword_t obj, l4_utcb_t *utcb)
00075 { return r.dispatch(tag, obj, utcb); }
00076 };
00077
00078 template< typename R >
00079 struct Direct_dispatch<R*>
00080 {
00081 R *r;
00082
00083 Direct_dispatch(R *r) : r(r) {}
00084
00085 l4_msgtag_t operator () (l4_msgtag_t tag, l4_umword_t obj, l4_utcb_t *utcb)
00086 { return r->dispatch(tag, obj, utcb); }
00087 };
00088
00089 #ifdef __EXCEPTIONS
00090 template< typename R, typename Exc> // = L4::Runtime_error>
00091 struct Exc_dispatch : private Direct_dispatch<R>
00092 {
00093 Exc_dispatch(R r) : Direct_dispatch<R>(r) {}
00094
00095 l4_msgtag_t operator () (l4_msgtag_t tag, l4_umword_t obj, l4_utcb_t *utcb)
00096 {
00097 try
00098 {
00099 return Direct_dispatch<R>::operator () (tag, obj, utcb);
00100 }
00101 catch (Exc &e)
00102 {
00103 return l4_msgtag(e.err_no(), 0, 0, 0);
00104 }
00105 catch (int err)
00106 {
00107 return l4_msgtag(err, 0, 0, 0);
00108 }
00109 catch (long err)
00110 {
00111 return l4_msgtag(err, 0, 0, 0);
00112 }
00113 }
00114 };
00115
00116 template< typename R, typename Exc, typename Printer >
00117 struct Dbg_dispatch : private Direct_dispatch<R>
00118 {
00119 Dbg_dispatch(R r, Printer p) : Direct_dispatch<R>(r), _printer(p) {}
00120
00121 l4_msgtag_t operator () (l4_msgtag_t tag, l4_umword_t obj, l4_utcb_t *utcb)
00122 {
00123 try
00124 {
00125 return Direct_dispatch<R>::operator () (tag, obj, utcb);
00126 }
00127 catch (Exc &e)
00128 {
00129 _printer.printf("Error in handling IPC: %s: %s\n", e.str(),
00130 e.extra_str());
00131 return l4_msgtag(e.err_no(), 0, 0, 0);
00132 }
00133 catch (int err)
00134 {
00135 return l4_msgtag(err, 0, 0, 0);
00136 }
00137 }
00138 };

```

```

00206 {
00207 _printer.printf("Error in handling IPC: %d (%s)\n", err,
00208 l4sys_errtostr(err));
00209 return l4_msgtag(err, 0, 0, 0);
00210 }
00211 catch (long err)
00212 {
00213 _printer.printf("Error in handling IPC: %ld (%s)\n", err,
00214 l4sys_errtostr(err));
00215 return l4_msgtag(err, 0, 0, 0);
00216 }
00217 }
00218
00219 Printer _printer;
00220 };
00221 #endif
00222
00223 class Br_manager_no_buffers : public Server_iface
00224 {
00225 public:
00226 int alloc_buffer_demand(Demand const &demand) override
00227 {
00228 if (!demand.no_demand())
00229 return -L4_ENOMEM;
00230 return L4_EOK;
00231 }
00232
00233 L4::Cap<void> get_rcv_cap(int) const override
00234 { return L4::Cap<void>::Invalid; }
00235
00236 int realloc_rcv_cap(int) override
00237 { return -L4_ENOMEM; }
00238
00239 int add_timeout(Timeout *, l4_kernel_clock_t) override
00240 { return -L4_ENOSYS; }
00241
00242 int remove_timeout(Timeout *) override
00243 { return -L4_ENOSYS; }
00244
00245 protected:
00246 unsigned first_free_br() const
00247 { return 1; }
00248
00249 void setup_wait(l4_utcb_t *utcb, L4::Ipc_svr::Reply_mode)
00250 {
00251 l4_buf_regs_t *br = l4_utcb_br_u(utcb);
00252 br->bdr = 0;
00253 br->br[0] = 0;
00254 }
00255 };
00256
00257 struct Default_loop_hooks :
00258 public Ignore_errors, public Default_timeout, public Compound_reply,
00259 Br_manager_no_buffers
00260 {};
00261
00262 template< typename LOOP_HOOKS = Ipc_svr::Default_loop_hooks >
00263 class Server :
00264 public LOOP_HOOKS
00265 {
00266 public:
00267 /* Internal note: After all users have been converted, remove this
00268 * constructor. Also remove the constructor below then. */
00269 explicit Server(l4_utcb_t *)
00270 L4_DEPRECATED("Do not specify the UTCB with the constructor. "
00271 "Supply it on the loop function if needed.")
00272 {}
00273
00274 /* Internal note: Remove this constructor when the above deprecated
00275 * constructor with the UTCB pointer is also removed. */
00276 Server() {}
00277
00278 template< typename DISPATCH >
00279 inline L4_NORETURN void internal_loop(DISPATCH dispatch, l4_utcb_t *);
00280
00281 template< typename R >
00282 inline L4_NORETURN void loop_noexc(R r, l4_utcb_t *u = l4_utcb())
00283 { internal_loop(Ipc_svr::Direct_dispatch<R>(r), u); }
00284
00285 #ifdef __EXCEPTIONS
00286 template< typename EXC, typename R >
00287 inline L4_NORETURN void loop(R r, l4_utcb_t *u = l4_utcb())
00288 {
00289 internal_loop(Ipc_svr::Exc_dispatch<R, EXC>(r), u);
00290 // function will never return
00291 }
00292 #endif
00293
00294 };

```

```

00359 }
00360
00367 template< typename EXC, typename R, typename Printer >
00368 inline L4_NORETURN void loop_dbg(R r, Printer p, l4_utcb_t *u = l4_utcb())
00369 {
00370 internal_loop(Ipc_svr::Dbg_dispatch<R, EXC, Printer>(r, p), u);
00371 // function will never return
00372 }
00373 #endif
00374 protected:
00376 inline l4_msgtag_t reply_n_wait(l4_msgtag_t reply, l4_umword_t *p, l4_utcb_t *);
00377 };
00378
00379 template< typename L >
00380 inline l4_msgtag_t
00381 Server<L>::reply_n_wait(l4_msgtag_t reply, l4_umword_t *p, l4_utcb_t *utcb)
00382 {
00383 if (reply.label() != -L4_ENOREPLY)
00384 {
00385 Ipc_svr::Reply_mode m = this->before_reply(reply, utcb);
00386 if (m == Ipc_svr::Reply_compound)
00387 {
00388 this->setup_wait(utcb, m);
00389 return l4_ipc_reply_and_wait(utcb, reply, p, this->timeout());
00390 }
00391 else
00392 {
00393 l4_msgtag_t res = l4_ipc_send(L4_INVALID_CAP | L4_SYSF_REPLY, utcb, reply, this->timeout());
00394 if (res.has_error())
00395 return res;
00396 }
00397 }
00398 this->setup_wait(utcb, Ipc_svr::Reply_separate);
00399 return l4_ipc_wait(utcb, p, this->timeout());
00400 }
00401
00402 template< typename L >
00403 template< typename DISPATCH >
00404 inline L4_NORETURN void
00405 Server<L>::internal_loop(DISPATCH dispatch, l4_utcb_t *utcb)
00406 {
00407 l4_msgtag_t res;
00408 l4_umword_t p;
00409 l4_msgtag_t r = l4_msgtag(-L4_ENOREPLY, 0, 0, 0);
00410
00411 while (true)
00412 {
00413 res = reply_n_wait(r, &p, utcb);
00414 if (res.has_error())
00415 {
00416 this->error(res, utcb);
00417 r = l4_msgtag(-L4_ENOREPLY, 0, 0, 0);
00418 continue;
00419 }
00420
00421 r = dispatch(res, p, utcb);
00422 }
00423 }
00424
00425 } // namespace L4

```

## 17.497 ipc\_string

```

00001 // vi:set ft=c++: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2014 Alexander Warg <alexander.warg@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include "types"
00010 #include "ipc_basics"
00011 #include "ipc_array"
00012
00013 namespace L4 { namespace Ipc {
00014
00015 template<typename CHAR = char const, typename LEN = unsigned long>
00016 struct String : Array<CHAR, LEN>
00017 {
00018 static LEN strlength(CHAR *d) { LEN l = 0; while (d[l]) ++l; return l; }
00019 String() {}
00020 String(CHAR *d) : Array<CHAR, LEN>(strlength(d) + 1, d) {}
00021 };
00022 } }

```

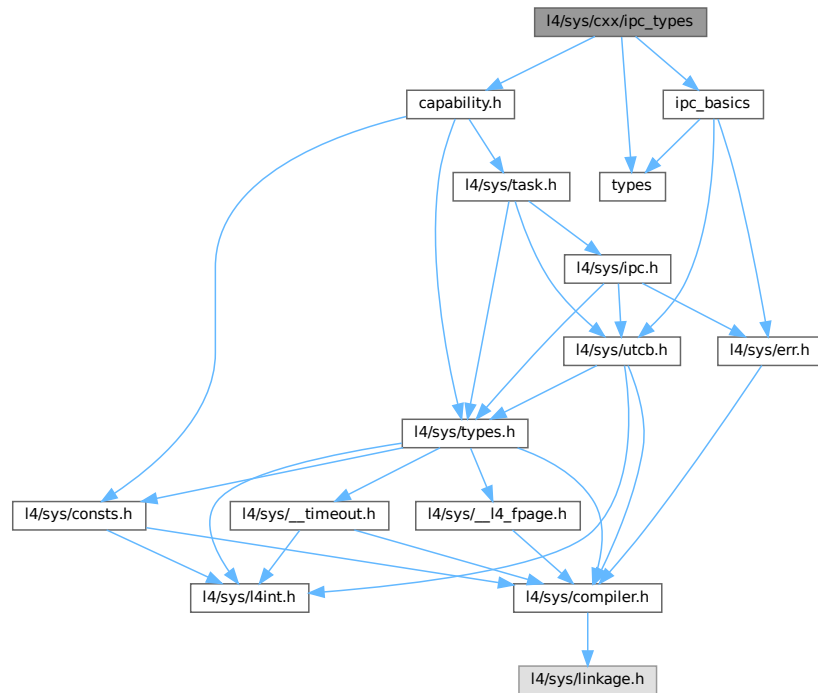
```

00021 String(LEN len, CHAR *d) : Array<CHAR, LEN>(len, d) {}
00022 void copy_in(CHAR const *s)
00023 {
00024 if (this->length < 1)
00025 return;
00026
00027 LEN i;
00028 for (i = 0; i < this->length - 1 && s[i]; ++i)
00029 this->data[i] = s[i];
00030 this->length = i + 1;
00031 this->data[i] = CHAR();
00032 }
00033 };
00034
00035 #if __cplusplus >= 201103L
00036 template< typename CHAR = char, typename LEN_TYPE = unsigned long,
00037 LEN_TYPE MAX = (L4_UTCB_GENERIC_DATA_SIZE *
00038 sizeof(l4_umword_t)) / sizeof(CHAR) >
00039 using String_in_buf = Array_in_buf<CHAR, LEN_TYPE, MAX>;
00040 #endif
00041
00042 namespace Msg {
00043 template<typename A, typename LEN>
00044 struct Clnt_xmit< String<A, LEN> > : Clnt_xmit< Array<A, LEN> > {};
00045
00046 template<typename A, typename LEN, typename CLASS>
00047 struct Svr_val_ops< String<A, LEN>, Dir_in, CLASS >
00048 : Svr_val_ops< Array_ref<A, LEN>, Dir_in, CLASS >
00049 {
00050 typedef Svr_val_ops< Array_ref<A, LEN>, Dir_in, CLASS > Base;
00051 typedef typename Base::svr_type svr_type;
00052 using Base::to_svr;
00053 static int to_svr(char *msg, unsigned offset, unsigned limit,
00054 svr_type &a, Dir_in dir, Cls_data cls)
00055 {
00056 int r = Base::to_svr(msg, offset, limit, a, dir, cls);
00057 if (r < 0)
00058 return r;
00059
00060 // correct clients send at least the zero terminator
00061 if (a.length < 1)
00062 return -L4_MSGTOOSHORT;
00063
00064 typedef typename L4::Types::Remove_const<A>::type elem_type;
00065 const_cast<elem_type*>(a.data)[a.length - 1] = A();
00066 return r;
00067 }
00068 };
00069
00070 template<typename A, typename LEN>
00071 struct Clnt_xmit<String<A, LEN> &> : Clnt_xmit<Array<A, LEN> &>
00072 {
00073 typedef Array<A, LEN> &type;
00074
00075 using Clnt_xmit<type>::from_msg;
00076 static int from_msg(char *msg, unsigned offset, unsigned limit, long ret,
00077 Array<A, LEN> &a, Dir_out dir, Cls_data cls)
00078 {
00079 int r = Clnt_xmit<type>::from_msg(msg, offset, limit, ret, a, dir, cls);
00080 if (r < 0)
00081 return r;
00082
00083 // check for a bad servers
00084 if (a.length < 1)
00085 return -L4_MSGTOOSHORT;
00086
00087 a.data[a.length - 1] = A();
00088 return r;
00089 };
00090 };
00091
00092 template<typename A, typename LEN>
00093 struct Clnt_xmit<String<A, LEN> *> : Clnt_xmit<String<A, LEN> &> {};
00094
00095 template<typename A, typename LEN, typename CLASS>
00096 struct Svr_val_ops<String<A, LEN>, Dir_out, CLASS>
00097 : Svr_val_ops<Array_ref<A, LEN>, Dir_out, CLASS>
00098 {};
00099
00100 template<typename A, typename LEN>
00101 struct Is_valid_rpc_type<String<A, LEN> const *> : L4::Types::False {};
00102 template<typename A, typename LEN>
00103 struct Is_valid_rpc_type<String<A, LEN> const &> : L4::Types::False {};
00104
00105 } // namespace Msg
00106
00107 }

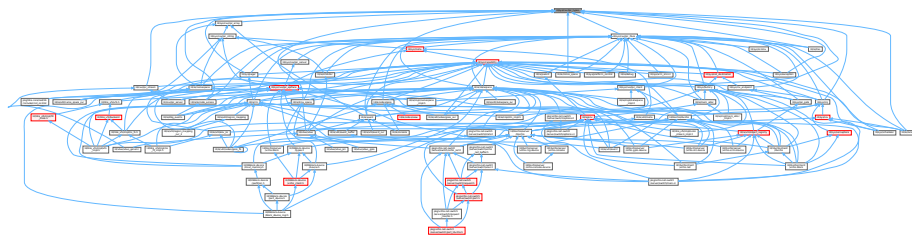
```

## 17.498 I4/sys/cxx/ipc\_types File Reference

```
#include "capability.h"
#include "types"
#include "ipc_basics"
Include dependency graph for ipc_types:
```



This graph shows which files directly or indirectly include this file:



### Data Structures

- struct [L4::lpc::In\\_out< T >](#)  
Mark an argument as in-out argument.
- struct [L4::lpc::As\\_value< T >](#)  
Pass the argument as plain data value.
- struct [L4::lpc::Opt< T >](#)  
Attribute for defining an optional RPC argument.

- class [L4::lpc::Small\\_buf](#)  
*A receive item for receiving a single object capability.*
- class [L4::lpc::Gen\\_fpage](#)  
*Generic RPC base for typed message items.*
- class [L4::lpc::Snd\\_fpage](#)  
*Send item or return item.*
- class [L4::lpc::Rcv\\_fpage](#)  
*Non-small receive item.*
- class [L4::lpc::Cap< T >](#)  
*Capability type for RPC interfaces (see [L4::Cap< T >](#)).*

## Namespaces

- namespace [L4](#)  
*L4 low-level kernel interface.*
- namespace [L4::lpc](#)  
*IPC related functionality.*
- namespace [L4::lpc::Msg](#)  
*IPC Message related functionality.*

## Functions

- `template<typename T>`  
`Cap< T > L4::lpc::make_cap (L4::Cap< T > cap, unsigned rights) noexcept`  
*Make an [L4::lpc::Cap< T >](#) for the given capability and rights.*
- `template<typename T>`  
`Cap< T > L4::lpc::make_cap_rw (L4::Cap< T > cap) noexcept`  
*Make an [L4::lpc::Cap< T >](#) for the given capability with [L4\\_CAP\\_FPAGE\\_RW](#) rights.*
- `template<typename T>`  
`Cap< T > L4::lpc::make_cap_rws (L4::Cap< T > cap) noexcept`  
*Make an [L4::lpc::Cap< T >](#) for the given capability with [L4\\_CAP\\_FPAGE\\_RWS](#) rights.*
- `template<typename T>`  
`Cap< T > L4::lpc::make_cap_full (L4::Cap< T > cap) noexcept`  
*Make an [L4::lpc::Cap< T >](#) for the given capability with full fpage and object-specific rights.*

## 17.499 ipc\_types

[Go to the documentation of this file.](#)

```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2014 Alexander Warg <alexander.warg@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include "capability.h"
00010 #include "types"
00011 #include "ipc_basics"
00012
00013 namespace L4 {
00014
00015 typedef int Opcode;
00016
00017 namespace Ipc {
```

```

00022
00031 template<typename T> struct L4_EXPORT Out;
00032
00033
00041 template<typename T> struct L4_EXPORT In_out
00042 {
00043 T v;
00044 In_out() {}
00045 In_out(T v) : v(v) {}
00046 operator T () const { return v; }
00047 operator T & () { return v; }
00048 };
00049
00050 namespace Msg {
00051 template<typename A> struct Elem< In_out<A *> > : Elem<A *> {};
00052
00053 template<typename A>
00054 struct Svr_xmit< In_out<A *> > : Svr_xmit<A *>, Svr_xmit<A const *>
00055 {
00056 using Svr_xmit<A *>::from_svr;
00057 using Svr_xmit<A const *>::to_svr;
00058 };
00059
00060 template<typename A>
00061 struct Clnt_xmit< In_out<A *> > : Clnt_xmit<A *>, Clnt_xmit<A const *>
00062 {
00063 using Clnt_xmit<A *>::from_msg;
00064 using Clnt_xmit<A const *>::to_msg;
00065 };
00066
00067 template<typename A>
00068 struct Is_valid_rpc_type< In_out<A *> > : Is_valid_rpc_type<A *> {};
00069 template<typename A>
00070 struct Is_valid_rpc_type< In_out<A const *> > : L4::Types::False {};
00071
00072 #ifdef CONFIG_ALLOW_REFS
00073 template<typename A> struct Elem< In_out<A &> > : Elem<A &> {};
00074
00075 template<typename A>
00076 struct Svr_xmit< In_out<A &> > : Svr_xmit<A &>, Svr_xmit<A const &>
00077 {
00078 using Svr_xmit<A &>::from_svr;
00079 using Svr_xmit<A const &>::to_svr;
00080 };
00081
00082 template<typename A>
00083 struct Clnt_xmit< In_out<A &> > : Clnt_xmit<A &>, Clnt_xmit<A const &>
00084 {
00085 using Clnt_xmit<A &>::from_msg;
00086 using Clnt_xmit<A const &>::to_msg;
00087 };
00088
00089 template<typename A>
00090 struct Is_valid_rpc_type< In_out<A &> > : Is_valid_rpc_type<A &> {};
00091 template<typename A>
00092 struct Is_valid_rpc_type< In_out<A const &> > : L4::Types::False {};
00093
00094 #else
00095
00096 template<typename A>
00097 struct Is_valid_rpc_type< In_out<A &> > : L4::Types::False {};
00098
00099 #endif
00100
00101 // Value types don't make sense for output.
00102 template<typename A>
00103 struct Is_valid_rpc_type< In_out<A > > : L4::Types::False {};
00104
00105 }
00106
00107
00116 template<typename T> struct L4_EXPORT As_value
00117 {
00118 typedef T value_type;
00119 T v;
00120 As_value() noexcept {}
00121 As_value(T v) noexcept : v(v) {}
00122 operator T () const noexcept { return v; }
00123 operator T & () noexcept { return v; }
00124 };
00125
00126 namespace Msg {
00127 template<typename T> struct Class< As_value<T> > : Cls_data {};
00128 template<typename T> struct Elem< As_value<T> > : Elem<T> {};
00129 template<typename T> struct Elem< As_value<T> *> : Elem<T *> {};
00130 }
00131

```

```

00132
00136 template<typename T> struct L4_EXPORT Opt
00137 {
00138 T _value;
00139 bool _valid;
00140
00142 Opt() noexcept : _valid(false) {}
00143
00145 Opt(T value) noexcept : _value(value), _valid(true) {}
00146
00148 Opt &operator = (T value) noexcept
00149 {
00150 this->_value = value;
00151 this->_valid = true;
00152 return *this;
00153 }
00154
00156 void set_valid(bool valid = true) noexcept { _valid = valid; }
00157
00159 T *operator -> () noexcept { return &this->_value; }
00161 T const *operator -> () const noexcept { return &this->_value; }
00163 T value() const noexcept { return this->_value; }
00165 T &value() noexcept { return this->_value; }
00167 bool is_valid() const noexcept { return this->_valid; }
00168 };
00169
00170 namespace Msg {
00171 template<typename T> struct Elem< Opt<T &> > : Elem<T &>
00172 {
00173 enum { Is_optional = true };
00174 typedef Opt<typename Elem<T &>::svr_type> &svr_arg_type;
00175 typedef Opt<typename Elem<T &>::svr_type> svr_type;
00176 };
00177
00178 template<typename T> struct Elem< Opt<T *> > : Elem<T *>
00179 {
00180 enum { Is_optional = true };
00181 typedef Opt<typename Elem<T *>::svr_type> &svr_arg_type;
00182 typedef Opt<typename Elem<T *>::svr_type> svr_type;
00183 };
00184
00185
00186
00187 template<typename T, typename CLASS>
00188 struct Svr_val_ops<Opt<T>, Dir_out, CLASS> : Svr_noops< Opt<T> >
00189 {
00190 typedef Opt<T> svr_type;
00191 typedef Svr_val_ops<T, Dir_out, CLASS> Native;
00192
00193 using Svr_noops< Opt<T> >::to_svr;
00194 static int to_svr(char *msg, unsigned offset, unsigned limit,
00195 Opt<T> &arg, Dir_out, CLASS) noexcept
00196 {
00197 return Native::to_svr(msg, offset, limit, arg.value(), Dir_out(), CLASS());
00198 }
00199
00200 using Svr_noops< Opt<T> >::from_svr;
00201 static int from_svr(char *msg, unsigned offset, unsigned limit, long ret,
00202 svr_type &arg, Dir_out, CLASS) noexcept
00203 {
00204 if (arg.is_valid())
00205 return Native::from_svr(msg, offset, limit, ret, arg.value(),
00206 Dir_out(), CLASS());
00207 return offset;
00208 }
00209 };
00210
00211 template<typename T> struct Elem< Opt<T> > : Elem<T>
00212 {
00213 enum { Is_optional = true };
00214 typedef Opt<T> arg_type;
00215 };
00216
00217 template<typename T> struct Elem< Opt<T const *> > : Elem<T const *>
00218 {
00219 enum { Is_optional = true };
00220 typedef Opt<T const *> arg_type;
00221 };
00222
00223 template<typename T>
00224 struct Is_valid_rpc_type< Opt<T const &> > : L4::Types::False {};
00225
00226 template<typename T, typename CLASS>
00227 struct Clnt_val_ops<Opt<T>, Dir_in, CLASS> : Clnt_noops< Opt<T> >
00228 {
00229 typedef Opt<T> arg_type;
00230 typedef Detail::_Clnt_val_ops<typename Elem<T>::arg_type, Dir_in, CLASS> Native;

```



```

00231
00232 using Clnt_noops< Opt<T> >::to_msg;
00233 static int to_msg(char *msg, unsigned offset, unsigned limit,
00234 arg_type arg, Dir_in, CLASS) noexcept
00235 {
00236 if (arg.is_valid())
00237 return Native::to_msg(msg, offset, limit,
00238 Detail::_Plain<T>::deref(arg.value()),
00239 Dir_in(), CLASS());
00240 return offset;
00241 }
00242 };
00243
00244 template<typename T> struct Class< Opt<T> > :
00245 Class< typename Detail::_Plain<T>::type > {};
00246 template<typename T> struct Direction< Opt<T> > : Direction<T> {};
00247 }
00248
00257 class L4_EXPORT Small_buf
00258 {
00259 public:
00267 explicit Small_buf(L4::Cap<void> cap, unsigned long flags = 0) noexcept
00268 : _data(cap.cap() | L4_RCV_ITEM_SINGLE_CAP | flags) {}
00269
00274 explicit Small_buf(l4_cap_idx_t cap, unsigned long flags = 0) noexcept
00275 : _data(cap | L4_RCV_ITEM_SINGLE_CAP | flags) {}
00276
00278 l4_umword_t raw() const noexcept { return _data; }
00279 private:
00280 l4_umword_t _data;
00281 };
00282
00286 class L4_EXPORT Gen_fpage
00287 {
00288 public:
00290 enum Type
00291 {
00292 Special = L4_FPAGE_SPECIAL « 4,
00293 Memory = L4_FPAGE_MEMORY « 4,
00294 Io = L4_FPAGE_IO « 4,
00295 Obj = L4_FPAGE_OBJ « 4
00296 };
00297
00299 Gen_fpage(l4_umword_t base, l4_umword_t data) noexcept
00300 : _base(base), _data(data)
00301 {}
00302
00304 l4_umword_t data() const noexcept { return _data; }
00306 l4_umword_t base_x() const noexcept { return _base; }
00307
00308 protected:
00309 l4_umword_t _base;
00310 l4_umword_t _data;
00311 };
00312
00323 class Snd_fpage : public Gen_fpage
00324 {
00325 public:
00328 enum Map_type
00329 {
00330 Map = L4_MAP_ITEM_MAP,
00331 Grant = L4_MAP_ITEM_GRANT,
00332 };
00333
00336 enum Cacheopt
00337 {
00338 None = 0,
00339 Cached = L4_FPAGE_CACHEABLE « 4,
00340 Buffered = L4_FPAGE_BUFFERABLE « 4,
00341 Uncached = L4_FPAGE_UNCACHEABLE « 4
00342 };
00343
00346 enum Continue
00347 {
00348 Single = 0,
00349 Last = 0,
00350 More = L4_ITEM_CONT,
00351 Compound = L4_ITEM_CONT,
00352 };
00353
00355 Snd_fpage(l4_umword_t base = 0, l4_umword_t data = 0) noexcept
00356 : Gen_fpage(base, data)
00357 {}
00358
00370 Snd_fpage(l4_fpage_t const &fp, l4_addr_t snd_base = 0,
00371 Map_type map_type = Map,
00372 Cacheopt cache = None, Continue cont = Last) noexcept

```

```

00373 : Gen_fpage(L4_ITEM_MAP | (snd_base & (~0UL << 12)) | l4_umword_t(map_type)
00374 | l4_umword_t(cache) | l4_umword_t(cont),
00375 fp.raw)
00376 {}
00377
00386 Snd_fpage(L4::Cap<void> cap, unsigned rights, Map_type map_type = Map) noexcept
00387 : Gen_fpage(L4_ITEM_MAP | l4_umword_t(map_type) | (rights & 0xf0),
00388 cap.fpage(rights).raw)
00389 {}
00390
00402 static Snd_fpage obj(l4_cap_idx_t base, int order,
00403 unsigned char rights,
00404 l4_addr_t snd_base = 0,
00405 Map_type map_type = Map,
00406 Continue cont = Last) noexcept
00407 {
00408 return Snd_fpage(l4_obj_fpage(base, order, rights), snd_base,
00409 map_type, None, cont);
00410 }
00411
00424 static Snd_fpage mem(l4_addr_t base, int order,
00425 unsigned char rights,
00426 l4_addr_t snd_base = 0,
00427 Map_type map_type = Map,
00428 Cacheopt cache = None, Continue cont = Last) noexcept
00429 {
00430 return Snd_fpage(l4_fpage(base, order, rights), snd_base, map_type, cache,
00431 cont);
00432 }
00433
00445 static Snd_fpage io(unsigned long base, int order,
00446 unsigned char rights,
00447 l4_addr_t snd_base = 0,
00448 Map_type map_type = Map,
00449 Continue cont = Last) noexcept
00450 {
00451 return Snd_fpage(l4_fpage_set_rights(l4_iofpage(base, order), rights),
00452 snd_base, map_type, None, cont);
00453 }
00454
00457 unsigned order() const noexcept { return (_data >> 6) & 0x3f; }
00458
00461 unsigned snd_order() const noexcept { return (_data >> 6) & 0x3f; }
00462
00465 unsigned rcv_order() const noexcept { return (_base >> 6) & 0x3f; }
00466
00469 l4_addr_t base() const noexcept { return _data & (~0UL << 12); }
00470
00473 l4_addr_t snd_base() const noexcept { return _base & (~0UL << 12); }
00474
00477 void snd_base(l4_addr_t b) noexcept { _base = (_base & ~(~0UL << 12)) | (b & (~0UL << 12)); }
00478
00480 bool is_valid() const noexcept { return _base & L4_ITEM_MAP; }
00481
00496 bool cap_received() const noexcept { return (_base & 0x3e) == 0x38; }
00512 bool id_received() const noexcept { return (_base & 0x3e) == 0x3c; }
00528 bool local_id_received() const noexcept { return (_base & 0x3e) == 0x3e; }
00535 bool is_compound() const noexcept { return _base & 1; }
00536 };
00537
00544 class Rcv_fpage : public Gen_fpage
00545 {
00546 public:
00550 Rcv_fpage() noexcept : Gen_fpage(0, 0), _rcv_task(L4_INVALID_CAP) {}
00551
00561 Rcv_fpage(l4_fpage_t const &fp, l4_addr_t snd_base = 0,
00562 l4_cap_idx_t rcv_task = L4_INVALID_CAP) noexcept
00563 : Gen_fpage(L4_ITEM_MAP | (snd_base & (~0UL << 12))
00564 | (l4_is_valid_cap(rcv_task) ? L4_RCV_ITEM_FORWARD_MAPPINGS : 0),
00565 fp.raw),
00566 _rcv_task(rcv_task)
00567 {}
00568
00578 static Rcv_fpage obj(l4_cap_idx_t base, int order, l4_addr_t snd_base = 0,
00579 L4::Cap<void> rcv_task = L4::Cap<void>::Invalid) noexcept
00580 {
00581 return Rcv_fpage(l4_obj_fpage(base, order, 0), snd_base,
00582 rcv_task.cap());
00583 }
00584
00594 static Rcv_fpage mem(l4_addr_t base, int order, l4_addr_t snd_base = 0,
00595 L4::Cap<void> rcv_task = L4::Cap<void>::Invalid) noexcept
00596 {
00597 return Rcv_fpage(l4_fpage(base, order, 0), snd_base, rcv_task.cap());
00598 }
00599
00609 static Rcv_fpage io(unsigned long base, int order, l4_addr_t snd_base = 0,

```

```

00610 L4::Cap<void> rcv_task = L4::Cap<void>::Invalid) noexcept
00611 {
00612 return Rcv_fpage(l4_iofpage(base, order), snd_base, rcv_task.cap());
00613 }
00614
00620 l4_cap_idx_t rcv_task() const { return _rcv_task; }
00621
00625 bool forward_mappings() const noexcept
00626 { return _base & L4_RCV_ITEM_FORWARD_MAPPINGS; }
00627
00628 protected:
00629 l4_cap_idx_t _rcv_task;
00630 };
00631
00632
00633 namespace Msg {
00634
00635 // Snd_fpage are out items
00636 template<> struct Class<L4::Ipc::Snd_fpage> : Cls_item {};
00637
00638 // Rcv_fpage are buffer items
00639 template<> struct Class<L4::Ipc::Rcv_fpage> : Cls_buffer {};
00640
00641 template<>
00642 struct Clnt_val_ops<L4::Ipc::Rcv_fpage, Dir_in, Cls_buffer>
00643 : Clnt_noops<L4::Ipc::Rcv_fpage>
00644 {
00645 using Clnt_noops<L4::Ipc::Rcv_fpage>::to_msg;
00646
00647 static int to_msg(char *msg, unsigned offs, unsigned limit,
00648 L4::Ipc::Rcv_fpage arg, Dir_in, Cls_buffer) noexcept
00649 {
00650 offs = align_to<l4_umword_t>(offs);
00651 unsigned words = arg.forward_mappings() ? 3 : 2;
00652 if (L4_UNLIKELY(!check_size<l4_umword_t>(offs, limit, words)))
00653 return -L4_EMSGTOOLONG;
00654 auto *buf = reinterpret_cast<l4_umword_t*>(msg + offs);
00655 *buf++ = arg.base_x();
00656 *buf++ = arg.data();
00657 if (arg.forward_mappings())
00658 *buf++ = arg.rcv_task();
00659 return offs + sizeof(l4_umword_t) * words;
00660 }
00661 };
00662
00663
00664 // Remove receive buffers from server-side arguments
00665 template<> struct Elem<L4::Ipc::Rcv_fpage>
00666 {
00667 typedef L4::Ipc::Rcv_fpage arg_type;
00668 typedef void svr_type;
00669 typedef void svr_arg_type;
00670 enum { Is_optional = false };
00671 };
00672
00673 // Small_buf are buffer items
00674 template<> struct Class<L4::Ipc::Small_buf> : Cls_buffer {};
00675
00676 // Remove receive buffers from server-side arguments
00677 template<> struct Elem<L4::Ipc::Small_buf>
00678 {
00679 typedef L4::Ipc::Small_buf arg_type;
00680 typedef void svr_type;
00681 typedef void svr_arg_type;
00682 enum { Is_optional = false };
00683 };
00684 } // namespace Msg
00685
00686 // L4::Cap<> handling
00687
00698 template<typename T> class Cap
00699 {
00700 template<typename O> friend class Cap;
00701 l4_umword_t _cap_n_rights;
00702
00703 public:
00704 enum
00705 {
00711 Rights_mask = 0xff,
00712
00717 Cap_mask = L4_CAP_MASK
00718 };
00719
00721 template<typename O>
00722 Cap(Cap<O> const &o) noexcept : _cap_n_rights(o._cap_n_rights)
00723 {
00724 L4::Cap<T>::template check_convertible_from<O>();

```

```

00725 }
00726
00728 Cap(L4::Cap<T> cap) noexcept
00729 : _cap_n_rights((cap.cap() & Cap_mask) | (cap ? L4_CAP_FPAGE_R : 0))
00730 {}
00731
00733 template<typename O>
00734 Cap(L4::Cap<O> cap) noexcept
00735 : _cap_n_rights((cap.cap() & Cap_mask) | (cap ? L4_CAP_FPAGE_R : 0))
00736 {
00737 L4::Cap<T>::template check_convertible_from<O>();
00738 }
00739
00741 Cap() noexcept : _cap_n_rights(L4_INVALID_CAP) {}
00742
00750 Cap(L4::Cap<T> cap, unsigned char rights) noexcept
00751 : _cap_n_rights((cap.cap() & Cap_mask) | (rights & Rights_mask)) {}
00752
00758 static Cap from_ci(l4_cap_idx_t c) noexcept
00759 { return Cap(L4::Cap<T>(c & Cap_mask), c & Rights_mask); }
00760
00762 L4::Cap<T> cap() const noexcept
00763 { return L4::Cap<T>(_cap_n_rights & Cap_mask); }
00764
00766 unsigned rights() const noexcept
00767 { return _cap_n_rights & Rights_mask; }
00768
00770 L4::Ipc::Snd_fpage fpage() const noexcept
00771 { return L4::Ipc::Snd_fpage(cap(), rights()); }
00772
00774 bool is_valid() const noexcept
00775 { return !(_cap_n_rights & L4_INVALID_CAP_BIT); }
00776 };
00777
00784 template<typename T>
00785 Cap<T> make_cap(L4::Cap<T> cap, unsigned rights) noexcept
00786 { return Cap<T>(cap, rights); }
00787
00794 template<typename T>
00795 Cap<T> make_cap_rw(L4::Cap<T> cap) noexcept
00796 { return Cap<T>(cap, L4_CAP_FPAGE_RW); }
00797
00804 template<typename T>
00805 Cap<T> make_cap_rws(L4::Cap<T> cap) noexcept
00806 { return Cap<T>(cap, L4_CAP_FPAGE_RWS); }
00807
00823 template<typename T>
00824 Cap<T> make_cap_full(L4::Cap<T> cap) noexcept
00825 { return Cap<T>(cap, L4_CAP_FPAGE_RWS | L4_FPAGE_C_OBJ_RIGHTS); }
00826
00827 // caps are special the have an invalid representation
00828 template<typename T> struct L4_EXPORT Opt< Cap<T> >
00829 {
00830 Cap<T> _value;
00831 Opt() noexcept {}
00832 Opt(Cap<T> value) noexcept : _value(value) {}
00833 Opt(L4::Cap<T> value) noexcept : _value(value) {}
00834 Opt &operator = (Cap<T> value) noexcept
00835 { this->_value = value; return *this; }
00836 Opt &operator = (L4::Cap<T> value) noexcept
00837 { this->_value = value; return *this; }
00838
00839 Cap<T> value() const noexcept { return this->_value; }
00840 bool is_valid() const noexcept { return this->_value.is_valid(); }
00841 };
00842
00843
00844 namespace Msg {
00845 // prohibit L4::Cap as argument
00846 template<typename A>
00847 struct Is_valid_rpc_type< L4::Cap<A> > : L4::Types::False {};
00848
00849 template<typename A> struct Class< Cap<A> > : Cls_item {};
00850 template<typename A> struct Elem< Cap<A> >
00851 {
00852 enum { Is_optional = false };
00853 typedef Cap<A> arg_type;
00854 typedef L4::Ipc::Snd_fpage svr_type;
00855 typedef L4::Ipc::Snd_fpage svr_arg_type;
00856 };
00857
00858
00859 template<typename A, typename CLASS>
00860 struct Svr_val_ops<Cap<A>, Dir_in, CLASS> :
00861 Svr_val_ops<L4::Ipc::Snd_fpage, Dir_in, CLASS>
00862 {};
00863

```

```

00864 template<typename A, typename CLASS>
00865 struct Clnt_val_ops<Cap<A>, Dir_in, CLASS> :
00866 Clnt_noops< Cap<A> >
00867 {
00868 using Clnt_noops< Cap<A> >::to_msg;
00869
00870 static int to_msg(char *msg, unsigned offset, unsigned limit,
00871 Cap<A> arg, Dir_in, Cls_item) noexcept
00872 {
00873 // passing an invalid cap as mandatory argument is an error
00874 // XXX: This checks for a client calling error, we could
00875 // also just ignore this for performance reasons and
00876 // let the client fail badly (Alex: I'd prefer this)
00877 if (L4_UNLIKELY(!arg.is_valid()))
00878 return -L4_MSGMISSARG;
00879
00880 return msg_add(msg, offset, limit, arg.fpage());
00881 }
00882 };
00883
00884 template<typename A>
00885 struct Elem<Out<L4::Cap<A> > >
00886 {
00887 enum { Is_optional = false };
00888 typedef L4::Cap<A> arg_type;
00889 typedef Ipc::Cap<A> svr_type;
00890 typedef svr_type &svr_arg_type;
00891 };
00892
00893 template<typename A> struct Direction< Out< L4::Cap<A> > > : Dir_out {};
00894 template<typename A> struct Class< Out< L4::Cap<A> > > : Cls_item {};
00895
00896 template<typename A>
00897 struct Clnt_val_ops< L4::Cap<A>, Dir_out, Cls_item > :
00898 Clnt_noops< L4::Cap<A> >
00899 {
00900 using Clnt_noops< L4::Cap<A> >::to_msg;
00901 static int to_msg(char *msg, unsigned offset, unsigned limit,
00902 L4::Cap<A> arg, Dir_in, Cls_buffer) noexcept
00903 {
00904 if (L4_UNLIKELY(!arg.is_valid()))
00905 return -L4_MSGMISSARG; // no buffer inserted
00906 return msg_add(msg, offset, limit, Small_buf(arg));
00907 }
00908 };
00909
00910 template<typename A>
00911 struct Svr_val_ops< L4::Ipc::Cap<A>, Dir_out, Cls_item > :
00912 Svr_noops<Cap<A> &>
00913 {
00914 using Svr_noops<Cap<A> &>::from_svr;
00915 static int from_svr(char *msg, unsigned offset, unsigned limit, long,
00916 Cap<A> arg, Dir_out, Cls_item) noexcept
00917 {
00918 if (L4_UNLIKELY(!arg.is_valid()))
00919 // do not map anything
00920 return msg_add(msg, offset, limit, L4::Ipc::Snd_fpage(arg.cap(), 0));
00921
00922 return msg_add(msg, offset, limit, arg.fpage());
00923 }
00924 };
00925
00926 // prohibit a UTCB pointer as normal RPC argument
00927 template<> struct Is_valid_rpc_type<l4_utcb_t *> : L4::Types::False {};
00928
00929 } // namespace Msg
00930 } // namespace Ipc
00931 } // namespace L4

```

## 17.500 ipc\_varg

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2014 Alexander Warg <alexander.warg@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008 #pragma GCC system_header
00009
00010 #include "types"
00011 #include "ipc_basics"
00012

```

```

00013 namespace L4 { namespace Ipc L4_EXPORT {
00014
00015 template< typename T, template <typename X> class B >
00016 struct Generic_va_type : B<T>
00017 {
00018 enum { Id = B<T>::Id };
00019 typedef B<T> ID;
00020 typedef T const &Ret_value;
00021 typedef T Value;
00022
00023 static Ret_value value(void const *d)
00024 { return *reinterpret_cast<Value const *>(d); }
00025
00026 static void const *addr_of(Value const &v) { return &v; }
00027
00028 static unsigned size(void const *) { return sizeof(T); }
00029
00030 static L4_varg_type unsigned_id()
00031 {
00032 return static_cast<L4_varg_type>(Id & ~L4_VARG_TYPE_SIGN);
00033 }
00034
00035 static L4_varg_type signed_id()
00036 {
00037 return static_cast<L4_varg_type>(Id | L4_VARG_TYPE_SIGN);
00038 }
00039
00040 static L4_varg_type id()
00041 {
00042 return static_cast<L4_varg_type>(Id);
00043 }
00044 };
00045
00046 template< typename T > struct Va_type_id;
00047 template<> struct Va_type_id<l4_umword_t> { enum { Id = L4_VARG_TYPE_UMWORD }; };
00048 template<> struct Va_type_id<l4_mword_t> { enum { Id = L4_VARG_TYPE_MWORD }; };
00049 template<> struct Va_type_id<l4_fpage_t> { enum { Id = L4_VARG_TYPE_FPAGE }; };
00050 template<> struct Va_type_id<void> { enum { Id = L4_VARG_TYPE_NIL }; };
00051 template<> struct Va_type_id<char const *> { enum { Id = L4_VARG_TYPE_STRING }; };
00052
00053 template< typename T > struct Va_type;
00054
00055 template<> struct Va_type<l4_umword_t> : Generic_va_type<l4_umword_t, Va_type_id> {};
00056 template<> struct Va_type<l4_mword_t> : Generic_va_type<l4_mword_t, Va_type_id> {};
00057 template<> struct Va_type<l4_fpage_t> : Generic_va_type<l4_fpage_t, Va_type_id> {};
00058
00059 template<> struct Va_type<void>
00060 {
00061 typedef void Ret_value;
00062 typedef void Value;
00063
00064 static void const *addr_of(void) { return 0; }
00065
00066 static void value(void const *) {}
00067 static L4_varg_type id() { return L4_VARG_TYPE_NIL; }
00068 static unsigned size(void const *) { return 0; }
00069 };
00070
00071 template<> struct Va_type<char const *>
00072 {
00073 typedef char const *Ret_value;
00074 typedef char const *Value;
00075
00076 static void const *addr_of(Value v) { return v; }
00077
00078 static L4_varg_type id() { return L4_VARG_TYPE_STRING; }
00079 static unsigned size(void const *s)
00080 {
00081 char const *_s = reinterpret_cast<char const *>(s);
00082 int l = 1;
00083 while (*_s)
00084 {
00085 ++_s; ++l;
00086 }
00087 return l;
00088 }
00089
00090 static Ret_value value(void const *d) { return static_cast<char const *>(d); }
00091 };
00092
00093 class Varg
00094 {
00095 private:
00096 enum { Direct_data = 0x8000 };
00097 l4_umword_t _tag;
00098 char const *_d;
00099
00100
00101
00102

```

```

00103 public:
00104
00106 typedef l4_umword_t Tag;
00107
00109 L4_varg_type type() const { return static_cast<L4_varg_type>(_tag & 0xff); }
00114 unsigned length() const { return _tag >> 16; }
00116 Tag tag() const { return _tag & ~Direct_data; }
00118 void tag(Tag tag) { _tag = tag; }
00120 void data(char const *d) { _d = d; }
00121
00123 char const *data() const
00124 {
00125 if (_tag & Direct_data)
00126 {
00127 union T { char const *d; char v[sizeof(char const *)]; };
00128 return reinterpret_cast<T const *>(&_d)->v;
00129 }
00130 return _d;
00131 }
00132
00134 #if __cplusplus >= 201103L
00135 Varg() = default;
00136 #else
00137 Varg() {}
00138 #endif
00139
00141 Varg(L4_varg_type t, void const *v, int len)
00142 : _tag(t | (static_cast<l4_mword_t>(len) << 16)),
00143 _d(static_cast<char const *>(v))
00144 {}
00145
00146 static Varg nil() { return Varg(L4_VARG_TYPE_NIL, 0, 0); }
00147
00154 template< typename V >
00155 typename Va_type<V>::Ret_value value() const
00156 {
00157 if (_tag & Direct_data)
00158 {
00159 union X { char const *d; V v; };
00160 return reinterpret_cast<X const *>(&_d).v;
00161 }
00162
00163 return Va_type<V>::value(_d);
00164 }
00165
00166
00168 template< typename T >
00169 bool is_of() const { return Va_type<T>::id() == type(); }
00170
00172 bool is_nil() const { return is_of<void>(); }
00173
00175 bool is_of_int() const
00176 { return (type() & ~L4_VARG_TYPE_SIGN) == L4_VARG_TYPE_UMWORD; }
00177
00184 template< typename T >
00185 bool get_value(typename Va_type<T>::Value *v) const
00186 {
00187 if (!is_of<T>())
00188 return false;
00189
00190 *v = this->value<T>();
00191 return true;
00192 }
00193
00195 template< typename T >
00196 void set_value(void const *d)
00197 {
00198 typedef Va_type<T> Vt;
00199 _tag = Vt::id() | (Vt::size(d) << 16);
00200 _d = static_cast<char const *>(d);
00201 }
00202
00204 template<typename T>
00205 void set_direct_value(T val, typename L4::Types::Enable_if<sizeof(T) <= sizeof(char const *)>,
00206 bool>::type = true)
00207 {
00208 static_assert(sizeof(T) <= sizeof(char const *), "direct Varg value too big");
00209 typedef Va_type<T> Vt;
00210 _tag = Vt::id() | (sizeof(T) << 16) | Direct_data;
00211 union X { char const *d; T v; };
00212 reinterpret_cast<X *>(&_d).v = val;
00213 }
00215 template<typename T> explicit
00216 Varg(T const *data) { set_value<T>(data); }
00218 Varg(char const *data) { set_value<char const *>(data); }
00219

```

```

00221 template<typename T> explicit
00222 Varg(T data, typename L4::Types::Enable_if<sizeof(T) <= sizeof(char const *)>, bool>::type = true)
00223 { set_direct_value<T>(data); }
00224 };
00225
00226
00227 template<typename T>
00228 class Varg_t : public Varg
00229 {
00230 public:
00231 typedef typename Va_type<T>::Value Value;
00232 explicit Varg_t(Value v) : Varg()
00233 { _data = v; set_value<T>(Va_type<T>::addr_of(_data)); }
00234
00235 private:
00236 Value _data;
00237 };
00238
00239 template<unsigned MAX = L4_UTCB_GENERIC_DATA_SIZE>
00240 class Varg_list;
00241
00253 class Varg_list_ref
00254 {
00255 private:
00256 template<unsigned T>
00257 friend class Varg_list;
00258
00260 class Iter_state
00261 {
00262 private:
00263 using M = l4_umword_t;
00264 using Mp = M const *;
00265 Mp _c;
00266 Mp _e;
00267
00269 Mp next_arg(Varg const &a) const
00270 {
00271 return _c + 1 + (Msg::align_to<M>(a.length()) / sizeof(M));
00272 }
00273
00274 public:
00276 Iter_state() : _c(nullptr), _e(nullptr) {}
00277
00279 Iter_state(Mp c, Mp e) : _c(c), _e(e)
00280 {}
00281
00283 bool valid() const
00284 { return _c && _c < _e; }
00285
00287 Mp begin() const { return _c; }
00288
00290 Mp end() const { return _e; }
00291
00296 Varg pop()
00297 {
00298 if (!valid())
00299 return Varg::nil();
00300
00301 Varg a;
00302 a.tag(_c[0]);
00303 a.data(reinterpret_cast<char const *>(&_c[1]));
00304 _c = next_arg(a);
00305 if (_c > _e)
00306 return Varg::nil();
00307
00308 return a;
00309 }
00310
00312 bool operator == (Iter_state const &o) const
00313 { return _c == o._c; }
00314
00316 bool operator != (Iter_state const &o) const
00317 { return _c != o._c; }
00318 };
00319
00320 Iter_state _s;
00321
00322 public:
00324 Varg_list_ref() = default;
00325
00332 Varg_list_ref(void const *start, void const *end)
00333 : _s(reinterpret_cast<l4_umword_t const *>(start),
00334 reinterpret_cast<l4_umword_t const *>(end))
00335 {}
00336
00338 class Iterator
00339 {

```



```

00340 private:
00341 Iter_state _s;
00342 Varg _a;
00343
00344 public:
00346 Iterator(Iter_state const &s)
00347 : _s(s)
00348 {
00349 _a = _s.pop();
00350 }
00351
00353 explicit operator bool () const
00354 { return !_a.is_nil(); }
00355
00357 Iterator &operator ++ ()
00358 {
00359 if (!_a.is_nil())
00360 _a = _s.pop();
00361
00362 return *this;
00363 }
00364
00366 Varg operator * () const
00367 { return _a; }
00368
00370 bool equals(Iterator const &o) const
00371 {
00372 if (_a.is_nil() && o._a.is_nil())
00373 return true;
00374
00375 return _s == o._s;
00376 }
00377
00378 bool operator == (Iterator const &o) const
00379 { return equals(o); }
00380
00381 bool operator != (Iterator const &o) const
00382 { return !equals(o); }
00383 };
00384
00386 Varg pop_front()
00387 { return _s.pop(); }
00388
00390 Varg next()
00391 { L4_DEPRECATED("Use range for or pop_front."); }
00392 { return _s.pop(); }
00393
00395 Iterator begin() const
00396 { return Iterator(_s); }
00397
00399 Iterator end() const
00400 { return Iterator(Iter_state()); }
00401 };
00402
00410 template<unsigned MAX>
00411 class Varg_list : public Varg_list_ref
00412 {
00413 l4_umword_t data[MAX];
00414 Varg_list(Varg_list const &);
00415
00416 public:
00418 Varg_list(Varg_list_ref const &r)
00419 {
00420 if (!r._s.valid())
00421 return;
00422
00423 l4_umword_t const *rs = r._s.begin();
00424 unsigned c = r._s.end() - rs;
00425 for (unsigned i = 0; i < c; ++i)
00426 data[i] = rs[i];
00427
00428 this->_s = Iter_state(data, data + c);
00429 }
00430 };
00431
00432 namespace Msg {
00433 template<> struct Elem<Varg const *>
00434 {
00435 typedef Varg const *arg_type;
00436 typedef Varg_list_ref svr_type;
00437 typedef Varg_list_ref svr_arg_type;
00438 enum { Is_optional = false };
00439 };
00440 };
00441
00442 template<> struct Is_valid_rpc_type<Varg> : L4::Types::False {};
00443 template<> struct Is_valid_rpc_type<Varg *> : L4::Types::False {};

```

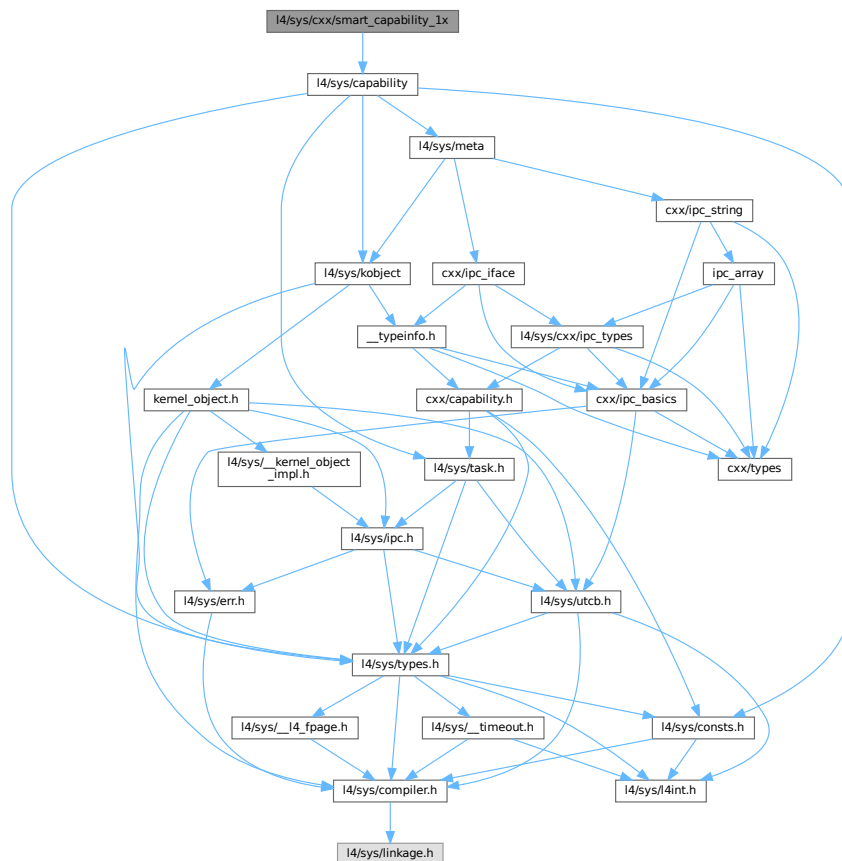
```

00444 template<> struct Is_valid_rpc_type<Varg &> : L4::Types::False {};
00445 template<> struct Is_valid_rpc_type<Varg const &> : L4::Types::False {};
00446
00447 template<> struct Direction<Varg const *> : Dir_in {};
00448 template<> struct Class<Varg const *> : Cls_data {};
00449
00450 template<typename DIR, typename CLASS>
00451 struct Clnt_val_ops<Varg, DIR, CLASS>;
00452
00453 template<>
00454 struct Clnt_val_ops<Varg, Dir_in, Cls_data> :
00455 Clnt_noops<Varg const &>
00456 {
00457 using Clnt_noops<Varg const &>::to_msg;
00458 static int to_msg(char *msg, unsigned offs, unsigned limit,
00459 Varg const &a, Dir_in, Cls_data)
00460 {
00461 for (Varg const *i = &a; i->tag(); ++i)
00462 {
00463 offs = align_to<l4_umword_t>(offs);
00464 if (L4_UNLIKELY(!check_size<l4_umword_t>(offs, limit)))
00465 return -L4_MSGTOOLONG;
00466 *reinterpret_cast<l4_umword_t*>(msg + offs) = i->tag();
00467 offs += sizeof(l4_umword_t);
00468 if (L4_UNLIKELY(!check_size<char>(offs, limit, i->length())))
00469 return -L4_MSGTOOLONG;
00470 char const *d = i->data();
00471 for (unsigned x = 0; x < i->length(); ++x)
00472 msg[offs++] = *d++;
00473 }
00474 return offs;
00475 }
00476 };
00477
00478 template<>
00479 struct Svr_val_ops<Varg_list_ref, Dir_in, Cls_data> :
00480 Svr_noops<Varg_list_ref>
00481 {
00482 using Svr_noops<Varg_list_ref>::to_svr;
00483 static int to_svr(char *msg, unsigned offset, unsigned limit,
00484 Varg_list_ref &a, Dir_in, Cls_data)
00485 {
00486 unsigned start = align_to<l4_umword_t>(offset);
00487 unsigned offs;
00488 for (offs = start; offs < limit;)
00489 {
00490 unsigned noffs = align_to<l4_umword_t>(offs);
00491 if (L4_UNLIKELY(!check_size<l4_umword_t>(noffs, limit)))
00492 break;
00493 offs = noffs;
00494 Varg arg;
00495 arg.tag(*reinterpret_cast<l4_umword_t*>(msg + offs));
00496 if (!arg.tag())
00497 break;
00498 offs += sizeof(l4_umword_t);
00499 if (L4_UNLIKELY(!check_size<char>(offs, limit, arg.length())))
00500 return -L4_MSGTOOLONG;
00501 offs += arg.length();
00502 }
00503 a = Varg_list_ref(msg + start, msg + align_to<l4_umword_t>(offs));
00504 return offs;
00505 }
00506 };
00507
00508
00509
00510
00511 }
00512 };
00513 }
00514 }

```

```
#include <linux/sys/capability>
```

Include dependency graph for smart\_capability\_1x:



- namespace **L4**  
*L4 low-level kernel interface.*

## 17.502 smart\_capability\_1x

[Go to the documentation of this file.](#)

```

00001 // vim:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2017 Alexander Warg <alexander.warg@kernkonzept.com>
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011
00012 #pragma once
00013
00014 #include <l4/sys/capability>
00015
00016 namespace L4 { namespace Detail {
00017
00018 template< typename T, typename IMPL >
00019 class Smart_cap_base : public Cap_base, protected IMPL
00020 {
00021 protected:
00022 template<typename X>
00023 static IMPL &impl(Smart_cap_base<X, IMPL> &o) { return o; }
00024
00025 template<typename X>
00026 static IMPL const &impl(Smart_cap_base<X, IMPL> const &o) { return o; }
00027
00028 public:
00029 template<typename X, typename I>
00030 friend class ::L4::Detail::Smart_cap_base;
00031
00032 Smart_cap_base(Smart_cap_base const &) = delete;
00033 Smart_cap_base &operator = (Smart_cap_base const &) = delete;
00034
00035 Smart_cap_base() noexcept : Cap_base(Invalid) {}
00036
00037 explicit Smart_cap_base(Cap_base::Cap_type t) noexcept
00038 : Cap_base(t)
00039 {}
00040
00041 template<typename O>
00042 explicit constexpr Smart_cap_base(Cap<O> c) noexcept
00043 : Cap_base(c.cap())
00044 {}
00045
00046 template<typename O>
00047 explicit constexpr Smart_cap_base(Cap<O> c, IMPL const &impl) noexcept
00048 : Cap_base(c.cap()), IMPL(impl)
00049 {}
00050
00051 Cap<T> release() noexcept
00052 {
00053 l4_cap_idx_t c = this->cap();
00054 IMPL::invalidate(*this);
00055 return Cap<T>(c);
00056 }
00057
00058 void reset()
00059 { IMPL::free(*this); }
00060
00061 Cap<T> operator -> () const noexcept { return Cap<T>(this->cap()); }
00062 Cap<T> get() const noexcept { return Cap<T>(this->cap()); }
00063 ~Smart_cap_base() noexcept { IMPL::free(*this); }
00064 };
00065
00066
00067 template< typename T, typename IMPL >
00068 class Unique_cap_impl final : public Smart_cap_base<T, IMPL>
00069 {
00070 private:
00071 typedef Smart_cap_base<T, IMPL> Base;
00072
00073 public:
00074 using Base::Base;
00075 Unique_cap_impl() noexcept = default;
00076
00077 Unique_cap_impl(Unique_cap_impl &&o) noexcept
00078 : Base(o.release(), Base::impl(o))
00079 {}
00080
00081 template<typename O>
00082 Unique_cap_impl(Unique_cap_impl<O, IMPL> &&o) noexcept
00083 : Base(o.release(), Base::impl(o))
00084 { Cap<T>::template check_convertible_from<O>(); }
00085
00086 Unique_cap_impl &operator = (Unique_cap_impl &&o) noexcept

```

```

00087 {
00088 if (&o == this)
00089 return *this;
00090
00091 IMPL::free(*this);
00092 this->_c = o.release().cap();
00093 this->IMPL::operator = (Base::impl(o));
00094 return *this;
00095 }
00096
00097 template<typename O>
00098 Unique_cap_impl &operator = (Unique_cap_impl<O, IMPL> &o) noexcept
00099 {
00100 Cap<T>::template check_convertible_from<O>();
00101
00102 IMPL::free(*this);
00103 this->_c = o.release().cap();
00104 this->IMPL::operator = (Base::impl(o));
00105 return *this;
00106 }
00107 };
00108
00109 template<typename T, typename IMPL>
00110 class Shared_cap_impl final : public Smart_cap_base<T, IMPL>
00111 {
00112 private:
00113 typedef Smart_cap_base<T, IMPL> Base;
00114
00115 public:
00116 using Base::Base;
00117 Shared_cap_impl() noexcept = default;
00118
00119 Shared_cap_impl(Shared_cap_impl &o) noexcept
00120 : Base(o.release(), Base::impl(o))
00121 {}
00122
00123 template<typename O>
00124 Shared_cap_impl(Shared_cap_impl<O, IMPL> &o) noexcept
00125 : Base(o.release(), Base::impl(o))
00126 { Cap<T>::template check_convertible_from<O>(); }
00127
00128 Shared_cap_impl &operator = (Shared_cap_impl &o) noexcept
00129 {
00130 if (&o == this)
00131 return *this;
00132
00133 IMPL::free(*this);
00134 this->_c = o.release().cap();
00135 this->IMPL::operator = (Base::impl(o));
00136 return *this;
00137 }
00138
00139 template<typename O>
00140 Shared_cap_impl &operator = (Shared_cap_impl<O, IMPL> &o) noexcept
00141 {
00142 Cap<T>::template check_convertible_from<O>();
00143
00144 IMPL::free(*this);
00145 this->_c = o.release().cap();
00146 this->IMPL::operator = (Base::impl(o));
00147 return *this;
00148 }
00149
00150 Shared_cap_impl(Shared_cap_impl const &o) noexcept
00151 : Base()
00152 {
00153 this->IMPL::operator = (Base::impl(o));
00154 this->_c = IMPL::copy(o).cap();
00155 }
00156
00157 template<typename O>
00158 Shared_cap_impl(Shared_cap_impl<O, IMPL> const &o) noexcept
00159 : Base()
00160 {
00161 Cap<T>::template check_convertible_from<O>();
00162 this->IMPL::operator = (Base::impl(o));
00163 this->_c = IMPL::copy(o).cap();
00164 }
00165
00166 Shared_cap_impl &operator = (Shared_cap_impl const &o) noexcept
00167 {
00168 if (&o == this)
00169 return *this;
00170
00171 IMPL::free(*this);
00172 this->IMPL::operator = (static_cast<IMPL const &>(o));
00173 this->_c = this->IMPL::copy(o).cap();

```

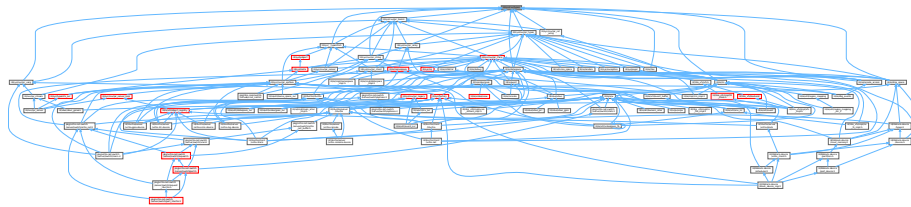
```

00174 return *this;
00175 }
00176
00177 template<typename O>
00178 Shared_cap_impl &operator = (Shared_cap_impl<O, IMPL> const &o) noexcept
00179 {
00180 Cap<T>::template check_convertible_from<O>();
00181 IMPL::free(*this);
00182 this->IMPL::operator = (static_cast<IMPL const &>(o));
00183 this->_c = this->IMPL::copy(o).cap();
00184 return *this;
00185 }
00186 };
00187
00188 }} // L4::Detail

```

## 17.503 I4/sys/cxx/types File Reference

This graph shows which files directly or indirectly include this file:



### Data Structures

- class [L4::Types::Flags< BITS\\_ENUM, UNDERLYING >](#)  
*Template for defining typical [Flags](#) bitmaps.*
- struct [L4::Types::Int\\_for\\_type< T >](#)  
*Metafunction to get an integral type of the same size as *T*.*
- struct [L4::Types::Flags\\_ops\\_t< DT >](#)  
*Mixin class to define a set of friend bitwise operators on *DT*.*
- struct [L4::Types::Flags\\_t< DT, T >](#)  
*Template type to define a flags type with bitwise operations.*
- struct [L4::Types::Bool< V >](#)  
*Boolean meta type.*
- struct [L4::Types::False](#)  
*[False](#) meta value.*
- struct [L4::Types::True](#)  
*[True](#) meta value.*
- struct [L4::Types::Same< A, B >](#)  
*Compare two data types for equality.*

### Namespaces

- namespace [L4](#)  
*[L4](#) low-level kernel interface.*
- namespace [L4::Types](#)  
*[L4](#) basic type helpers for C++.*

## Macros

- `#define L4_TYPES_FLAGS_OPS_DEF(T)`  
*Helper macro to define a set of bitwise operators on an enum type.*

### 17.503.1 Macro Definition Documentation

#### 17.503.1.1 L4\_TYPES\_FLAGS\_OPS\_DEF

```
#define L4_TYPES_FLAGS_OPS_DEF(
 T)
```

#### Value:

```
friend constexpr T operator ~ (T f)
{
 return T(~static_cast<typename L4::Types::Int_for_type<T>::type>(f));
}

friend constexpr T operator | (T l, T r)
{
 return T(static_cast<typename L4::Types::Int_for_type<T>::type>(l)
 | static_cast<typename L4::Types::Int_for_type<T>::type>(r));
}

friend constexpr T operator & (T l, T r)
{
 return T(static_cast<typename L4::Types::Int_for_type<T>::type>(l)
 & static_cast<typename L4::Types::Int_for_type<T>::type>(r));
}
```

Helper macro to define a set of bitwise operators on an enum type.

This allows to use the enum type as bitmask type with '&', '|', and '~' operators that keep the enum type as result.

Definition at line 195 of file [types](#).

## 17.504 types

[Go to the documentation of this file.](#)

```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2014 Alexander Warg <alexander.warg@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008
00009
00010 #pragma once
00011
00012 // very simple type traits for basic L4 functions, for a more complete set
00013 // use <l4/cxx/type_traits> or the standard <type_traits>.
00014
00015 namespace L4 {
00016
00020 namespace Types {
00021
00051 template<typename BITS_ENUM, typename UNDERLYING = unsigned long>
00052 class Flags
00053 {
00054 public:
00056 typedef UNDERLYING value_type;
00058 typedef BITS_ENUM bits_enum_type;
00060 typedef Flags<BITS_ENUM, UNDERLYING> type;
00061
00062 private:
00063 value_type _v;
```

```

00064 explicit Flags(value_type v) : _v(v) {}
00065
00066 public:
00067 enum None_type { None };
00068
00069 Flags(None_type) : _v(0) {}
00070
00071 Flags() : _v(0) {}
00072
00073 Flags(BITS_ENUM e) : _v((value_type{1}) << e) {}
00074
00075 static type from_raw(value_type v) { return type(v); }
00076
00077 explicit operator bool () const
00078 { return _v != 0; }
00079
00080 bool operator ! () const { return _v == 0; }
00081
00082 friend type operator | (type lhs, type rhs)
00083 { return type(lhs._v | rhs._v); }
00084
00085 friend type operator | (type lhs, bits_enum_type rhs)
00086 { return lhs | type(rhs); }
00087
00088 friend type operator & (type lhs, type rhs)
00089 { return type(lhs._v & rhs._v); }
00090
00091 friend type operator & (type lhs, bits_enum_type rhs)
00092 { return lhs & type(rhs); }
00093
00094 type &operator |= (type rhs) { _v |= rhs._v; return *this; }
00095 type &operator |= (bits_enum_type rhs) { return operator |= (type(rhs)); }
00096
00097 type &operator &= (type rhs) { _v &= rhs._v; return *this; }
00098 type &operator &= (bits_enum_type rhs) { return operator &= (type(rhs)); }
00099
00100 type operator ~ () const { return type(~_v); }
00101
00102 type &clear(bits_enum_type flag) { return operator &= (~type(flag)); }
00103
00104 value_type as_value() const { return _v; }
00105 };
00106
00107 template<unsigned SIZE, bool = true> struct Int_for_size;
00108
00109 template<> struct Int_for_size<sizeof(unsigned char), true>
00110 { typedef unsigned char type; };
00111
00112 template<> struct Int_for_size<sizeof(unsigned short),
00113 (sizeof(unsigned short) > sizeof(unsigned char))>
00114 { typedef unsigned short type; };
00115
00116 template<> struct Int_for_size<sizeof(unsigned),
00117 (sizeof(unsigned) > sizeof(unsigned short))>
00118 { typedef unsigned type; };
00119
00120 template<> struct Int_for_size<sizeof(unsigned long),
00121 (sizeof(unsigned long) > sizeof(unsigned))>
00122 { typedef unsigned long type; };
00123
00124 template<> struct Int_for_size<sizeof(unsigned long long),
00125 (sizeof(unsigned long long) > sizeof(unsigned long))>
00126 { typedef unsigned long long type; };
00127
00128 template<typename T> struct Int_for_type
00129 {
00130 typedef typename Int_for_size<sizeof(T)>::type type;
00131 };
00132
00133 #define L4_TYPES_FLAGS_OPS_DEF(T)
00134 friend constexpr T operator ~ (T f)
00135 {
00136 return T(~static_cast<typename L4::Types::Int_for_type<T>::type>(f));
00137 }
00138
00139 friend constexpr T operator | (T l, T r)
00140 {
00141 return T(static_cast<typename L4::Types::Int_for_type<T>::type>(l)
00142 | static_cast<typename L4::Types::Int_for_type<T>::type>(r));
00143 }
00144
00145 friend constexpr T operator & (T l, T r)
00146 {
00147 return T(static_cast<typename L4::Types::Int_for_type<T>::type>(l)
00148 & static_cast<typename L4::Types::Int_for_type<T>::type>(r));
00149 }
00150
00151

```



```

00219 template<typename DT>
00220 struct Flags_ops_t
00221 {
00223 friend constexpr DT operator | (DT l, DT r)
00224 { return DT(l.raw | r.raw); }
00225
00227 friend constexpr DT operator & (DT l, DT r)
00228 { return DT(l.raw & r.raw); }
00229
00231 friend constexpr bool operator == (DT l, DT r)
00232 { return l.raw == r.raw; }
00233
00235 friend constexpr bool operator != (DT l, DT r)
00236 { return l.raw != r.raw; }
00237
00239 DT operator |= (DT r)
00240 {
00241 static_cast<DT *>(this)->raw |= r.raw;
00242 return *static_cast<DT *>(this);
00243 }
00244
00246 DT operator &= (DT r)
00247 {
00248 static_cast<DT *>(this)->raw &= r.raw;
00249 return *static_cast<DT *>(this);
00250 }
00251
00253 explicit constexpr operator bool () const
00254 {
00255 return static_cast<DT const *>(this)->raw != 0;
00256 }
00257
00259 constexpr DT operator ~ () const
00260 { return DT(~static_cast<DT const *>(this)->raw); }
00261 };
00262
00271 template<typename DT, typename T>
00272 struct Flags_t : Flags_ops_t<Flags_t<DT, T>
00273 {
00275 T raw;
00277 Flags_t() = default;
00279 explicit constexpr Flags_t(T f) : raw(f) {}
00280 };
00281
00282
00288 template< bool V > struct Bool
00289 {
00290 typedef Bool<V> type;
00291 enum { value = V };
00292 };
00293
00296 struct False : Bool<false> {};
00297
00300 struct True : Bool<true> {};
00301
00302 /*****/
00311 template<typename A, typename B>
00312 struct Same : False {};
00313
00314 template<typename A>
00315 struct Same<A, A> : True {};
00316
00317 template<bool EXP, typename T = void> struct Enable_if {};
00318 template<typename T> struct Enable_if<true, T> { typedef T type; };
00319
00320 template<typename T1, typename T2, typename T = void>
00321 struct Enable_if_same : Enable_if<Same<T1, T2>::value, T> {};
00322
00323 template<typename T> struct Remove_const { typedef T type; };
00324 template<typename T> struct Remove_const<T const> { typedef T type; };
00325 template<typename T> struct Remove_volatile { typedef T type; };
00326 template<typename T> struct Remove_volatile<T volatile> { typedef T type; };
00327 template<typename T> struct Remove_cv
00328 { typedef typename Remove_const<typename Remove_volatile<T>::type>::type type; };
00329
00330 template<typename T> struct Remove_pointer { typedef T type; };
00331 template<typename T> struct Remove_pointer<T*> { typedef T type; };
00332 template<typename T> struct Remove_reference { typedef T type; };
00333 template<typename T> struct Remove_reference<T&> { typedef T type; };
00334 template<typename T> struct Remove_pr { typedef T type; };
00335 template<typename T> struct Remove_pr<T&> { typedef T type; };
00336 template<typename T> struct Remove_pr<T*> { typedef T type; };
00337 } // Types
00338 } // L4

```

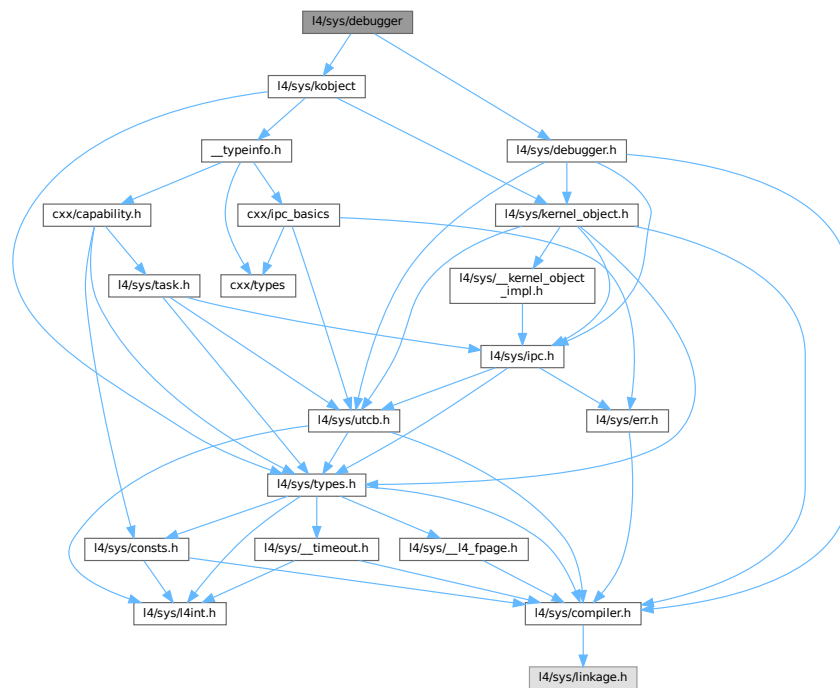
## 17.505 I4/sys/debugger File Reference

The debugger interface specifies common debugging related definitions.

```
#include <l4/sys/debugger.h>
```

```
#include <l4/sys/kobject>
```

Include dependency graph for debugger:



### Data Structures

- class [L4::Debugger](#)  
*C++ kernel debugger API.*

### Namespaces

- namespace [L4](#)  
*[L4](#) low-level kernel interface.*

### 17.505.1 Detailed Description

The debugger interface specifies common debugging related definitions.

Definition in file [debugger](#).

## 17.506 debugger

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2010-2011 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 #pragma once
00010
00011 #include <l4/sys/debugger.h>
00012 #include <l4/sys/kobject>
00013
00014 namespace L4 {
00015
00016 class Debugger : public Kobject_t<Debugger, Kobject, L4_PROTO_DEBUGGER>
00017 {
00018 public:
00019 enum
00020 {
00021 Switch_log_on = L4_DEBUGGER_SWITCH_LOG_ON,
00022 Switch_log_off = L4_DEBUGGER_SWITCH_LOG_OFF,
00023 };
00024
00025 l4_msgtag_t set_object_name(const char *name,
00026 l4_utcb_t *utcb = l4_utcb()) noexcept
00027 { return l4_debugger_set_object_name_u(cap(), name, utcb); }
00028
00029 unsigned long global_id(l4_utcb_t *utcb = l4_utcb()) noexcept
00030 { return l4_debugger_global_id_u(cap(), utcb); }
00031
00032 unsigned long kobj_to_id(l4_addr_t kobjp,
00033 l4_utcb_t *utcb = l4_utcb()) noexcept
00034 { return l4_debugger_kobj_to_id_u(cap(), kobjp, utcb); }
00035
00036 long query_log_typeid(const char *name, unsigned idx,
00037 l4_utcb_t *utcb = l4_utcb()) noexcept
00038 { return l4_debugger_query_log_typeid_u(cap(), name, idx, utcb); }
00039
00040 long query_log_name(unsigned idx,
00041 char *name, unsigned namelen,
00042 char *shortname, unsigned shortnamelen,
00043 l4_utcb_t *utcb = l4_utcb()) noexcept
00044 { return l4_debugger_query_log_name_u(cap(), idx, name, namelen,
00045 shortname, shortnamelen, utcb); }
00046
00047 l4_msgtag_t switch_log(const char *name, unsigned on_off,
00048 l4_utcb_t *utcb = l4_utcb()) noexcept
00049 { return l4_debugger_switch_log_u(cap(), name, on_off, utcb); }
00050
00051 l4_msgtag_t get_object_name(unsigned id, char *name, unsigned size,
00052 l4_utcb_t *utcb = l4_utcb()) noexcept
00053 { return l4_debugger_get_object_name_u(cap(), id, name, size, utcb); }
00054
00055 l4_msgtag_t add_image_info(l4_addr_t base, const char *name,
00056 l4_utcb_t *utcb = l4_utcb()) noexcept
00057 { return l4_debugger_add_image_info_u(cap(), base, name, utcb); }
00058 };
00059
00060 }
```

## 17.507 l4/sys/debugger.h File Reference

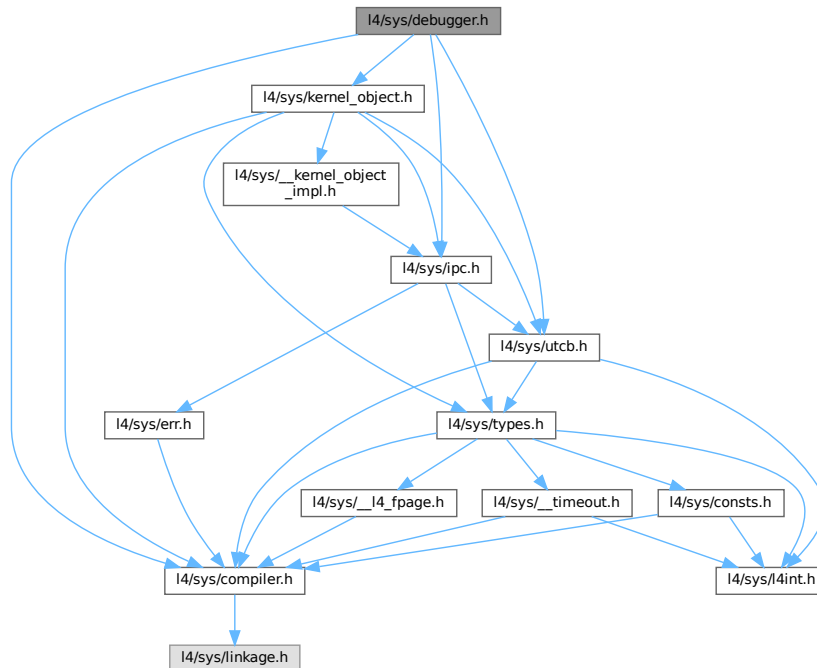
Debugger related definitions.

```

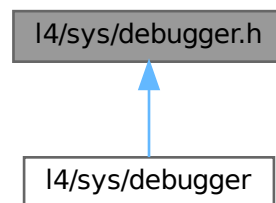
#include <l4/sys/compiler.h>
#include <l4/sys/utcb.h>
#include <l4/sys/ipc.h>
```

```
#include <l4/sys/kernel_object.h>
```

Include dependency graph for debugger.h:



This graph shows which files directly or indirectly include this file:



## Functions

- [l4\\_msgtag\\_t l4\\_debugger\\_set\\_object\\_name](#) ([l4\\_cap\\_idx\\_t](#) cap, const char \*name) [L4\\_NOTHROW](#)  
Set the name of a kernel object.
- [l4\\_msgtag\\_t l4\\_debugger\\_get\\_object\\_name](#) ([l4\\_cap\\_idx\\_t](#) cap, unsigned id, char \*name, unsigned size) [L4\\_NOTHROW](#)  
Get name of the kernel object with Id id.
- unsigned long [l4\\_debugger\\_global\\_id](#) ([l4\\_cap\\_idx\\_t](#) cap) [L4\\_NOTHROW](#)  
Get the globally unique ID of the object behind a capability.

- unsigned long [l4\\_debugger\\_kobj\\_to\\_id](#) ([l4\\_cap\\_idx\\_t](#) cap, [l4\\_addr\\_t](#) kobjp) [L4\\_NOTHROW](#)  
*Get the globally unique ID of the object behind the kobject pointer.*
- long [l4\\_debugger\\_query\\_log\\_typeid](#) ([l4\\_cap\\_idx\\_t](#) cap, const char \*name, unsigned idx) [L4\\_NOTHROW](#)  
*Query the log-id for a log type.*
- long [l4\\_debugger\\_query\\_log\\_name](#) ([l4\\_cap\\_idx\\_t](#) cap, unsigned idx, char \*name, unsigned namelen, char \*shortname, unsigned shortnamelen) [L4\\_NOTHROW](#)  
*Query the name of a log type given the ID.*
- [l4\\_msgtag\\_t](#) [l4\\_debugger\\_switch\\_log](#) ([l4\\_cap\\_idx\\_t](#) cap, const char \*name, int on\_off) [L4\\_NOTHROW](#)  
*Set or unset log.*
- [l4\\_msgtag\\_t](#) [l4\\_debugger\\_add\\_image\\_info](#) ([l4\\_cap\\_idx\\_t](#) cap, [l4\\_addr\\_t](#) base, const char \*name) [L4\\_NOTHROW](#)  
*Add loaded image information for a task.*

## 17.507.1 Detailed Description

Debugger related definitions.

Definition in file [debugger.h](#).

## 17.508 debugger.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00007 /*
00008 * (c) 2008-2011 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00009 * Alexander Warg <warg@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014
00015 #include <l4/sys/compiler.h>
00016 #include <l4/sys/utcb.h>
00017 #include <l4/sys/ipc.h>
00018
00031
00042 L4_INLINE l4_msgtag_t
00043 l4_debugger_set_object_name(l4_cap_idx_t cap, const char *name) L4_NOTHROW;
00044
00048 L4_INLINE l4_msgtag_t
00049 l4_debugger_set_object_name_u(l4_cap_idx_t cap, const char *name, l4_utcb_t *utcb) L4_NOTHROW;
00050
00063 L4_INLINE l4_msgtag_t
00064 l4_debugger_get_object_name(l4_cap_idx_t cap, unsigned id,
00065 char *name, unsigned size) L4_NOTHROW;
00066
00070 L4_INLINE l4_msgtag_t
00071 l4_debugger_get_object_name_u(l4_cap_idx_t cap, unsigned id,
00072 char *name, unsigned size,
00073 l4_utcb_t *utcb) L4_NOTHROW;
00074
00086 L4_INLINE unsigned long
00087 l4_debugger_global_id(l4_cap_idx_t cap) L4_NOTHROW;
00088
00092 L4_INLINE unsigned long
00093 l4_debugger_global_id_u(l4_cap_idx_t cap, l4_utcb_t *utcb) L4_NOTHROW;
00094
00107 L4_INLINE unsigned long
00108 l4_debugger_kobj_to_id(l4_cap_idx_t cap, l4_addr_t kobjp) L4_NOTHROW;
00109
00113 L4_INLINE unsigned long
00114 l4_debugger_kobj_to_id_u(l4_cap_idx_t cap, l4_addr_t kobjp, l4_utcb_t *utcb) L4_NOTHROW;
00115
00128 L4_INLINE long
00129 l4_debugger_query_log_typeid(l4_cap_idx_t cap, const char *name,
00130 unsigned idx) L4_NOTHROW;
00131

```

```

00135 L4_INLINE long
00136 l4_debugger_query_log_typeid_u(l4_cap_idx_t cap, const char *name,
00137 unsigned idx, l4_utcb_t *utcb) L4_NOTHROW;
00138
00155 L4_INLINE long
00156 l4_debugger_query_log_name(l4_cap_idx_t cap, unsigned idx,
00157 char *name, unsigned namelen,
00158 char *shortname, unsigned shortnamelen) L4_NOTHROW;
00159
00163 L4_INLINE long
00164 l4_debugger_query_log_name_u(l4_cap_idx_t cap, unsigned idx,
00165 char *name, unsigned namelen,
00166 char *shortname, unsigned shortnamelen,
00167 l4_utcb_t *utcb) L4_NOTHROW;
00168
00179 L4_INLINE l4_msgtag_t
00180 l4_debugger_switch_log(l4_cap_idx_t cap, const char *name,
00181 int on_off) L4_NOTHROW;
00182
00186 L4_INLINE l4_msgtag_t
00187 l4_debugger_switch_log_u(l4_cap_idx_t cap, const char *name, int on_off,
00188 l4_utcb_t *utcb) L4_NOTHROW;
00189
00200 L4_INLINE l4_msgtag_t
00201 l4_debugger_add_image_info(l4_cap_idx_t cap, l4_addr_t base,
00202 const char *name) L4_NOTHROW;
00203
00207 L4_INLINE l4_msgtag_t
00208 l4_debugger_add_image_info_u(l4_cap_idx_t cap, l4_addr_t base, const char *name,
00209 l4_utcb_t *utcb) L4_NOTHROW;
00210
00211 enum
00212 {
00213 L4_DEBUGGER_NAME_SET_OP = 0UL,
00214 L4_DEBUGGER_GLOBAL_ID_OP = 1UL,
00215 L4_DEBUGGER_KOBJ_TO_ID_OP = 2UL,
00216 L4_DEBUGGER_QUERY_LOG_TYPEID_OP = 3UL,
00217 L4_DEBUGGER_SWITCH_LOG_OP = 4UL,
00218 L4_DEBUGGER_NAME_GET_OP = 5UL,
00219 L4_DEBUGGER_QUERY_LOG_NAME_OP = 6UL,
00220 L4_DEBUGGER_ADD_IMAGE_INFO_OP = 7UL,
00221 };
00222
00223 enum
00224 {
00225 L4_DEBUGGER_SWITCH_LOG_ON = 1,
00226 L4_DEBUGGER_SWITCH_LOG_OFF = 0,
00227 };
00228
00229 /* IMPLEMENTATION -----*/
00230
00231 #include <l4/sys/kernel_object.h>
00232
00246 L4_INLINE unsigned
00247 __strcpy_maxlen(char *dst, char const *src, unsigned maxlen)
00248 {
00249 unsigned i;
00250 if (!maxlen)
00251 return 0;
00252
00253 for (i = 0; i < maxlen - 1 && src[i]; ++i)
00254 dst[i] = src[i];
00255 dst[i] = '\0';
00256
00257 return i + 1;
00258 }
00259
00260 L4_INLINE l4_msgtag_t
00261 l4_debugger_set_object_name_u(l4_cap_idx_t cap,
00262 const char *name, l4_utcb_t *utcb) L4_NOTHROW
00263 {
00264 unsigned i;
00265 l4_utcb_mr_u(utcb)->mr[0] = L4_DEBUGGER_NAME_SET_OP;
00266 i = __strcpy_maxlen((char *)&l4_utcb_mr_u(utcb)->mr[1], name,
00267 (L4_UTCB_GENERIC_DATA_SIZE - 2) * sizeof(l4_umword_t));
00268 i = l4_bytes_to_mwords(i);
00269 return l4_invoke_debugger(cap, l4_msgtag(0, 1 + i, 0, 0), utcb);
00270 }
00271
00272 L4_INLINE unsigned long
00273 l4_debugger_global_id_u(l4_cap_idx_t cap, l4_utcb_t *utcb) L4_NOTHROW
00274 {
00275 l4_utcb_mr_u(utcb)->mr[0] = L4_DEBUGGER_GLOBAL_ID_OP;
00276 if (l4_error_u(l4_invoke_debugger(cap, l4_msgtag(0, 1, 0, 0), utcb), utcb))
00277 return ~0UL;
00278 return l4_utcb_mr_u(utcb)->mr[0];
00279 }

```

```

00280
00281 L4_INLINE unsigned long
00282 l4_debugger_kobj_to_id_u(l4_cap_idx_t cap, l4_addr_t kobjp, l4_utcb_t *utcb) L4_NOTHROW
00283 {
00284 l4_utcb_mr_u(utcb)->mr[0] = L4_DEBUGGER_KOBJ_TO_ID_OP;
00285 l4_utcb_mr_u(utcb)->mr[1] = kobjp;
00286 if (l4_error_u(l4_invoke_debugger(cap, l4_msgtag(0, 2, 0, 0), utcb), utcb))
00287 return ~0UL;
00288 return l4_utcb_mr_u(utcb)->mr[0];
00289 }
00290
00291 L4_INLINE long
00292 l4_debugger_query_log_typeid_u(l4_cap_idx_t cap, const char *name,
00293 unsigned idx,
00294 l4_utcb_t *utcb) L4_NOTHROW
00295 {
00296 unsigned i;
00297 long e;
00298 l4_utcb_mr_u(utcb)->mr[0] = L4_DEBUGGER_QUERY_LOG_TYPEID_OP;
00299 l4_utcb_mr_u(utcb)->mr[1] = idx;
00300 i = __strcpy_maxlen((char *)&l4_utcb_mr_u(utcb)->mr[2], name, 32);
00301 i = l4_bytes_to_mwords(i);
00302 e = l4_error_u(l4_invoke_debugger(cap, l4_msgtag(0, 2 + i, 0, 0), utcb), utcb);
00303 if (e < 0)
00304 return e;
00305 return l4_utcb_mr_u(utcb)->mr[0];
00306 }
00307
00308 L4_INLINE long
00309 l4_debugger_query_log_name_u(l4_cap_idx_t cap, unsigned idx,
00310 char *name, unsigned namelen,
00311 char *shortname, unsigned shortnamelen,
00312 l4_utcb_t *utcb) L4_NOTHROW
00313 {
00314 long e;
00315 char const *n;
00316 l4_utcb_mr_u(utcb)->mr[0] = L4_DEBUGGER_QUERY_LOG_NAME_OP;
00317 l4_utcb_mr_u(utcb)->mr[1] = idx;
00318 e = l4_error_u(l4_invoke_debugger(cap, l4_msgtag(0, 2, 0, 0), utcb), utcb);
00319 if (e < 0)
00320 return e;
00321 n = (char const *)&l4_utcb_mr_u(utcb)->mr[0];
00322 __strcpy_maxlen(name, n, namelen);
00323 __strcpy_maxlen(shortname, n + __builtin_strlen(n) + 1, shortnamelen);
00324 return 0;
00325 }
00326
00327
00328 L4_INLINE l4_msgtag_t
00329 l4_debugger_switch_log_u(l4_cap_idx_t cap, const char *name, int on_off,
00330 l4_utcb_t *utcb) L4_NOTHROW
00331 {
00332 unsigned i;
00333 l4_utcb_mr_u(utcb)->mr[0] = L4_DEBUGGER_SWITCH_LOG_OP;
00334 l4_utcb_mr_u(utcb)->mr[1] = on_off;
00335 i = __strcpy_maxlen((char *)&l4_utcb_mr_u(utcb)->mr[2], name, 32);
00336 i = l4_bytes_to_mwords(i);
00337 return l4_invoke_debugger(cap, l4_msgtag(0, 2 + i, 0, 0), utcb);
00338 }
00339
00340 L4_INLINE l4_msgtag_t
00341 l4_debugger_get_object_name_u(l4_cap_idx_t cap, unsigned id,
00342 char *name, unsigned size,
00343 l4_utcb_t *utcb) L4_NOTHROW
00344 {
00345 l4_msgtag_t t;
00346 l4_utcb_mr_u(utcb)->mr[0] = L4_DEBUGGER_NAME_GET_OP;
00347 l4_utcb_mr_u(utcb)->mr[1] = id;
00348 t = l4_invoke_debugger(cap, l4_msgtag(0, 2, 0, 0), utcb);
00349 __strcpy_maxlen(name, (char const *)&l4_utcb_mr_u(utcb)->mr[0], size);
00350 return t;
00351 }
00352
00353 L4_INLINE l4_msgtag_t
00354 l4_debugger_add_image_info_u(l4_cap_idx_t cap, l4_addr_t base,
00355 const char *name, l4_utcb_t *utcb) L4_NOTHROW
00356 {
00357 unsigned i;
00358 l4_utcb_mr_u(utcb)->mr[0] = L4_DEBUGGER_ADD_IMAGE_INFO_OP;
00359 l4_utcb_mr_u(utcb)->mr[1] = base;
00360 i = __strcpy_maxlen((char *)&l4_utcb_mr_u(utcb)->mr[2], name,
00361 (L4_UTCB_GENERIC_DATA_SIZE - 3) * sizeof(l4_umword_t));
00362 i = l4_bytes_to_mwords(i);
00363 return l4_invoke_debugger(cap, l4_msgtag(0, 2 + i, 0, 0), utcb);
00364 }
00365
00366

```

```

00367 L4_INLINE l4_msgtag_t
00368 l4_debugger_set_object_name(l4_cap_idx_t cap,
00369 const char *name) L4_NOTHROW
00370 {
00371 return l4_debugger_set_object_name_u(cap, name, l4_utcb());
00372 }
00373
00374 L4_INLINE unsigned long
00375 l4_debugger_global_id(l4_cap_idx_t cap) L4_NOTHROW
00376 {
00377 return l4_debugger_global_id_u(cap, l4_utcb());
00378 }
00379
00380 L4_INLINE unsigned long
00381 l4_debugger_kobj_to_id(l4_cap_idx_t cap, l4_addr_t kobjp) L4_NOTHROW
00382 {
00383 return l4_debugger_kobj_to_id_u(cap, kobjp, l4_utcb());
00384 }
00385
00386 L4_INLINE long
00387 l4_debugger_query_log_typeid(l4_cap_idx_t cap, const char *name,
00388 unsigned idx) L4_NOTHROW
00389 {
00390 return l4_debugger_query_log_typeid_u(cap, name, idx, l4_utcb());
00391 }
00392
00393 L4_INLINE long
00394 l4_debugger_query_log_name(l4_cap_idx_t cap, unsigned idx,
00395 char *name, unsigned namelen,
00396 char *shortname, unsigned shortnamelen) L4_NOTHROW
00397 {
00398 return l4_debugger_query_log_name_u(cap, idx, name, namelen,
00399 shortname, shortnamelen, l4_utcb());
00400 }
00401
00402 L4_INLINE l4_msgtag_t
00403 l4_debugger_switch_log(l4_cap_idx_t cap, const char *name,
00404 int on_off) L4_NOTHROW
00405 {
00406 return l4_debugger_switch_log_u(cap, name, on_off, l4_utcb());
00407 }
00408
00409 L4_INLINE l4_msgtag_t
00410 l4_debugger_get_object_name(l4_cap_idx_t cap, unsigned id,
00411 char *name, unsigned size) L4_NOTHROW
00412 {
00413 return l4_debugger_get_object_name_u(cap, id, name, size, l4_utcb());
00414 }
00415
00416 L4_INLINE l4_msgtag_t
00417 l4_debugger_add_image_info(l4_cap_idx_t cap, l4_addr_t base,
00418 const char *name) L4_NOTHROW
00419 {
00420 return l4_debugger_add_image_info_u(cap, base, name, l4_utcb());
00421 }

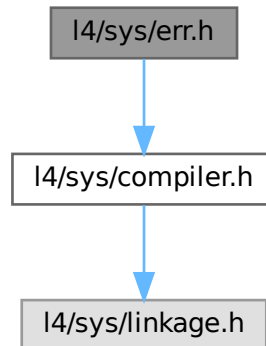
```

## 17.509 l4/sys/err.h File Reference

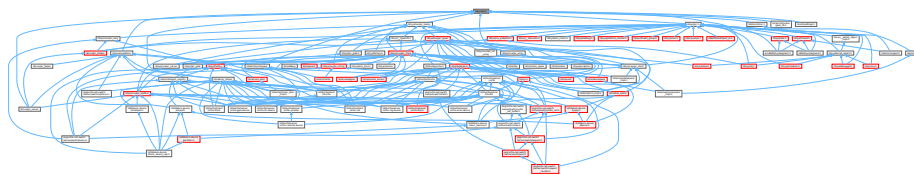
Error codes.



```
#include <l4/sys/compiler.h>
Include dependency graph for err.h:
```



This graph shows which files directly or indirectly include this file:



## Enumerations

```
enum l4_error_code_t {
 L4_EOK = 0, L4_EPERM = 1, L4_ENOENT = 2, L4_EIO = 5,
 L4_ENXIO = 6, L4_E2BIG = 7, L4_EAGAIN = 11, L4_ENOMEM = 12,
 L4_EACCESS = 13, L4_EFAULT = 14, L4_EBUSY = 16, L4_EEXIST = 17,
 L4_ENODEV = 19, L4_ENOTDIR = 20, L4_EINVAL = 22, L4_ENOSPC = 28,
 L4_ERANGE = 34, L4_ENAMETOOLONG = 36, L4_ENOSYS = 38, L4_EBADPROTO = 39,
 L4_EADDRNOTAVAIL = 99, L4_ERRNOMAX = 100, L4_ENOREPLY = 1000, L4_MSGTOOSHORT =
 1001,
 L4_MSGTOOLONG = 1002, L4_MSGMISSARG = 1003, L4_EIPC_LO = 2000, L4_EIPC_HI = 2000 +
 0x1f }
```

*L4 error codes.*

## 17.509.1 Detailed Description

Error codes.

Definition in file [err.h](#).

## 17.510 err.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 *
00010 * License: see LICENSE.spdx (in this directory or the directories above)
00011 */
00012 #pragma once
00013
00014 #include <l4/sys/compiler.h>
00015
00023
00030 enum l4_error_code_t
00031 {
00032 L4_EOK = 0,
00033 L4_EPERM = 1,
00034 L4_ENOENT = 2,
00035 L4_EIO = 5,
00036 L4_ENXIO = 6,
00037 L4_E2BIG = 7,
00038 L4_EAGAIN = 11,
00039 L4_ENOMEM = 12,
00040 L4_EACCESS = 13,
00041 L4_EFAULT = 14,
00042 L4_EBUSY = 16,
00043 L4_EEXIST = 17,
00044 L4_ENODEV = 19,
00045 L4_ENOTDIR = 20,
00046 L4_EINVAL = 22,
00047 L4_ENOSPC = 28,
00048 L4_ERANGE = 34,
00049 L4_ENAMETOOLONG = 36,
00050 L4_ENOSYS = 38,
00051 L4_EBADPROTO = 39,
00052 L4_EADDRNOTAVAIL = 99,
00053 L4_ERRNOMAX = 100,
00054
00055 L4_ENOREPLY = 1000,
00056 L4_MSGTOOSHORT = 1001,
00057 L4_MSGTOOLONG = 1002,
00058 L4_MSGMISSARG = 1003,
00059
00060 L4_EIPC_LO = 2000,
00061 L4_EIPC_HI = 2000 + 0x1f,
00062 };
00063
00064 L4_BEGIN_DECLS
00065 L4_CV char const *l4sys_errtostr(long err) L4_NOTHROW;
00066 L4_END_DECLS
00067
00068

```

## 17.511 l4/sys/exception File Reference

Exception C++ interface.

```

#include <l4/sys/capability>
#include <l4/sys/types.h>
#include <l4/sys/cxx/ipc_types>
#include <l4/sys/cxx/ipc_iface>

```

[illegible]

- class L4::Exception  
*Exception* interface.

- namespace **L4**  
*L4 low-level kernel interface.*

Definition in file [exception.](#)

## 17.512 exception

[Go to the documentation of this file.](#)

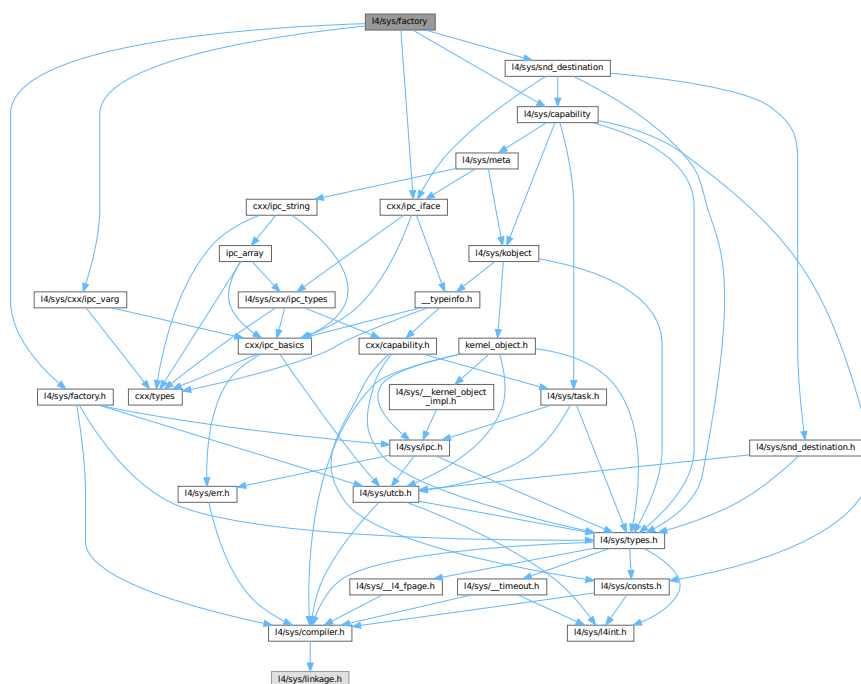
```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2014 Alexander Warg <alexander.warg@kernkonzept.com>
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011
00012 #pragma once
00013
00014 #include <l4/sys/capability>
00015 #include <l4/sys/types.h>
00016 #include <l4/sys/cxx/ipc_types>
00017 #include <l4/sys/cxx/ipc_iface>
00018
00019 namespace L4 {
00020
00031 class L4_EXPORT Exception :
00032 public Kobject_0t<Exception, L4_PROTO_EXCEPTION>
00033 {
00034 public:
00035 // TODO: pass a reference/pointer to the UTCB not copy the regs
00046 L4_INLINE_RPC(
00047 l4_msgtag_t, exception, (L4::Ipc::In_out<l4_exc_regs_t *> regs,
00048 L4::Ipc::Rcv_fpage rwin,
00049 L4::Ipc::Opt<L4::Ipc::Snd_fpage &> fp));
00050
00051 typedef L4::Typeid::Rpc_nocode<exception_t> Rpcs;
00052 };
00053
00054 }
```

## 17.513 l4/sys/factory File Reference

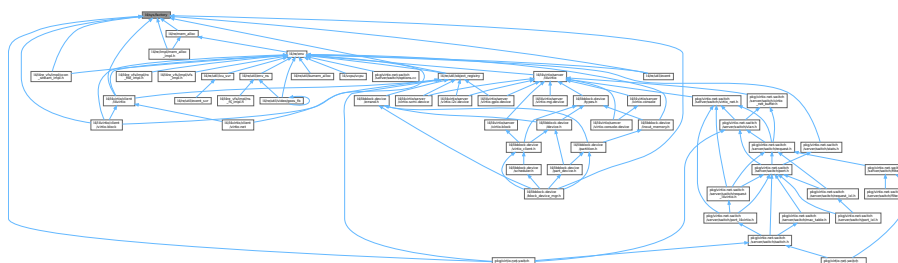
Common factory related definitions.

```
#include <l4/sys/factory.h>
#include <l4/sys/capability>
#include <l4/sys/snd_destination>
#include <l4/sys/cxx/ipc_iface>
#include <l4/sys/cxx/ipc_varg>
```

Include dependency graph for factory:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class `L4::Factory`  
*C++ Factory interface, see `Factory` for the C interface.*
- struct `L4::Factory::Nil`  
*Special type to add a void argument into the factory create stream.*
- struct `L4::Factory::Lstr`  
*Special type to add a pascal string into the factory create stream.*
- class `L4::Factory::S`  
*Stream class for the `create()` argument stream.*

## Namespaces

- namespace **L4**  
*L4 low-level kernel interface.*

## 17.513.1 Detailed Description

Common factory related definitions.

Definition in file [factory](#).

## 17.514 factory

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013
00014 #pragma once
00015
00016 #include <l4/sys/factory.h>
00017 #include <l4/sys/capability>
00018 #include <l4/sys/snd_destination>
00019 #include <l4/sys/cxx/ipc_iface>
00020 #include <l4/sys/cxx/ipc_varg>
00021
00022 namespace L4 {
00023
00038 class Factory : public Kobject_t<Factory, Kobject, L4_PROTO_FACTORY>
00039 {
00040 public:
00041
00042 typedef l4_mword_t Proto;
00043
00047 struct Nil {};
00048
00054 struct Lstr
00055 {
00059 char const *s;
00060
00064 unsigned len;
00065
00070 Lstr(char const *s, unsigned len) noexcept : s(s), len(len) {}
00071 };
00072
00079 class S
00080 {
00081 private:
00082 l4_utcb_t *u;
00083 l4_msgtag_t t;
00084 l4_cap_idx_t f;
00085
00086 template<typename T>
00087 static T &&_move(T &c) { return static_cast<T &&>(c); }
00088
00089 public:
00090 S(S const &) = delete;
00091 S &operator = (S const &) &= delete;
00092
00098 S(S &&o) noexcept
00099 : u(o.u), t(o.t), f(o.f)
00100 { o.t.raw = 0; }
00101
00102 S &operator = (S &&o) & noexcept
00103 {
00104 u = o.u;
00105 t = o.t;
00106 f = o.f;
00107 o.t.raw = 0;
00108 return *this;
00109 }
00110
00125 S(l4_cap_idx_t f, long obj, L4::Cap<void> target,
00126 l4_utcb_t *utcb) noexcept
00127 : u(utcb), t(l4_factory_create_start_u(obj, target.cap(), u)), f(f)
00128 {}
00129
00139 ~S() noexcept

```

```

00140 {
00141 if (t.raw)
00142 l4_factory_create_commit_u(f, t, u);
00143 }
00144
00157 operator l4_msgtag_t () noexcept
00158 {
00159 l4_msgtag_t r = l4_factory_create_commit_u(f, t, u);
00160 t.raw = 0;
00161 return r;
00162 }
00163
00169 void put(l4_mword_t i) noexcept
00170 {
00171 l4_factory_create_add_int_u(i, &t, u);
00172 }
00173
00179 void put(l4_umword_t i) noexcept
00180 {
00181 l4_factory_create_add_uint_u(i, &t, u);
00182 }
00183
00191 void put(char const *s) & noexcept
00192 {
00193 l4_factory_create_add_str_u(s, &t, u);
00194 }
00195
00205 void put(Lstr const &s) & noexcept
00206 {
00207 l4_factory_create_add_lstr_u(s.s, s.len, &t, u);
00208 }
00209
00213 void put(Nil) & noexcept
00214 {
00215 l4_factory_create_add_nil_u(&t, u);
00216 }
00217
00223 void put(l4_fpage_t d) & noexcept
00224 {
00225 l4_factory_create_add_fpage_u(d, &t, u);
00226 }
00227
00236 template<typename T>
00237 S &operator « (T const &d) & noexcept
00238 {
00239 put(d);
00240 return *this;
00241 }
00242
00251 template<typename T>
00252 S &&operator « (T const &d) && noexcept
00253 {
00254 put(d);
00255 return _move(*this);
00256 }
00257 };
00258
00259 public:
00260
00292 S create(Cap<void> target, long obj, l4_utcb_t *utcb = l4_utcb()) noexcept
00293 {
00294 return S(cap(), obj, target, utcb);
00295 }
00296
00328 template<typename OBJ>
00329 S create(Cap<OBJ> target, l4_utcb_t *utcb = l4_utcb()) noexcept
00330 {
00331 return S(cap(), OBJ::Protocol, target, utcb);
00332 }
00333
00334 L4_INLINE_RPC_NF(
00335 l4_msgtag_t, create, (L4::Ipc::Out<L4::Cap<void> > target, l4_mword_t obj,
00336 L4::Ipc::Varg const *args),
00337 L4::Ipc::Call_t<L4_CAP_FPAGE_S>);
00338
00370 l4_msgtag_t create_task(Cap<Task> const & target_cap,
00371 l4_fpage_t *utcb_area,
00372 l4_utcb_t *utcb = l4_utcb()) noexcept
00373 { return l4_factory_create_task_u(cap(), target_cap.cap(), utcb_area, utcb); }
00374
00404 l4_msgtag_t create_factory(Cap<Factory> const &target_cap,
00405 unsigned long limit,
00406 l4_utcb_t *utcb = l4_utcb()) noexcept
00407 { return l4_factory_create_factory_u(cap(), target_cap.cap(), limit, utcb); }
00408
00440 l4_msgtag_t create_gate(Cap<void> const &target_cap,

```

```

00441 Cap<Snd_destination> const &snd_dst_cap,
00442 l4_umword_t label,
00443 l4_utcb_t *utcb = l4_utcb()) noexcept
00444 {
00445 return l4_factory_create_gate_u(cap(), target_cap.cap(), snd_dst_cap.cap(),
00446 label, utcb);
00447 }
00448
00475 l4_msgtag_t create_thread_group(Cap<Thread_group> const &target_cap,
00476 unsigned policy,
00477 l4_utcb_t *utcb = l4_utcb()) noexcept
00478 {
00479 return l4_factory_create_thread_group_u(cap(), target_cap.cap(),
00480 policy, utcb);
00481 }
00482
00483 typedef L4::Typeid::Rpc_nocode<create_t> Rpccs;
00484 };
00485
00486 }

```

## 17.515 l4/sys/factory.h File Reference

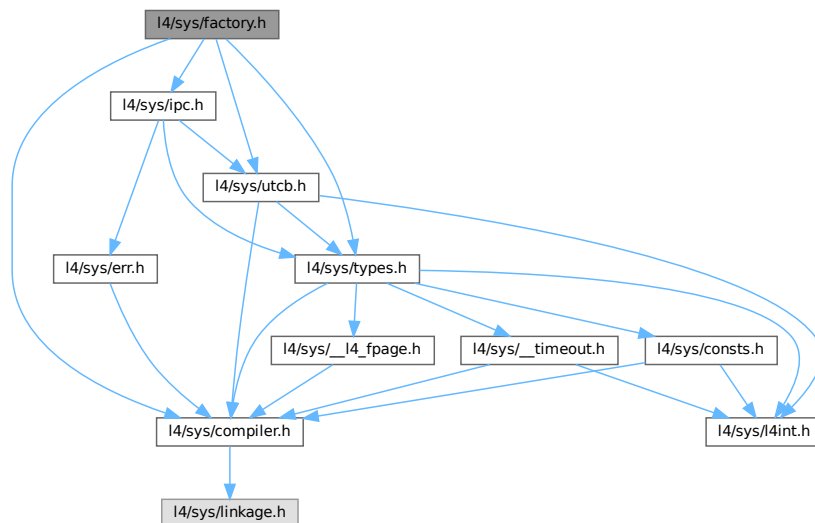
Common factory related definitions.

```

#include <l4/sys/compiler.h>
#include <l4/sys/types.h>
#include <l4/sys/utcb.h>
#include <l4/sys/ipc.h>

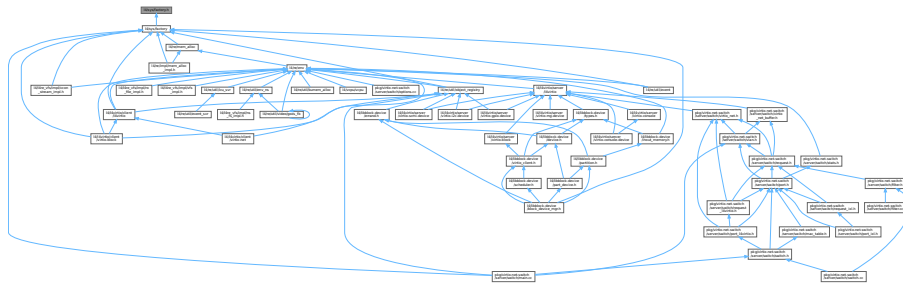
```

Include dependency graph for factory.h:





This graph shows which files directly or indirectly include this file:



## Functions

- [l4\\_msgtag\\_t l4\\_factory\\_create\\_task](#) ([l4\\_cap\\_idx\\_t](#) factory, [l4\\_cap\\_idx\\_t](#) target\_cap, [l4\\_fpage\\_t](#) \*utcb\_area) [L4\\_NOTHROW](#)  
*Create a new task.*
- [l4\\_msgtag\\_t l4\\_factory\\_create\\_thread](#) ([l4\\_cap\\_idx\\_t](#) factory, [l4\\_cap\\_idx\\_t](#) target\_cap) [L4\\_NOTHROW](#)  
*Create a new thread.*
- [l4\\_msgtag\\_t l4\\_factory\\_create\\_factory](#) ([l4\\_cap\\_idx\\_t](#) factory, [l4\\_cap\\_idx\\_t](#) target\_cap, unsigned long limit) [L4\\_NOTHROW](#)  
*Create a new factory.*
- [l4\\_msgtag\\_t l4\\_factory\\_create\\_gate](#) ([l4\\_cap\\_idx\\_t](#) factory, [l4\\_cap\\_idx\\_t](#) target\_cap, [l4\\_cap\\_idx\\_t](#) snd\_dst, [l4\\_umword\\_t](#) label) [L4\\_NOTHROW](#)  
*Create a new IPC gate, optionally bound to a send destination (a thread or thread group).*
- [l4\\_msgtag\\_t l4\\_factory\\_create\\_irq](#) ([l4\\_cap\\_idx\\_t](#) factory, [l4\\_cap\\_idx\\_t](#) target\_cap) [L4\\_NOTHROW](#)  
*Create a new IRQ sender.*
- [l4\\_msgtag\\_t l4\\_factory\\_create\\_vm](#) ([l4\\_cap\\_idx\\_t](#) factory, [l4\\_cap\\_idx\\_t](#) target\_cap) [L4\\_NOTHROW](#)  
*Create a new virtual machine.*
- [l4\\_msgtag\\_t l4\\_factory\\_create\\_vcpu\\_context](#) ([l4\\_cap\\_idx\\_t](#) factory, [l4\\_cap\\_idx\\_t](#) target\_cap) [L4\\_NOTHROW](#)  
*Create a new vCPU context.*
- [l4\\_msgtag\\_t l4\\_factory\\_create\\_thread\\_group](#) ([l4\\_cap\\_idx\\_t](#) factory, [l4\\_cap\\_idx\\_t](#) target\_cap, unsigned policy) [L4\\_NOTHROW](#)  
*Create a new thread group.*
- [l4\\_msgtag\\_t l4\\_factory\\_create](#) ([l4\\_cap\\_idx\\_t](#) factory, long obj, [l4\\_cap\\_idx\\_t](#) target) [L4\\_NOTHROW](#)  
*Create a new object.*

## 17.515.1 Detailed Description

Common factory related definitions.

Definition in file [factory.h](#).

## 17.516 factory.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00009 * Björn Döbel <doebel@os.inf.tu-dresden.de>,
00010 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>,
00011 * Henning Schild <hschild@os.inf.tu-dresden.de>
00012 * economic rights: Technische Universität Dresden (Germany)
00013 *
00014 * License: see LICENSE.spdx (in this directory or the directories above)
00015 */
00016 #pragma once
00017
00018 #include <l4/sys/compiler.h>
00019 #include <l4/sys/types.h>
00020 #include <l4/sys/utcb.h>
00021
00050
00056
00086 L4_INLINE l4_msgtag_t
00087 l4_factory_create_task(l4_cap_idx_t factory, l4_cap_idx_t target_cap,
00088 l4_fpage_t *utcb_area) L4_NOTHROW;
00089
00094 L4_INLINE l4_msgtag_t
00095 l4_factory_create_task_u(l4_cap_idx_t factory, l4_cap_idx_t target_cap,
00096 l4_fpage_t *utcb_area, l4_utcb_t *utcb) L4_NOTHROW;
00097
00116 L4_INLINE l4_msgtag_t
00117 l4_factory_create_thread(l4_cap_idx_t factory,
00118 l4_cap_idx_t target_cap) L4_NOTHROW;
00119
00124 L4_INLINE l4_msgtag_t
00125 l4_factory_create_thread_u(l4_cap_idx_t factory,
00126 l4_cap_idx_t target_cap, l4_utcb_t *utcb) L4_NOTHROW;
00127
00153 L4_INLINE l4_msgtag_t
00154 l4_factory_create_factory(l4_cap_idx_t factory, l4_cap_idx_t target_cap,
00155 unsigned long limit) L4_NOTHROW;
00156
00161 L4_INLINE l4_msgtag_t
00162 l4_factory_create_factory_u(l4_cap_idx_t factory, l4_cap_idx_t target_cap,
00163 unsigned long limit, l4_utcb_t *utcb) L4_NOTHROW;
00164
00194 L4_INLINE l4_msgtag_t
00195 l4_factory_create_gate(l4_cap_idx_t factory,
00196 l4_cap_idx_t target_cap,
00197 l4_cap_idx_t snd_dst_cap, l4_umword_t label) L4_NOTHROW;
00198
00203 L4_INLINE l4_msgtag_t
00204 l4_factory_create_gate_u(l4_cap_idx_t factory,
00205 l4_cap_idx_t target_cap,
00206 l4_cap_idx_t snd_dst_cap, l4_umword_t label,
00207 l4_utcb_t *utcb) L4_NOTHROW;
00208
00225 L4_INLINE l4_msgtag_t
00226 l4_factory_create_irq(l4_cap_idx_t factory,
00227 l4_cap_idx_t target_cap) L4_NOTHROW;
00228
00233 L4_INLINE l4_msgtag_t
00234 l4_factory_create_irq_u(l4_cap_idx_t factory,
00235 l4_cap_idx_t target_cap, l4_utcb_t *utcb) L4_NOTHROW;
00236
00255 L4_INLINE l4_msgtag_t
00256 l4_factory_create_vm(l4_cap_idx_t factory,
00257 l4_cap_idx_t target_cap) L4_NOTHROW;
00258
00278 L4_INLINE l4_msgtag_t
00279 l4_factory_create_vcpu_context(l4_cap_idx_t factory,
00280 l4_cap_idx_t target_cap) L4_NOTHROW;
00281
00309 L4_INLINE l4_msgtag_t
00310 l4_factory_create_thread_group(l4_cap_idx_t factory,
00311 l4_cap_idx_t target_cap,
00312 unsigned policy) L4_NOTHROW;
00313
00318 L4_INLINE l4_msgtag_t
00319 l4_factory_create_vm_u(l4_cap_idx_t factory,
00320 l4_cap_idx_t target_cap, l4_utcb_t *utcb) L4_NOTHROW;
00321
00326 L4_INLINE l4_msgtag_t
00327 l4_factory_create_vcpu_context_u(l4_cap_idx_t factory,
00328 l4_cap_idx_t target_cap,

```

```

00329 l4_utcb_t *utcb) L4_NOTHROW;
00330
00335 L4_INLINE l4_msgtag_t
00336 l4_factory_create_thread_group_u(l4_cap_idx_t factory,
00337 l4_cap_idx_t target_cap,
00338 unsigned policy,
00339 l4_utcb_t *u) L4_NOTHROW;
00340
00345 L4_INLINE l4_msgtag_t
00346 l4_factory_create_start_u(long obj, l4_cap_idx_t target,
00347 l4_utcb_t *utcb) L4_NOTHROW;
00348
00353 L4_INLINE int
00354 l4_factory_create_add_fpage_u(l4_fpage_t d, l4_msgtag_t *tag,
00355 l4_utcb_t *utcb) L4_NOTHROW;
00356
00361 L4_INLINE int
00362 l4_factory_create_add_int_u(l4_mword_t d, l4_msgtag_t *tag,
00363 l4_utcb_t *utcb) L4_NOTHROW;
00364
00369 L4_INLINE int
00370 l4_factory_create_add_uint_u(l4_umword_t d, l4_msgtag_t *tag,
00371 l4_utcb_t *utcb) L4_NOTHROW;
00372
00377 L4_INLINE int
00378 l4_factory_create_add_str_u(char const *s, l4_msgtag_t *tag,
00379 l4_utcb_t *utcb) L4_NOTHROW;
00380
00385 L4_INLINE int
00386 l4_factory_create_add_lstr_u(char const *s, unsigned len, l4_msgtag_t *tag,
00387 l4_utcb_t *utcb) L4_NOTHROW;
00388
00393 L4_INLINE int
00394 l4_factory_create_add_nil_u(l4_msgtag_t *tag, l4_utcb_t *utcb) L4_NOTHROW;
00395
00400 L4_INLINE l4_msgtag_t
00401 l4_factory_create_commit_u(l4_cap_idx_t factory, l4_msgtag_t tag,
00402 l4_utcb_t *utcb) L4_NOTHROW;
00403
00408 L4_INLINE l4_msgtag_t
00409 l4_factory_create_u(l4_cap_idx_t factory, long obj, l4_cap_idx_t target,
00410 l4_utcb_t *utcb) L4_NOTHROW;
00411
00412
00430 L4_INLINE l4_msgtag_t
00431 l4_factory_create(l4_cap_idx_t factory, long obj,
00432 l4_cap_idx_t target) L4_NOTHROW;
00433
00434 /* IMPLEMENTATION -----*/
00435
00436 #include <l4/sys/ipc.h>
00437
00438 L4_INLINE l4_msgtag_t
00439 l4_factory_create_task_u(l4_cap_idx_t factory,
00440 l4_cap_idx_t target_cap, l4_fpage_t *utcb_area,
00441 l4_utcb_t *u) L4_NOTHROW
00442 {
00443 l4_msgtag_t t;
00444 t = l4_factory_create_start_u(L4_PROTO_TASK, target_cap, u);
00445 l4_factory_create_add_fpage_u(*utcb_area, &t, u);
00446 t = l4_factory_create_commit_u(factory, t, u);
00447 if (!l4_msgtag_has_error(t))
00448 {
00449 l4_msg_regs_t *v = l4_utcb_mr_u(u);
00450 utcb_area->raw = v->mr[0];
00451 }
00452 return t;
00453 }
00454
00455 L4_INLINE l4_msgtag_t
00456 l4_factory_create_thread_u(l4_cap_idx_t factory,
00457 l4_cap_idx_t target_cap, l4_utcb_t *u) L4_NOTHROW
00458 {
00459 return l4_factory_create_u(factory, L4_PROTO_THREAD, target_cap, u);
00460 }
00461
00462 L4_INLINE l4_msgtag_t
00463 l4_factory_create_factory_u(l4_cap_idx_t factory,
00464 l4_cap_idx_t target_cap, unsigned long limit,
00465 l4_utcb_t *u) L4_NOTHROW
00466 {
00467 l4_msgtag_t t;
00468 t = l4_factory_create_start_u(L4_PROTO_FACTORY, target_cap, u);
00469 l4_factory_create_add_uint_u(limit, &t, u);
00470 return l4_factory_create_commit_u(factory, t, u);
00471 }
00472

```

```

00473 L4_INLINE l4_msgtag_t
00474 l4_factory_create_gate_u(l4_cap_idx_t factory,
00475 l4_cap_idx_t target_cap,
00476 l4_cap_idx_t snd_dst_cap, l4_umword_t label,
00477 l4_utcb_t *u) L4_NOTHROW
00478 {
00479 l4_msgtag_t t;
00480 l4_msg_regs_t *v;
00481 int items = 0;
00482 t = l4_factory_create_start_u(0, target_cap, u);
00483 l4_factory_create_add_uint_u(label, &t, u);
00484 v = l4_utcb_mr_u(u);
00485 if (!(snd_dst_cap & L4_INVALID_CAP_BIT))
00486 {
00487 items = 1;
00488 v->mr[3] = l4_map_obj_control(0,0);
00489 v->mr[4] = l4_obj_fpage(snd_dst_cap, 0, L4_CAP_FPAGE_RWS).raw;
00490 }
00491 t = l4_msgtag(l4_msgtag_label(t), l4_msgtag_words(t), items, l4_msgtag_flags(t));
00492 return l4_factory_create_commit_u(factory, t, u);
00493 }
00494
00495 L4_INLINE l4_msgtag_t
00496 l4_factory_create_irq_u(l4_cap_idx_t factory,
00497 l4_cap_idx_t target_cap, l4_utcb_t *u) L4_NOTHROW
00498 {
00499 return l4_factory_create_u(factory, L4_PROTO_IRQ_SENDER, target_cap, u);
00500 }
00501
00502 L4_INLINE l4_msgtag_t
00503 l4_factory_create_vm_u(l4_cap_idx_t factory,
00504 l4_cap_idx_t target_cap,
00505 l4_utcb_t *u) L4_NOTHROW
00506 {
00507 return l4_factory_create_u(factory, L4_PROTO_VM, target_cap, u);
00508 }
00509
00510 L4_INLINE l4_msgtag_t
00511 l4_factory_create_vcpu_context_u(l4_cap_idx_t factory,
00512 l4_cap_idx_t target_cap,
00513 l4_utcb_t *u) L4_NOTHROW
00514 {
00515 return l4_factory_create_u(factory, L4_PROTO_VCPU_CONTEXT, target_cap, u);
00516 }
00517
00518 L4_INLINE l4_msgtag_t
00519 l4_factory_create_thread_group_u(l4_cap_idx_t factory,
00520 l4_cap_idx_t target_cap,
00521 unsigned policy,
00522 l4_utcb_t *u) L4_NOTHROW
00523 {
00524 l4_msgtag_t t = l4_factory_create_start_u(L4_PROTO_THREAD_GROUP, target_cap, u);
00525 l4_factory_create_add_uint_u(policy, &t, u);
00526 return l4_factory_create_commit_u(factory, t, u);
00527 }
00528
00529
00530 L4_INLINE l4_msgtag_t
00531 l4_factory_create_task(l4_cap_idx_t factory,
00532 l4_cap_idx_t target_cap, l4_fpage_t *utcb_area) L4_NOTHROW
00533 {
00534 return l4_factory_create_task_u(factory, target_cap, utcb_area, l4_utcb());
00535 }
00536
00537 L4_INLINE l4_msgtag_t
00538 l4_factory_create_thread(l4_cap_idx_t factory,
00539 l4_cap_idx_t target_cap) L4_NOTHROW
00540 {
00541 return l4_factory_create_thread_u(factory, target_cap, l4_utcb());
00542 }
00543
00544 L4_INLINE l4_msgtag_t
00545 l4_factory_create_factory(l4_cap_idx_t factory,
00546 l4_cap_idx_t target_cap, unsigned long limit) L4_NOTHROW
00547 {
00548 return l4_factory_create_factory_u(factory, target_cap, limit, l4_utcb());
00549 }
00550
00551
00552 L4_INLINE l4_msgtag_t
00553 l4_factory_create_gate(l4_cap_idx_t factory,
00554 l4_cap_idx_t target_cap,
00555 l4_cap_idx_t snd_dst_cap, l4_umword_t label) L4_NOTHROW
00556 {
00557 return l4_factory_create_gate_u(factory, target_cap, snd_dst_cap, label, l4_utcb());
00558 }
00559

```

```

00560 L4_INLINE l4_msgtag_t
00561 l4_factory_create_irq(l4_cap_idx_t factory,
00562 l4_cap_idx_t target_cap) L4_NOTHROW
00563 {
00564 return l4_factory_create_irq_u(factory, target_cap, l4_utcb());
00565 }
00566
00567 L4_INLINE l4_msgtag_t
00568 l4_factory_create_vm(l4_cap_idx_t factory,
00569 l4_cap_idx_t target_cap) L4_NOTHROW
00570 {
00571 return l4_factory_create_vm_u(factory, target_cap, l4_utcb());
00572 }
00573
00574 L4_INLINE l4_msgtag_t
00575 l4_factory_create_vcpu_context(l4_cap_idx_t factory,
00576 l4_cap_idx_t target_cap) L4_NOTHROW
00577 {
00578 return l4_factory_create_vcpu_context_u(factory, target_cap, l4_utcb());
00579 }
00580
00581 L4_INLINE l4_msgtag_t
00582 l4_factory_create_thread_group(l4_cap_idx_t factory,
00583 l4_cap_idx_t target_cap,
00584 unsigned policy) L4_NOTHROW
00585 {
00586 return l4_factory_create_thread_group_u(factory, target_cap, policy, l4_utcb());
00587 }
00588
00589
00590 L4_INLINE l4_msgtag_t
00591 l4_factory_create_start_u(long obj, l4_cap_idx_t target_cap,
00592 l4_utcb_t *u) L4_NOTHROW
00593 {
00594 l4_msg_regs_t *v = l4_utcb_mr_u(u);
00595 l4_buf_regs_t *b = l4_utcb_br_u(u);
00596 v->mr[0] = obj;
00597 b->bdr = 0;
00598 b->br[0] = target_cap | L4_RCV_ITEM_SINGLE_CAP;
00599 return l4_msgtag(L4_PROTO_FACTORY, 1, 0, 0);
00600 }
00601
00602 L4_INLINE int
00603 l4_factory_create_add_fpage_u(l4_fpage_t d, l4_msgtag_t *tag,
00604 l4_utcb_t *u) L4_NOTHROW
00605 {
00606 l4_msg_regs_t *v = l4_utcb_mr_u(u);
00607 int w = l4_msgtag_words(*tag);
00608 if (w + 2 > L4_UTCB_GENERIC_DATA_SIZE)
00609 return 0;
00610 v->mr[w] = L4_VARG_TYPE_FPAGE | (sizeof(l4_fpage_t) << 16);
00611 v->mr[w + 1] = d.raw;
00612 w += 2;
00613 tag->raw = (tag->raw & ~0x3fUL) | (w & 0x3f);
00614 return 1;
00615 }
00616
00617 L4_INLINE int
00618 l4_factory_create_add_int_u(l4_mword_t d, l4_msgtag_t *tag,
00619 l4_utcb_t *u) L4_NOTHROW
00620 {
00621 l4_msg_regs_t *v = l4_utcb_mr_u(u);
00622 int w = l4_msgtag_words(*tag);
00623 if (w + 2 > L4_UTCB_GENERIC_DATA_SIZE)
00624 return 0;
00625 v->mr[w] = L4_VARG_TYPE_MWORD | (sizeof(l4_mword_t) << 16);
00626 v->mr[w + 1] = d;
00627 w += 2;
00628 tag->raw = (tag->raw & ~0x3fUL) | (w & 0x3f);
00629 return 1;
00630 }
00631
00632 L4_INLINE int
00633 l4_factory_create_add_uint_u(l4_umword_t d, l4_msgtag_t *tag,
00634 l4_utcb_t *u) L4_NOTHROW
00635 {
00636 l4_msg_regs_t *v = l4_utcb_mr_u(u);
00637 int w = l4_msgtag_words(*tag);
00638 if (w + 2 > L4_UTCB_GENERIC_DATA_SIZE)
00639 return 0;
00640 v->mr[w] = L4_VARG_TYPE_UMWORD | (sizeof(l4_umword_t) << 16);
00641 v->mr[w + 1] = d;
00642 w += 2;
00643 tag->raw = (tag->raw & ~0x3fUL) | (w & 0x3f);
00644 return 1;
00645 }
00646

```

```

00647 L4_INLINE int
00648 l4_factory_create_add_str_u(char const *s, l4_msgtag_t *tag,
00649 l4_utcb_t *u) L4_NOTHROW
00650 {
00651 return l4_factory_create_add_lstr_u(s, __builtin_strlen(s) + 1, tag, u);
00652 }
00653
00654 L4_INLINE int
00655 l4_factory_create_add_lstr_u(char const *s, unsigned len, l4_msgtag_t *tag,
00656 l4_utcb_t *u) L4_NOTHROW
00657 {
00658
00659 l4_msg_regs_t *v = l4_utcb_mr_u(u);
00660 unsigned w = l4_msgtag_words(*tag);
00661 char *c;
00662 unsigned i;
00663
00664 if (w + 1 + l4_bytes_to_mwords(len) > L4_UTCB_GENERIC_DATA_SIZE)
00665 return 0;
00666
00667 v->mr[w] = L4_VARG_TYPE_STRING | (len << 16);
00668 c = (char*)&v->mr[w + 1];
00669 for (i = 0; i < len; ++i)
00670 *c++ = *s++;
00671
00672 w = w + 1 + l4_bytes_to_mwords(len);
00673
00674 tag->raw = (tag->raw & ~0x3fUL) | (w & 0x3f);
00675 return 1;
00676 }
00677
00678 L4_INLINE int
00679 l4_factory_create_add_nil_u(l4_msgtag_t *tag, l4_utcb_t *utcb) L4_NOTHROW
00680 {
00681 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00682 int w = l4_msgtag_words(*tag);
00683 v->mr[w] = L4_VARG_TYPE_NIL;
00684 ++w;
00685 tag->raw = (tag->raw & ~0x3fUL) | (w & 0x3f);
00686 return 1;
00687 }
00688
00689
00690 L4_INLINE l4_msgtag_t
00691 l4_factory_create_commit_u(l4_cap_idx_t factory, l4_msgtag_t tag,
00692 l4_utcb_t *u) L4_NOTHROW
00693 {
00694 return l4_ipc_call(factory, u, tag, L4_IPC_NEVER);
00695 }
00696
00697 L4_INLINE l4_msgtag_t
00698 l4_factory_create_u(l4_cap_idx_t factory, long obj, l4_cap_idx_t target,
00699 l4_utcb_t *utcb) L4_NOTHROW
00700 {
00701 l4_msgtag_t t = l4_factory_create_start_u(obj, target, utcb);
00702 return l4_factory_create_commit_u(factory, t, utcb);
00703 }
00704
00705
00706 L4_INLINE l4_msgtag_t
00707 l4_factory_create(l4_cap_idx_t factory, long obj,
00708 l4_cap_idx_t target) L4_NOTHROW
00709 {
00710 return l4_factory_create_u(factory, obj, target, l4_utcb());
00711 }

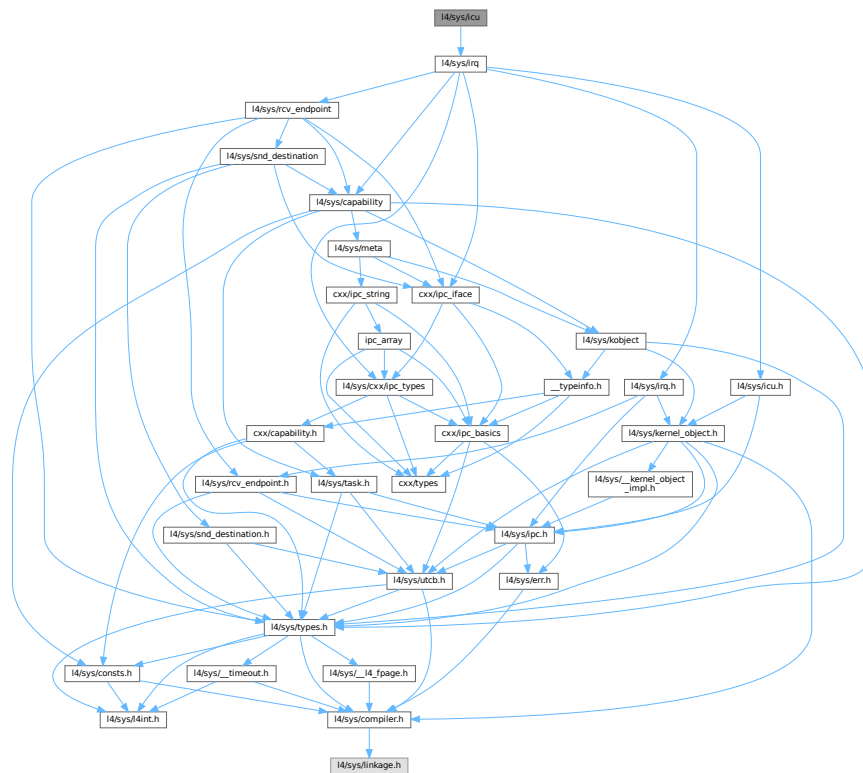
```

## 17.517 I4/sys/icu File Reference

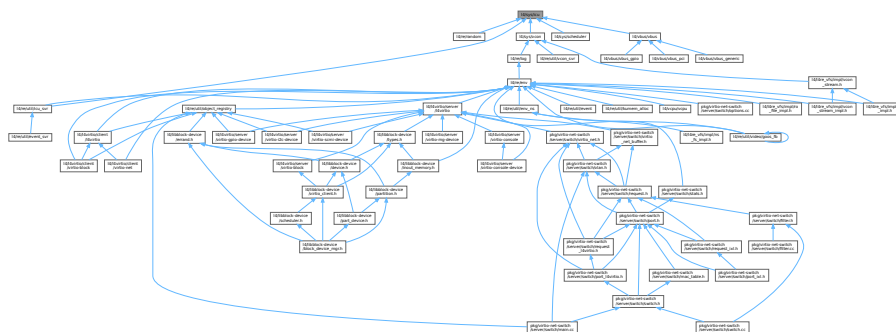
Interrupt controller.

```
#include <l4/sys/irq>
```

Include dependency graph for icu:



This graph shows which files directly or indirectly include this file:



## 17.517.1 Detailed Description

Interrupt controller.

Definition in file [icu](#).

## 17.518 icu

[Go to the documentation of this file.](#)

```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00007 /*
00008 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #pragma once
00014
00015 #include <l4/sys/irq>
```

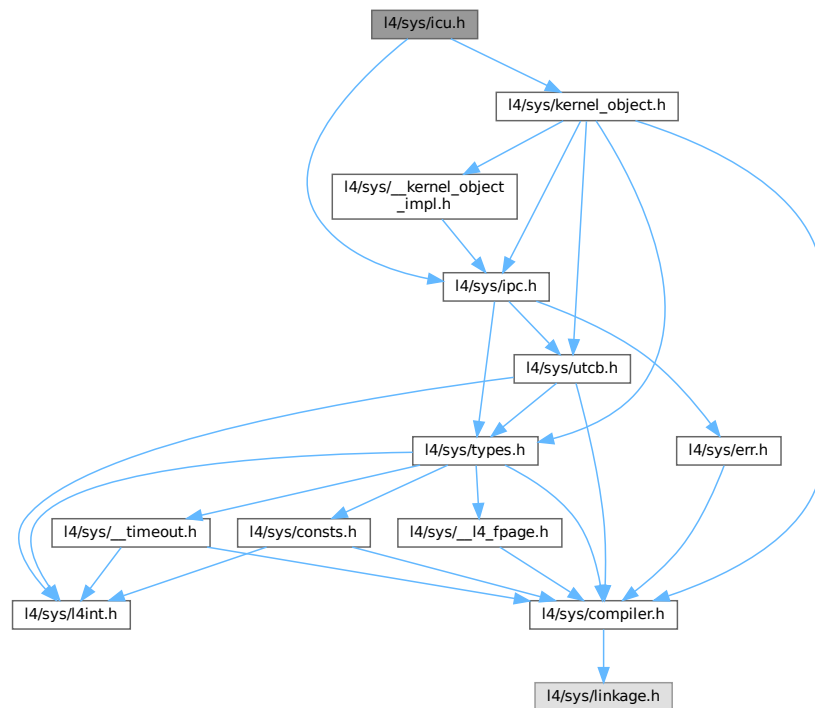
## 17.519 l4/sys/icu.h File Reference

Interrupt controller.

```
#include <l4/sys/kernel_object.h>
```

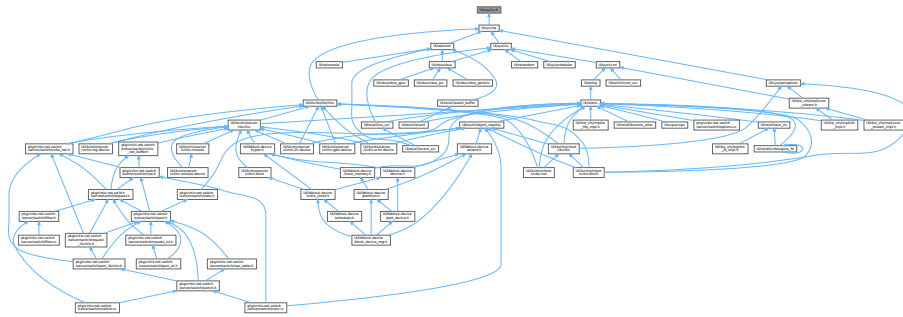
```
#include <l4/sys/ipc.h>
```

Include dependency graph for icu.h:





This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [l4\\_icu\\_info\\_t](#)  
*Info structure for an ICU.*
- struct [l4\\_icu\\_msi\\_info\\_t](#)  
*Info to use for a specific MSI.*

## Typedefs

- typedef struct [l4\\_icu\\_info\\_t](#) [l4\\_icu\\_info\\_t](#)  
*Info structure for an ICU.*
- typedef struct [l4\\_icu\\_msi\\_info\\_t](#) [l4\\_icu\\_msi\\_info\\_t](#)  
*Info to use for a specific MSI.*

## Enumerations

- enum [L4\\_icu\\_flags](#) { [L4\\_ICU\\_FLAG\\_MSI](#) }  
*Flags for IRQ numbers used for the ICU.*
- enum [L4\\_irq\\_mode](#) {  
[L4\\_IRQ\\_F\\_NONE](#) = 0 , [L4\\_IRQ\\_F\\_SET\\_MODE](#) = 0x1 , [L4\\_IRQ\\_F\\_LEVEL](#) = 0x2 , [L4\\_IRQ\\_F\\_EDGE](#) = 0x0 ,  
[L4\\_IRQ\\_F\\_POS](#) = 0x0 , [L4\\_IRQ\\_F\\_NEG](#) = 0x4 , [L4\\_IRQ\\_F\\_BOTH](#) = 0x8 , [L4\\_IRQ\\_F\\_LEVEL\\_HIGH](#) =  
[L4\\_IRQ\\_F\\_SET\\_MODE](#) | [L4\\_IRQ\\_F\\_LEVEL](#) | [L4\\_IRQ\\_F\\_POS](#) ,  
[L4\\_IRQ\\_F\\_LEVEL\\_LOW](#) = [L4\\_IRQ\\_F\\_SET\\_MODE](#) | [L4\\_IRQ\\_F\\_LEVEL](#) | [L4\\_IRQ\\_F\\_NEG](#) , [L4\\_IRQ\\_F\\_POS\\_EDGE](#)  
= [L4\\_IRQ\\_F\\_SET\\_MODE](#) | [L4\\_IRQ\\_F\\_EDGE](#) | [L4\\_IRQ\\_F\\_POS](#) , [L4\\_IRQ\\_F\\_NEG\\_EDGE](#) = [L4\\_IRQ\\_F\\_](#)  
[SET\\_MODE](#) | [L4\\_IRQ\\_F\\_EDGE](#) | [L4\\_IRQ\\_F\\_NEG](#) , [L4\\_IRQ\\_F\\_BOTH\\_EDGE](#) = [L4\\_IRQ\\_F\\_SET\\_MODE](#) |  
[L4\\_IRQ\\_F\\_EDGE](#) | [L4\\_IRQ\\_F\\_BOTH](#) ,  
[L4\\_IRQ\\_F\\_MASK](#) = 0xf , [L4\\_IRQ\\_F\\_SET\\_WAKEUP](#) = 0x10 , [L4\\_IRQ\\_F\\_CLEAR\\_WAKEUP](#) = 0x20 }  
*Interrupt attributes.*
- enum [L4\\_icu\\_opcode](#) {  
[L4\\_ICU\\_OP\\_BIND](#) , [L4\\_ICU\\_OP\\_UNBIND](#) , [L4\\_ICU\\_OP\\_INFO](#) , [L4\\_ICU\\_OP\\_MSI\\_INFO](#) ,  
[L4\\_ICU\\_OP\\_UNMASK](#) , [L4\\_ICU\\_OP\\_MASK](#) , [L4\\_ICU\\_OP\\_SET\\_MODE](#) }  
*Opcodes to the ICU interface.*

## Functions

- [l4\\_msgtag\\_t l4\\_icu\\_bind](#) ([l4\\_cap\\_idx\\_t](#) icu, unsigned irqnum, [l4\\_cap\\_idx\\_t](#) irq) [L4\\_NOTHROW](#)  
*Bind an interrupt line of an interrupt controller to an interrupt object.*
- [l4\\_msgtag\\_t l4\\_icu\\_bind\\_u](#) ([l4\\_cap\\_idx\\_t](#) icu, unsigned irqnum, [l4\\_cap\\_idx\\_t](#) irq, [l4\\_utcb\\_t](#) \*utcb) [L4\\_NOTHROW](#)  
*Bind an interrupt line of an interrupt controller to an interrupt object.*
- [l4\\_msgtag\\_t l4\\_icu\\_unbind](#) ([l4\\_cap\\_idx\\_t](#) icu, unsigned irqnum, [l4\\_cap\\_idx\\_t](#) irq) [L4\\_NOTHROW](#)  
*Remove binding of an interrupt line from the interrupt controller object.*
- [l4\\_msgtag\\_t l4\\_icu\\_unbind\\_u](#) ([l4\\_cap\\_idx\\_t](#) icu, unsigned irqnum, [l4\\_cap\\_idx\\_t](#) irq, [l4\\_utcb\\_t](#) \*utcb) [L4\\_NOTHROW](#)  
*Remove binding of an interrupt line from the interrupt controller object.*
- [l4\\_msgtag\\_t l4\\_icu\\_set\\_mode](#) ([l4\\_cap\\_idx\\_t](#) icu, unsigned irqnum, [l4\\_umword\\_t](#) mode) [L4\\_NOTHROW](#)  
*Set interrupt mode.*
- [l4\\_msgtag\\_t l4\\_icu\\_set\\_mode\\_u](#) ([l4\\_cap\\_idx\\_t](#) icu, unsigned irqnum, [l4\\_umword\\_t](#) mode, [l4\\_utcb\\_t](#) \*utcb) [L4\\_NOTHROW](#)  
*Set interrupt mode.*
- [l4\\_msgtag\\_t l4\\_icu\\_info](#) ([l4\\_cap\\_idx\\_t](#) icu, [l4\\_icu\\_info\\_t](#) \*info) [L4\\_NOTHROW](#)  
*Get information about the ICU features.*
- [l4\\_msgtag\\_t l4\\_icu\\_info\\_u](#) ([l4\\_cap\\_idx\\_t](#) icu, [l4\\_icu\\_info\\_t](#) \*info, [l4\\_utcb\\_t](#) \*utcb) [L4\\_NOTHROW](#)  
*Get information about the ICU features.*
- [l4\\_msgtag\\_t l4\\_icu\\_msi\\_info](#) ([l4\\_cap\\_idx\\_t](#) icu, unsigned irqnum, [l4\\_uint64\\_t](#) source, [l4\\_icu\\_msi\\_info\\_t](#) \*msi\_info) [L4\\_NOTHROW](#)  
*Get MSI info about IRQ.*
- [l4\\_msgtag\\_t l4\\_icu\\_msi\\_info\\_u](#) ([l4\\_cap\\_idx\\_t](#) icu, unsigned irqnum, [l4\\_uint64\\_t](#) source, [l4\\_icu\\_msi\\_info\\_t](#) \*msi\_info, [l4\\_utcb\\_t](#) \*utcb) [L4\\_NOTHROW](#)  
*Get MSI info about IRQ.*
- [l4\\_msgtag\\_t l4\\_icu\\_unmask](#) ([l4\\_cap\\_idx\\_t](#) icu, unsigned irqnum, [l4\\_umword\\_t](#) \*label, [l4\\_timeout\\_t](#) to) [L4\\_NOTHROW](#)  
*Unmask an IRQ line.*
- [l4\\_msgtag\\_t l4\\_icu\\_unmask\\_u](#) ([l4\\_cap\\_idx\\_t](#) icu, unsigned irqnum, [l4\\_umword\\_t](#) \*label, [l4\\_timeout\\_t](#) to, [l4\\_utcb\\_t](#) \*utcb) [L4\\_NOTHROW](#)  
*Unmask the given interrupt line.*
- [l4\\_msgtag\\_t l4\\_icu\\_mask](#) ([l4\\_cap\\_idx\\_t](#) icu, unsigned irqnum, [l4\\_umword\\_t](#) \*label, [l4\\_timeout\\_t](#) to) [L4\\_NOTHROW](#)  
*Mask an IRQ line.*
- [l4\\_msgtag\\_t l4\\_icu\\_mask\\_u](#) ([l4\\_cap\\_idx\\_t](#) icu, unsigned irqnum, [l4\\_umword\\_t](#) \*label, [l4\\_timeout\\_t](#) to, [l4\\_utcb\\_t](#) \*utcb) [L4\\_NOTHROW](#)  
*Mask an IRQ line.*

### 17.519.1 Detailed Description

Interrupt controller.

Definition in file [icu.h](#).

## 17.520 icu.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00009 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #pragma once
00015
00016 #include <l4/sys/kernel_object.h>
00017 #include <l4/sys/ipc.h>
00018
00047
00048
00053 enum L4_icu_flags
00054 {
00062 L4_ICU_FLAG_MSI = 0x80000000,
00063 };
00064
00065
00070 enum L4_irq_mode
00071 {
00073 L4_IRQ_F_NONE = 0,
00074 L4_IRQ_F_SET_MODE = 0x1,
00075 L4_IRQ_F_LEVEL = 0x2,
00076 L4_IRQ_F_EDGE = 0x0,
00077 L4_IRQ_F_POS = 0x0,
00078 L4_IRQ_F_NEG = 0x4,
00079 L4_IRQ_F_BOTH = 0x8,
00080 L4_IRQ_F_LEVEL_HIGH = L4_IRQ_F_SET_MODE | L4_IRQ_F_LEVEL | L4_IRQ_F_POS,
00081 L4_IRQ_F_LEVEL_LOW = L4_IRQ_F_SET_MODE | L4_IRQ_F_LEVEL | L4_IRQ_F_NEG,
00082 L4_IRQ_F_POS_EDGE = L4_IRQ_F_SET_MODE | L4_IRQ_F_EDGE | L4_IRQ_F_POS,
00083 L4_IRQ_F_NEG_EDGE = L4_IRQ_F_SET_MODE | L4_IRQ_F_EDGE | L4_IRQ_F_NEG,
00084 L4_IRQ_F_BOTH_EDGE = L4_IRQ_F_SET_MODE | L4_IRQ_F_EDGE | L4_IRQ_F_BOTH,
00085 L4_IRQ_F_MASK = 0xf,
00086
00088 L4_IRQ_F_SET_WAKEUP = 0x10,
00089 L4_IRQ_F_CLEAR_WAKEUP = 0x20,
00090 };
00091
00092
00097 enum L4_icu_opcode
00098 {
00104 L4_ICU_OP_BIND = 0,
00105
00111 L4_ICU_OP_UNBIND = 1,
00112
00118 L4_ICU_OP_INFO = 2,
00119
00125 L4_ICU_OP_MSI_INFO = 3,
00126
00132 L4_ICU_OP_UNMASK = 4,
00133
00139 L4_ICU_OP_MASK = 5,
00140
00146 L4_ICU_OP_SET_MODE = 6,
00147 };
00148
00149 enum L4_icu_ctl_op
00150 {
00151 L4_ICU_CTL_UNMASK = 0,
00152 L4_ICU_CTL_MASK = 1
00153 };
00154
00155
00163 typedef struct l4_icu_info_t
00164 {
00170 unsigned features;
00171
00175 unsigned nr_irqs;
00176
00180 unsigned nr_msis;
00181 } l4_icu_info_t;
00182
00184 typedef struct l4_icu_msi_info_t
00185 {
00187 l4_uint64_t msi_addr;
00189 l4_uint32_t msi_data;
00190 } l4_icu_msi_info_t;
00191

```

```

00219 L4_INLINE l4_msgtag_t
00220 l4_icu_bind(l4_cap_idx_t icu, unsigned irqnum, l4_cap_idx_t irq) L4_NOTHROW;
00221
00222 L4_INLINE l4_msgtag_t
00223 l4_icu_bind_u(l4_cap_idx_t icu, unsigned irqnum, l4_cap_idx_t irq,
00230 l4_utcb_t *utcb) L4_NOTHROW;
00231
00242 L4_INLINE l4_msgtag_t
00243 l4_icu_unbind(l4_cap_idx_t icu, unsigned irqnum, l4_cap_idx_t irq) L4_NOTHROW;
00244
00251 L4_INLINE l4_msgtag_t
00252 l4_icu_unbind_u(l4_cap_idx_t icu, unsigned irqnum, l4_cap_idx_t irq,
00253 l4_utcb_t *utcb) L4_NOTHROW;
00254
00265 L4_INLINE l4_msgtag_t
00266 l4_icu_set_mode(l4_cap_idx_t icu, unsigned irqnum, l4_umword_t mode) L4_NOTHROW;
00267
00274 L4_INLINE l4_msgtag_t
00275 l4_icu_set_mode_u(l4_cap_idx_t icu, unsigned irqnum, l4_umword_t mode,
00276 l4_utcb_t *utcb) L4_NOTHROW;
00277
00286 L4_INLINE l4_msgtag_t
00287 l4_icu_info(l4_cap_idx_t icu, l4_icu_info_t *info) L4_NOTHROW;
00288
00295 L4_INLINE l4_msgtag_t
00296 l4_icu_info_u(l4_cap_idx_t icu, l4_icu_info_t *info,
00297 l4_utcb_t *utcb) L4_NOTHROW;
00298
00305 L4_INLINE l4_msgtag_t
00306 l4_icu_msi_info(l4_cap_idx_t icu, unsigned irqnum, l4_uint64_t source,
00307 l4_icu_msi_info_t *msi_info) L4_NOTHROW;
00308
00315 L4_INLINE l4_msgtag_t
00316 l4_icu_msi_info_u(l4_cap_idx_t icu, unsigned irqnum, l4_uint64_t source,
00317 l4_icu_msi_info_t *msi_info, l4_utcb_t *utcb) L4_NOTHROW;
00318
00319
00337 L4_INLINE l4_msgtag_t
00338 l4_icu_unmask(l4_cap_idx_t icu, unsigned irqnum, l4_umword_t *label,
00339 l4_timeout_t to) L4_NOTHROW;
00340
00347 L4_INLINE l4_msgtag_t
00348 l4_icu_unmask_u(l4_cap_idx_t icu, unsigned irqnum, l4_umword_t *label,
00349 l4_timeout_t to, l4_utcb_t *utcb) L4_NOTHROW;
00350
00368 L4_INLINE l4_msgtag_t
00369 l4_icu_mask(l4_cap_idx_t icu, unsigned irqnum, l4_umword_t *label,
00370 l4_timeout_t to) L4_NOTHROW;
00371
00378 L4_INLINE l4_msgtag_t
00379 l4_icu_mask_u(l4_cap_idx_t icu, unsigned irqnum, l4_umword_t *label,
00380 l4_timeout_t to, l4_utcb_t *utcb) L4_NOTHROW;
00381
00385 L4_INLINE l4_msgtag_t
00386 l4_icu_control_u(l4_cap_idx_t icu, unsigned irqnum, unsigned op,
00387 l4_umword_t *label, l4_timeout_t to,
00388 l4_utcb_t *utcb) L4_NOTHROW;
00389
00390
00391 /*****
00392 * Implementations
00393 */
00394
00395 L4_INLINE l4_msgtag_t
00396 l4_icu_bind_u(l4_cap_idx_t icu, unsigned irqnum, l4_cap_idx_t irq,
00397 l4_utcb_t *utcb) L4_NOTHROW
00398 {
00399 l4_msg_regs_t *m = l4_utcb_mr_u(utcb);
00400 m->mr[0] = L4_ICU_OP_BIND;
00401 m->mr[1] = irqnum;
00402 m->mr[2] = l4_map_obj_control(0, 0);
00403 m->mr[3] = l4_obj_fpage(irq, 0, L4_CAP_FPAGE_RWS).raw;
00404 return l4_ipc_call(icu, utcb, l4_msgtag(L4_PROTO_IRQ, 2, 1, 0), L4_IPC_NEVER);
00405 }
00406
00407 L4_INLINE l4_msgtag_t
00408 l4_icu_unbind_u(l4_cap_idx_t icu, unsigned irqnum, l4_cap_idx_t irq,
00409 l4_utcb_t *utcb) L4_NOTHROW
00410 {
00411 l4_msg_regs_t *m = l4_utcb_mr_u(utcb);
00412 m->mr[0] = L4_ICU_OP_UNBIND;
00413 m->mr[1] = irqnum;
00414 m->mr[2] = l4_map_obj_control(0, 0);
00415 m->mr[3] = l4_obj_fpage(irq, 0, L4_CAP_FPAGE_RWS).raw;
00416 return l4_ipc_call(icu, utcb, l4_msgtag(L4_PROTO_IRQ, 2, 1, 0), L4_IPC_NEVER);
00417 }
00418

```

```

00419 L4_INLINE l4_msgtag_t
00420 l4_icu_info_u(l4_cap_idx_t icu, l4_icu_info_t *info,
00421 l4_utcb_t *utcb) L4_NOTHROW
00422 {
00423 l4_msgtag_t res;
00424 l4_msg_regs_t *m = l4_utcb_mr_u(utcb);
00425 m->mr[0] = L4_ICU_OP_INFO;
00426 res = l4_ipc_call(icu, utcb, l4_msgtag(L4_PROTO_IRQ, 1, 0, 0), L4_IPC_NEVER);
00427 info->features = m->mr[0];
00428 info->nr_irqs = m->mr[1];
00429 info->nr_msis = m->mr[2];
00430 return res;
00431 }
00432
00433 L4_INLINE l4_msgtag_t
00434 l4_icu_msi_info_u(l4_cap_idx_t icu, unsigned irqnum, l4_uint64_t source,
00435 l4_icu_msi_info_t *msi_info, l4_utcb_t *utcb) L4_NOTHROW
00436 {
00437 l4_msgtag_t res;
00438 l4_msg_regs_t *m = l4_utcb_mr_u(utcb);
00439 m->mr[0] = L4_ICU_OP_MSI_INFO;
00440 m->mr[1] = irqnum;
00441 m->mr64[l4_utcb_mr64_idx(2)] = source;
00442 res = l4_ipc_call(icu, utcb, l4_msgtag(L4_PROTO_IRQ,
00443 2 + 1 * sizeof(l4_uint64_t)
00444 / sizeof(l4_umword_t),
00445 0, 0), L4_IPC_NEVER);
00446 if (L4_UNLIKELY(l4_msgtag_has_error(res)))
00447 return res;
00448
00449 if (L4_UNLIKELY(l4_msgtag_words(res) * sizeof(l4_umword_t) < sizeof(*msi_info)))
00450 return res;
00451
00452 __builtin_memcpy(msi_info, &m->mr[0], sizeof(*msi_info));
00453 return res;
00454 }
00455
00456 L4_INLINE l4_msgtag_t
00457 l4_icu_set_mode_u(l4_cap_idx_t icu, unsigned irqnum, l4_umword_t mode,
00458 l4_utcb_t *utcb) L4_NOTHROW
00459 {
00460 l4_msg_regs_t *mr = l4_utcb_mr_u(utcb);
00461 mr->mr[0] = L4_ICU_OP_SET_MODE;
00462 mr->mr[1] = irqnum;
00463 mr->mr[2] = mode;
00464 return l4_ipc_call(icu, utcb, l4_msgtag(L4_PROTO_IRQ, 3, 0, 0), L4_IPC_NEVER);
00465 }
00466
00467 L4_INLINE l4_msgtag_t
00468 l4_icu_control_u(l4_cap_idx_t icu, unsigned irqnum, unsigned op,
00469 l4_umword_t *label, l4_timeout_t to,
00470 l4_utcb_t *utcb) L4_NOTHROW
00471 {
00472 l4_msg_regs_t *m = l4_utcb_mr_u(utcb);
00473 m->mr[0] = L4_ICU_OP_UNMASK + op;
00474 m->mr[1] = irqnum;
00475 if (label)
00476 return l4_ipc_send_and_wait(icu, utcb, l4_msgtag(L4_PROTO_IRQ, 2, 0, 0),
00477 label, to);
00478 else
00479 return l4_ipc_send(icu, utcb, l4_msgtag(L4_PROTO_IRQ, 2, 0, 0), to);
00480 }
00481
00482 L4_INLINE l4_msgtag_t
00483 l4_icu_mask_u(l4_cap_idx_t icu, unsigned irqnum, l4_umword_t *label,
00484 l4_timeout_t to, l4_utcb_t *utcb) L4_NOTHROW
00485 { return l4_icu_control_u(icu, irqnum, L4_ICU_CTL_MASK, label, to, utcb); }
00486
00487 L4_INLINE l4_msgtag_t
00488 l4_icu_unmask_u(l4_cap_idx_t icu, unsigned irqnum, l4_umword_t *label,
00489 l4_timeout_t to, l4_utcb_t *utcb) L4_NOTHROW
00490 { return l4_icu_control_u(icu, irqnum, L4_ICU_CTL_UNMASK, label, to, utcb); }
00491
00492
00493
00494
00495 L4_INLINE l4_msgtag_t
00496 l4_icu_bind(l4_cap_idx_t icu, unsigned irqnum, l4_cap_idx_t irq) L4_NOTHROW
00497 { return l4_icu_bind_u(icu, irqnum, irq, l4_utcb()); }
00498
00499 L4_INLINE l4_msgtag_t
00500 l4_icu_unbind(l4_cap_idx_t icu, unsigned irqnum, l4_cap_idx_t irq) L4_NOTHROW
00501 { return l4_icu_unbind_u(icu, irqnum, irq, l4_utcb()); }
00502
00503 L4_INLINE l4_msgtag_t
00504 l4_icu_info(l4_cap_idx_t icu, l4_icu_info_t *info) L4_NOTHROW
00505 { return l4_icu_info_u(icu, info, l4_utcb()); }

```

```

00506
00507 L4_INLINE l4_msgtag_t
00508 l4_icu_msi_info(l4_cap_idx_t icu, unsigned irqnum, l4_uint64_t source,
00509 l4_icu_msi_info_t *msi_info) L4_NOTHROW
00510 { return l4_icu_msi_info_u(icu, irqnum, source, msi_info, l4_utcb()); }
00511
00512 L4_INLINE l4_msgtag_t
00513 l4_icu_unmask(l4_cap_idx_t icu, unsigned irqnum, l4_umword_t *label,
00514 l4_timeout_t to) L4_NOTHROW
00515 { return l4_icu_control_u(icu, irqnum, L4_ICU_CTL_UNMASK, label, to, l4_utcb()); }
00516
00517 L4_INLINE l4_msgtag_t
00518 l4_icu_mask(l4_cap_idx_t icu, unsigned irqnum, l4_umword_t *label,
00519 l4_timeout_t to) L4_NOTHROW
00520 { return l4_icu_control_u(icu, irqnum, L4_ICU_CTL_MASK, label, to, l4_utcb()); }
00521
00522 L4_INLINE l4_msgtag_t
00523 l4_icu_set_mode(l4_cap_idx_t icu, unsigned irqnum, l4_umword_t mode) L4_NOTHROW
00524 {
00525 return l4_icu_set_mode_u(icu, irqnum, mode, l4_utcb());
00526 }

```

## 17.521 iommu

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /* \file
00003 * IO-MMU interface description.
00004 */
00005 #pragma once
00006
00007 #include <l4/sys/cxx/ipc_iface>
00008
00009 namespace L4 {
00021 class Iommu :
00022 public Kobject_x<Iommu, Proto_t<L4_PROTO_IOMMU>, Type_info::Demand_t<1> >
00023 {
00024 public:
00037 L4_INLINE_RPC(
00038 l4_msgtag_t, bind, (l4_uint64_t src_id, Ipc::Cap<Task> dma_space));
00039
00050 L4_INLINE_RPC(
00051 l4_msgtag_t, unbind, (l4_uint64_t src_id, Ipc::Cap<Task> dma_space));
00052
00053 typedef Typeid::Rpc_code<l4_umword_t>::F<bind_t, unbind_t> Rpcs;
00054 };
00055
00056 }

```

## 17.522 ipc.h

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00009 * Frank Mehnert <fm3@os.inf.tu-dresden.de>,
00010 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00011 * economic rights: Technische Universität Dresden (Germany)
00012 *
00013 * License: see LICENSE.spdx (in this directory or the directories above)
00014 */
00015 #ifndef __L4SYS__INCLUDE__ARCH_AMD64__L4API_L4F__IPC_H__
00016 #define __L4SYS__INCLUDE__ARCH_AMD64__L4API_L4F__IPC_H__
00017
00018 #include_next <l4/sys/ipc.h>
00019
00020 L4_INLINE l4_msgtag_t
00021 l4_ipc(l4_cap_idx_t dest, l4_utcb_t *utcb,
00022 l4_umword_t flags,
00023 l4_umword_t slabel,
00024 l4_msgtag_t tag,
00025 l4_umword_t *rlabel,
00026 l4_timeout_t timeout) L4_NOTHROW
00027 {
00028 l4_umword_t dummy, dummy2;
00029 register l4_umword_t to __asm__("r8") = timeout.raw;
00030
00031 (void)utcb;
00032 }

```

```

00033 __asm__ __volatile__
00034 ("syscall"
00035 :
00036 "=d" (dummy2),
00037 "=S" (slabel),
00038 "=D" (dummy),
00039 "=a" (tag.raw)
00040 :
00041 "S" (slabel),
00042 "r" (to),
00043 "a" (tag.raw),
00044 "d" (dest | flags)
00045 :
00046 "memory", "cc", "rcx", "r11", "r15"
00047);
00048
00049 if (rlabel)
00050 *rlabel = slabel;
00051
00052 return tag;
00053 }
00054
00055 #endif /* ! __L4SYS__INCLUDE__ARCH_AMD64__L4API_L4F__IPC_H__ */

```

## 17.523 ipc.h

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #pragma once
00014
00015 #include_next <l4/sys/ipc.h>
00016
00017 #ifdef __GNUC__
00018
00019 #include <l4/sys/compiler.h>
00020 #include <l4/sys/syscall_defs.h>
00021
00022 L4_INLINE l4_msgtag_t
00023 l4_ipc(l4_cap_idx_t dest, l4_utcb_t *utcb,
00024 l4_umword_t flags,
00025 l4_umword_t slabel,
00026 l4_msgtag_t tag,
00027 l4_umword_t *rlabel,
00028 l4_timeout_t timeout) L4_NOTHROW
00029 {
00030 register l4_umword_t _dest __asm__("r2") = dest | flags;
00031 register l4_umword_t _timeout __asm__("r3") = timeout.raw;
00032 register l4_umword_t _tag __asm__("r0") = tag.raw;
00033 register l4_umword_t _label __asm__("r4") = slabel;
00034 (void)utcb;
00035
00036 __asm__ __volatile__
00037 (" .type __l4_sys_syscall, #function\n"
00038 "mov r5, %[sc] \n"
00039 "blx __l4_sys_syscall \n"
00040 :
00041 "+r" (_dest),
00042 "+r" (_timeout),
00043 "+r" (_label),
00044 "+r" (_tag)
00045 :
00046 [sc] "i" (L4_SYSCALL_INVOKE)
00047 :
00048 "cc", "memory", "r5", "ip", "lr");
00049
00050 if (rlabel)
00051 *rlabel = _label;
00052 tag.raw = _tag;
00053
00054 return tag;
00055 }
00056
00057 #endif // __GNUC__

```

## 17.524 ipc.h

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #pragma once
00014
00015 #include_next <l4/sys/ipc.h>
00016
00017 #ifdef __GNUC__
00018
00019 #include <l4/sys/compiler.h>
00020
00021 L4_BEGIN_DECLS
00022
00023 struct __l4_sys_syscall_res
00024 {
00025 l4_mword_t tag;
00026 l4_umword_t label;
00027 };
00028
00029 extern struct __l4_sys_syscall_res
00030 __l4_sys_syscall(l4_mword_t tag,
00031 l4_umword_t slabel,
00032 l4_umword_t dest,
00033 l4_umword_t timeout) L4_NOTHROW;
00034
00035 L4_END_DECLS
00036
00037 L4_INLINE l4_msgtag_t
00038 l4_ipc(l4_cap_idx_t dest, l4_utcb_t *utcb,
00039 l4_umword_t flags,
00040 l4_umword_t slabel,
00041 l4_msgtag_t tag,
00042 l4_umword_t *rlabel,
00043 l4_timeout_t timeout) L4_NOTHROW
00044 {
00045 // No need for memory clobbers. The compiler has to assume that all global
00046 // data is read/written because __l4_sys_syscall is implemented in a
00047 // different translation unit.
00048 struct __l4_sys_syscall_res res
00049 = __l4_sys_syscall(tag.raw, slabel, dest | flags, timeout.raw);
00050
00051 (void)utcb;
00052
00053 if (rlabel)
00054 *rlabel = res.label;
00055 tag.raw = res.tag;
00056
00057 return tag;
00058 }
00059
00060 #endif // __GNUC__

```

## 17.525 l4/sys/ipc.h File Reference

Common IPC interface.

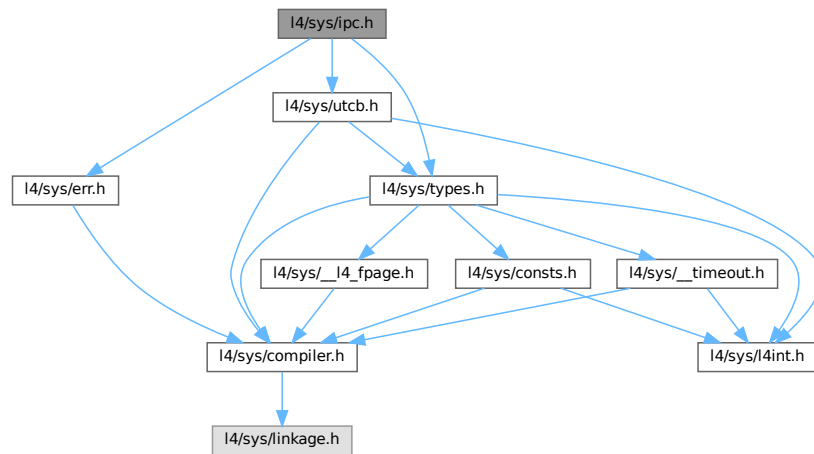
```

#include <l4/sys/types.h>
#include <l4/sys/utcb.h>
#include <l4/sys/err.h>

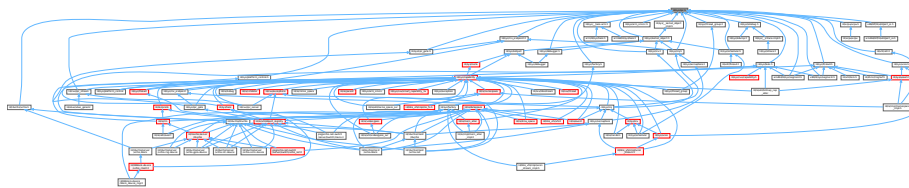
```



Include dependency graph for ipc.h:



This graph shows which files directly or indirectly include this file:



## Enumerations

- enum `l4_ipc_tcr_error_t` {  
`L4_IPC_ERROR_MASK` = 0x1F , `L4_IPC_SND_ERR_MASK` = 0x01 , `L4_IPC_ENOT_EXISTENT` = 0x04 ,  
`L4_IPC_RETIMEOUT` = 0x03 ,  
`L4_IPC_SETIMEOUT` = 0x02 , `L4_IPC_RECANCELED` = 0x07 , `L4_IPC_SECANCELED` = 0x06 ,  
`L4_IPC_REMAPFAILED` = 0x11 ,  
`L4_IPC_SEMAPFAILED` = 0x10 , `L4_IPC_RESNDPFTO` = 0x0b , `L4_IPC_SESNDPFTO` = 0x0a ,  
`L4_IPC_RERCVPFTO` = 0x0d ,  
`L4_IPC_SERCVPFTO` = 0x0c , `L4_IPC_REABORTED` = 0x0f , `L4_IPC_SEABORTED` = 0x0e ,  
`L4_IPC_REMSGCUT` = 0x09 ,  
`L4_IPC_SEMSGCUT` = 0x08 }

*Error codes in the error TCR.*

## Functions

- `l4_umword_t l4_ipc_error(l4_msgtag_t tag, l4_utcb_t *utcb) L4_NOTHROW`  
*Get the IPC error code for an IPC operation.*
- `long l4_error(l4_msgtag_t tag) L4_NOTHROW`  
*Get IPC error code if any or message tag label otherwise for an IPC call.*
- `int l4_ipc_is_snd_error(l4_utcb_t *utcb) L4_NOTHROW`  
*Returns whether an error occurred in send phase of an invocation.*

- `int l4_ipc_is_rcv_error (l4_utcb_t *utcb) L4_NOTHROW`  
*Returns whether an error occurred in receive phase of an invocation.*
- `int l4_ipc_error_code (l4_utcb_t *utcb) L4_NOTHROW`  
*Get the error condition of the last invocation from the TCR.*
- `long l4_ipc_to_errno (unsigned long ipc_error_code) L4_NOTHROW`  
*Get a negative error code for the given IPC error code.*
- `l4_msgtag_t l4_ipc_send (l4_cap_idx_t dest, l4_utcb_t *utcb, l4_msgtag_t tag, l4_timeout_t timeout) L4_NOTHROW`  
*Send a message to an object (do **not** wait for a reply).*
- `l4_msgtag_t l4_ipc_wait (l4_utcb_t *utcb, l4_umword_t *label, l4_timeout_t timeout) L4_NOTHROW`  
*Wait for an incoming message from any possible sender.*
- `l4_msgtag_t l4_ipc_receive (l4_cap_idx_t object, l4_utcb_t *utcb, l4_timeout_t timeout) L4_NOTHROW`  
*Wait for a message from a specific source.*
- `l4_msgtag_t l4_ipc_call (l4_cap_idx_t object, l4_utcb_t *utcb, l4_msgtag_t tag, l4_timeout_t timeout) L4_NOTHROW`  
*Object call (usual invocation).*
- `l4_msgtag_t l4_ipc_reply_and_wait (l4_utcb_t *utcb, l4_msgtag_t tag, l4_umword_t *label, l4_timeout_t timeout) L4_NOTHROW`  
*Reply and wait operation (uses the reply capability).*
- `l4_msgtag_t l4_ipc_send_and_wait (l4_cap_idx_t dest, l4_utcb_t *utcb, l4_msgtag_t tag, l4_umword_t *label, l4_timeout_t timeout) L4_NOTHROW`  
*Send a message and do an open wait.*
- `l4_msgtag_t l4_ipc (l4_cap_idx_t dest, l4_utcb_t *utcb, l4_umword_t flags, l4_umword_t slabel, l4_msgtag_t tag, l4_umword_t *rlabel, l4_timeout_t timeout) L4_NOTHROW`  
*Generic L4 object invocation.*
- `l4_msgtag_t l4_ipc_sleep (l4_timeout_t timeout) L4_NOTHROW`  
*Sleep for an amount of time.*
- `l4_msgtag_t l4_ipc_sleep_ms (l4_uint32_t ms) L4_NOTHROW`  
*Sleep for a certain amount of milliseconds.*
- `l4_msgtag_t l4_ipc_sleep_us (l4_uint64_t us) L4_NOTHROW`  
*Sleep for a certain amount of microseconds.*
- `int l4_sndfpage_add (l4_fpage_t const snd_fpage, unsigned long snd_base, l4_msgtag_t *tag) L4_NOTHROW`  
*Add a flexpage to be sent to the UTCB.*

## 17.525.1 Detailed Description

Common IPC interface.

Definition in file [ipc.h](#).

## 17.525.2 Function Documentation

### 17.525.2.1 l4\_ipc\_to\_errno()

```
long l4_ipc_to_errno (
 unsigned long ipc_error_code) [inline]
```

Get a negative error code for the given IPC error code.

#### Parameters

---

|                       |                                                                                                               |
|-----------------------|---------------------------------------------------------------------------------------------------------------|
| <i>ipc_error_code</i> | IPC error code as delivered by the kernel. (or returned by the <a href="#">l4_ipc_error_code()</a> function). |
|-----------------------|---------------------------------------------------------------------------------------------------------------|

## Returns

negative error code in the range of [L4\\_EIPC\\_LO](#) to [L4\\_EIPC\\_HI](#).

Definition at line 561 of file [ipc.h](#).

References [L4\\_EIPC\\_LO](#), [L4\\_INLINE](#), and [L4\\_NOTHROW](#).

## 17.526 ipc.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00009 * Björn Döbel <doebel@os.inf.tu-dresden.de>,
00010 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00011 * economic rights: Technische Universität Dresden (Germany)
00012 *
00013 * License: see LICENSE.spdx (in this directory or the directories above)
00014 */
00015 #ifndef __L4SYS__INCLUDE__L4API_FIASCO__IPC_H__
00016 #define __L4SYS__INCLUDE__L4API_FIASCO__IPC_H__
00017
00018 #include <l4/sys/types.h>
00019 #include <l4/sys/utcb.h>
00020 #include <l4/sys/err.h>
00021
00061
00062 /*****
00063 *** IPC result checking
00064 *****/
00065
00073
00081 enum l4_ipc_tcr_error_t
00082 {
00083 L4_IPC_ERROR_MASK = 0x1F,
00084 L4_IPC_SND_ERR_MASK = 0x01,
00085
00086 L4_IPC_ENOT_EXISTENT = 0x04,
00089 L4_IPC_RETIMEOUT = 0x03,
00092 L4_IPC_SETIMEOUT = 0x02,
00095 L4_IPC_RECANCELED = 0x07,
00098 L4_IPC_SECANCELED = 0x06,
00101 L4_IPC_REMAPFAILED = 0x11,
00105 L4_IPC_SEMAPFAILED = 0x10,
00108 L4_IPC_RESNDPFTO = 0x0b,
00112 L4_IPC_SESNDPFTO = 0x0a,
00116 L4_IPC_RERCVPFTO = 0x0d,
00120 L4_IPC_SERCVPFTO = 0x0c,
00124 L4_IPC_REABORTED = 0x0f,
00127 L4_IPC_SEABORTED = 0x0e,
00136 L4_IPC_REMSGCUT = 0x09,
00137
00143 L4_IPC_SEMSGCUT = 0x08,
00144 };
00145
00146
00157 L4_INLINE l4_umword_t
00158 l4_ipc_error(l4_msgtag_t tag, l4_utcb_t *utcb) L4_NOTHROW;
00159
00160
00177 L4_INLINE long
00178 l4_error(l4_msgtag_t tag) L4_NOTHROW;
00179
00180 L4_INLINE long
00181 l4_error_u(l4_msgtag_t tag, l4_utcb_t *utcb) L4_NOTHROW;
00182
00183 /*****
00184 *** IPC results
00185 *****/

```

```

00186
00196 L4_INLINE int l4_ipc_is_snd_error(l4_utcb_t *utcb) L4_NOTHROW;
00197
00207 L4_INLINE int l4_ipc_is_rcv_error(l4_utcb_t *utcb) L4_NOTHROW;
00208
00218 L4_INLINE int l4_ipc_error_code(l4_utcb_t *utcb) L4_NOTHROW;
00219
00226 L4_INLINE long l4_ipc_to_errno(unsigned long ipc_error_code) L4_NOTHROW;
00227
00228
00229 /*****
00230 *** IPC calls
00231 *****/
00232
00261 L4_INLINE l4_msgtag_t
00262 l4_ipc_send(l4_cap_idx_t dest, l4_utcb_t *utcb, l4_msgtag_t tag,
00263 l4_timeout_t timeout) L4_NOTHROW;
00264
00265
00292 L4_INLINE l4_msgtag_t
00293 l4_ipc_wait(l4_utcb_t *utcb, l4_umword_t *label,
00294 l4_timeout_t timeout) L4_NOTHROW;
00295
00296
00321 L4_INLINE l4_msgtag_t
00322 l4_ipc_receive(l4_cap_idx_t object, l4_utcb_t *utcb,
00323 l4_timeout_t timeout) L4_NOTHROW;
00324
00355 L4_INLINE l4_msgtag_t
00356 l4_ipc_call(l4_cap_idx_t object, l4_utcb_t *utcb, l4_msgtag_t tag,
00357 l4_timeout_t timeout) L4_NOTHROW;
00358
00359
00385 L4_INLINE l4_msgtag_t
00386 l4_ipc_reply_and_wait(l4_utcb_t *utcb, l4_msgtag_t tag,
00387 l4_umword_t *label, l4_timeout_t timeout) L4_NOTHROW;
00388
00417 L4_INLINE l4_msgtag_t
00418 l4_ipc_send_and_wait(l4_cap_idx_t dest, l4_utcb_t *utcb, l4_msgtag_t tag,
00419 l4_umword_t *label, l4_timeout_t timeout) L4_NOTHROW;
00420
00426
00427 #if 0
00438 L4_INLINE l4_msgtag_t
00439 l4_ipc_wait_next_period(l4_utcb_t *utcb,
00440 l4_umword_t *label,
00441 l4_timeout_t timeout);
00442
00443 #endif
00444
00464 L4_ALWAYS_INLINE l4_msgtag_t
00465 l4_ipc(l4_cap_idx_t dest,
00466 l4_utcb_t *utcb,
00467 l4_umword_t flags,
00468 l4_umword_t slabel,
00469 l4_msgtag_t tag,
00470 l4_umword_t *rlabel,
00471 l4_timeout_t timeout) L4_NOTHROW;
00472
00489 L4_INLINE l4_msgtag_t
00490 l4_ipc_sleep(l4_timeout_t timeout) L4_NOTHROW;
00491
00508 L4_INLINE l4_msgtag_t
00509 l4_ipc_sleep_ms(l4_uint32_t ms) L4_NOTHROW;
00510
00527 L4_INLINE l4_msgtag_t
00528 l4_ipc_sleep_us(l4_uint64_t us) L4_NOTHROW;
00529
00544 L4_INLINE int
00545 l4_sndfpage_add(l4_fpage_t const snd_fpage, unsigned long snd_base,
00546 l4_msgtag_t *tag) L4_NOTHROW;
00547
00548 /*
00549 * \internal
00550 * \ingroup l4_ipc_api
00551 */
00552 L4_INLINE int
00553 l4_sndfpage_add_u(l4_fpage_t const snd_fpage, unsigned long snd_base,
00554 l4_msgtag_t *tag, l4_utcb_t *utcb) L4_NOTHROW;
00555
00556
00557 /*****
00558 * Implementations
00559 *****/
00560
00561 L4_INLINE long l4_ipc_to_errno(unsigned long ipc_error_code) L4_NOTHROW
00562 { return -(L4_EIPC_LO + ipc_error_code); }

```

```

00563
00564 L4_INLINE l4_msgtag_t
00565 l4_ipc_call(l4_cap_idx_t object, l4_utcb_t *utcb, l4_msgtag_t tag,
00566 l4_timeout_t timeout) L4_NOTHROW
00567 {
00568 return l4_ipc(object, utcb, L4_SYSF_CALL, 0, tag, 0, timeout);
00569 }
00570
00571 L4_INLINE l4_msgtag_t
00572 l4_ipc_reply_and_wait(l4_utcb_t *utcb, l4_msgtag_t tag,
00573 l4_umword_t *label, l4_timeout_t timeout) L4_NOTHROW
00574 {
00575 return l4_ipc(L4_INVALID_CAP, utcb, L4_SYSF_REPLY_AND_WAIT, 0, tag, label, timeout);
00576 }
00577
00578 L4_INLINE l4_msgtag_t
00579 l4_ipc_send_and_wait(l4_cap_idx_t dest, l4_utcb_t *utcb, l4_msgtag_t tag,
00580 l4_umword_t *label, l4_timeout_t timeout) L4_NOTHROW
00581 {
00582 return l4_ipc(dest, utcb, L4_SYSF_SEND_AND_WAIT, 0, tag, label, timeout);
00583 }
00584
00585 L4_INLINE l4_msgtag_t
00586 l4_ipc_send(l4_cap_idx_t dest, l4_utcb_t *utcb, l4_msgtag_t tag,
00587 l4_timeout_t timeout) L4_NOTHROW
00588 {
00589 return l4_ipc(dest, utcb, L4_SYSF_SEND, 0, tag, 0, timeout);
00590 }
00591
00592 L4_INLINE l4_msgtag_t
00593 l4_ipc_wait(l4_utcb_t *utcb, l4_umword_t *label,
00594 l4_timeout_t timeout) L4_NOTHROW
00595 {
00596 l4_msgtag_t t;
00597 t.raw = 0;
00598 return l4_ipc(L4_INVALID_CAP, utcb, L4_SYSF_WAIT, 0, t, label, timeout);
00599 }
00600
00601 L4_INLINE l4_msgtag_t
00602 l4_ipc_receive(l4_cap_idx_t object, l4_utcb_t *utcb,
00603 l4_timeout_t timeout) L4_NOTHROW
00604 {
00605 l4_msgtag_t t;
00606 t.raw = 0;
00607 return l4_ipc(object, utcb, L4_SYSF_RECV, 0, t, 0, timeout);
00608 }
00609
00610 L4_INLINE l4_msgtag_t
00611 l4_ipc_sleep(l4_timeout_t timeout) L4_NOTHROW
00612 { return l4_ipc_receive(L4_INVALID_CAP, NULL, timeout); }
00613
00614 L4_INLINE l4_msgtag_t
00615 l4_ipc_sleep_ms(l4_uint32_t ms) L4_NOTHROW
00616 {
00617 l4_uint64_t us = ms * 1000ULL; // cannot overflow because ms < 2^32
00618 return l4_ipc_sleep(l4_timeout(L4_IPC_TIMEOUT_NEVER, l4_timeout_from_us(us)));
00619 }
00620
00621 L4_INLINE l4_msgtag_t
00622 l4_ipc_sleep_us(l4_uint64_t us) L4_NOTHROW
00623 {
00624 return l4_ipc_sleep(l4_timeout(L4_IPC_TIMEOUT_NEVER,
00625 l4_timeout_from_us(us)));
00626 }
00627
00628 L4_INLINE l4_umword_t
00629 l4_ipc_error(l4_msgtag_t tag, l4_utcb_t *utcb) L4_NOTHROW
00630 {
00631 if (L4_LIKELY(!l4_msgtag_has_error(tag)))
00632 return 0;
00633 return l4_utcb_tcr_u(utcb)->error & L4_IPC_ERROR_MASK;
00634 }
00635
00636 L4_INLINE long
00637 l4_error_u(l4_msgtag_t tag, l4_utcb_t *u) L4_NOTHROW
00638 {
00639 if (L4_UNLIKELY(l4_msgtag_has_error(tag)))
00640 return l4_ipc_to_errno(l4_utcb_tcr_u(u)->error & L4_IPC_ERROR_MASK);
00641 return l4_msgtag_label(tag);
00642 }
00643
00644
00645 L4_INLINE long
00646 l4_error(l4_msgtag_t tag) L4_NOTHROW
00647 {
00648 return l4_error_u(tag, l4_utcb());
00649 }

```

```

00650
00651
00652 L4_INLINE int l4_ipc_is_snd_error(l4_utcb_t *u) L4_NOTHROW
00653 { return (l4_utcb_tcr_u(u)->error & 1) == 0; }
00654
00655 L4_INLINE int l4_ipc_is_rcv_error(l4_utcb_t *u) L4_NOTHROW
00656 { return l4_utcb_tcr_u(u)->error & 1; }
00657
00658 L4_INLINE int l4_ipc_error_code(l4_utcb_t *u) L4_NOTHROW
00659 { return l4_utcb_tcr_u(u)->error & L4_IPC_ERROR_MASK; }
00660
00661
00662 /*
00663 * \internal
00664 * \ingroup l4_ipc_api
00665 */
00666 L4_INLINE int
00667 l4_sndfpage_add_u(l4_fpage_t const snd_fpage, unsigned long snd_base,
00668 l4_msgtag_t *tag, l4_utcb_t *utcb) L4_NOTHROW
00669 {
00670 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00671 int i = l4_msgtag_words(*tag) + 2 * l4_msgtag_items(*tag);
00672
00673 if (i >= L4_UTCB_GENERIC_DATA_SIZE - 1)
00674 return -L4_ENOMEM;
00675
00676 v->mr[i] = snd_base | L4_ITEM_MAP | L4_ITEM_CONT;
00677 v->mr[i + 1] = snd_fpage.raw;
00678
00679 *tag = l4_msgtag(l4_msgtag_label(*tag), l4_msgtag_words(*tag),
00680 l4_msgtag_items(*tag) + 1, l4_msgtag_flags(*tag));
00681 return 0;
00682 }
00683
00684 L4_INLINE int
00685 l4_sndfpage_add(l4_fpage_t const snd_fpage, unsigned long snd_base,
00686 l4_msgtag_t *tag) L4_NOTHROW
00687 {
00688 return l4_sndfpage_add_u(snd_fpage, snd_base, tag, l4_utcb());
00689 }
00690
00691
00692 #endif /* ! __L4SYS__INCLUDE__L4API_FIASCO__IPC_H__ */

```

## 17.527 x86/i4f/i4/sys/ipc.h File Reference

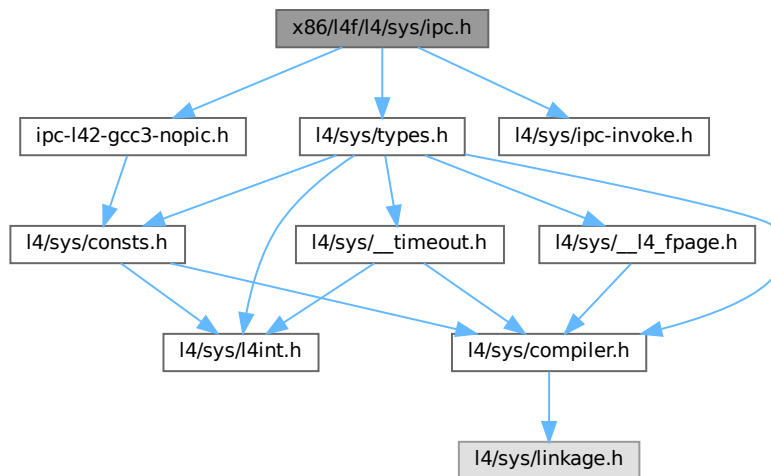
L4 IPC System Calls, x86.

```

#include <l4/sys/types.h>
#include <l4/sys/ipc-invoke.h>
#include "ipc-l42-gcc3-nopic.h"

```

Include dependency graph for ipc.h:



### 17.527.1 Detailed Description

[L4 IPC System Calls, x86.](#)

Definition in file [ipc.h](#).

## 17.528 ipc.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00009 * Lars Reuther <reuther@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #ifndef __L4_IPC_H__
00015 #define __L4_IPC_H__
00016
00017 #include <l4/sys/types.h>
00018
00019 #include_next <l4/sys/ipc.h>
00020
00021 /*****
00022 *** Implementation
00023 *****/
00024
00025 #include <l4/sys/ipc-invoke.h>
00026 #include "ipc-l42-gcc3-nopic.h"
00027
00028 #endif /* !__L4_IPC_H__ */

```





## Namespaces

- namespace [L4](#)  
*L4 low-level kernel interface.*

## 17.529.1 Detailed Description

The C++ IPC gate interface.

Definition in file [ipc\\_gate](#).

## 17.530 ipc\_gate

[Go to the documentation of this file.](#)

```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2009-2010 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #pragma once
00014
00015 #include <l4/sys/ipc_gate.h>
00016 #include <l4/sys/capability>
00017 #include <l4/sys/rcv_endpoint>
00018 #include <l4/sys/cxx/ipc_iface>
00019
00020 namespace L4 {
00021
00022 class Thread;
00023
00085 class L4_EXPORT Ipc_gate :
00086 public Kobject_t<Ipc_gate, Rcv_endpoint, L4_PROTO_KOBJECT,
00087 Type_info::Demand_t<1> >
00088 {
00089 public:
00103 L4_INLINE_RPC_OP(L4_IPC_GATE_GET_INFO_OP,
00104 l4_msgtag_t, get_infos, (l4_umword_t *label));
00105
00106 typedef L4::Typeid::Rpcsys_sys<bind_thread_t, get_infos_t> Rpcsys;
00107 };
00108
00109 }
```

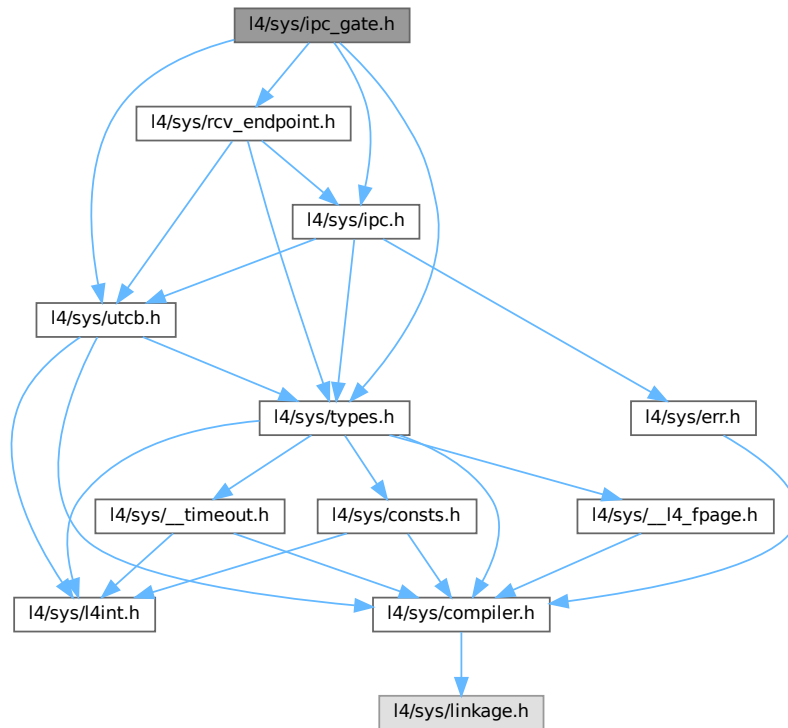
## 17.531 l4/sys/ipc\_gate.h File Reference

The C IPC gate interface, see [L4::ipc\\_gate](#) for the C++ interface.

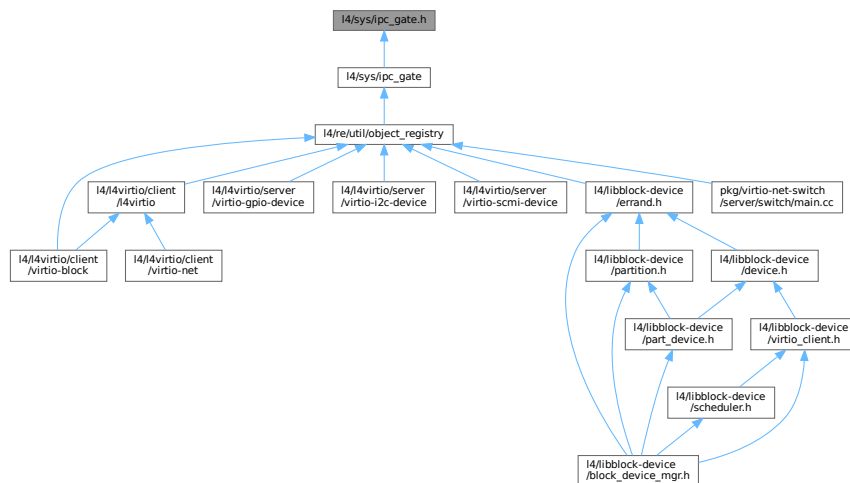
```
#include <l4/sys/utcb.h>
#include <l4/sys/types.h>
#include <l4/sys/rcv_endpoint.h>
```

```
#include <l4/sys/ipc.h>
```

Include dependency graph for ipc\_gate.h:



This graph shows which files directly or indirectly include this file:



## Enumerations

- enum [L4\\_ipc\\_gate\\_ops](#) { [L4\\_IPC\\_GATE\\_BIND\\_OP](#) = 0x10 , [L4\\_IPC\\_GATE\\_GET\\_INFO\\_OP](#) = 0x11 }
- Operations on the IPC-gate.

## Functions

- [l4\\_msgtag\\_t l4\\_ipc\\_gate\\_get\\_infos](#) ([l4\\_cap\\_idx\\_t](#) gate, [l4\\_umword\\_t](#) \*label)

*Get information about the IPC-gate.*

### 17.531.1 Detailed Description

The C IPC gate interface, see [L4::ipc\\_gate](#) for the C++ interface.

IPC gates are used to create secure communication channels between protection domains. An IPC gate can be created using the [Factory](#) interface.

Depending on the permissions of the capability used, an IPC gate forwards IPC to the [Thread](#) the IPC gate is *bound* to (cf. [l4\\_rcv\\_ep\\_bind\\_thread\(\)](#) and [l4\\_rcv\\_ep\\_bind\\_snd\\_destination\(\)](#)). If the capability has the [L4\\_FPAGE\\_C\\_IPCGATE\\_SVR](#) permission, only IPC using a protocol different from the [L4\\_PROTO\\_KOBJECT](#) protocol is forwarded. Without the [L4\\_FPAGE\\_C\\_IPCGATE\\_SVR](#) permission, all IPC is forwarded. The latter is the usual case for a client in a client/server scenario. When not bound to a thread or thread group yet, the forwarded IPC blocks until the IPC gate is bound to a thread or thread group, or the IPC times out.

Forwarded IPC is always forwarded to the userland of the thread the IPC gate is bound to, either directly or indirectly using a thread group. That means, the [Thread](#) interface of that thread is not accessible via an IPC gate. The [IPC-Gate API](#) of an IPC gate is only accessible if the capability used has the [L4\\_FPAGE\\_C\\_IPCGATE\\_SVR](#) permission (cf. previous paragraph). Conversely that means, if the capability used lacks the [L4\\_FPAGE\\_C\\_IPCGATE\\_SVR](#) permission, [IPC-Gate API](#) calls are forwarded to the thread or thread group the IPC gate is bound to instead of being processed by the IPC gate itself. In a client/server scenario, a client should only get IPC gate capabilities without [L4\\_FPAGE\\_C\\_IPCGATE\\_SVR](#) permission so the client cannot tamper with the IPC gate.

When binding an IPC gate to a thread or thread group, a user-defined, kernel protected, machine-word sized payload called the IPC gate's *label* is assigned to the IPC gate (note that the two least significant bits of the label must be zero; cf. [l4\\_rcv\\_ep\\_bind\\_thread\(\)](#) and [l4\\_rcv\\_ep\\_bind\\_snd\\_destination\(\)](#)). When a send-only IPC or call IPC is forwarded via an IPC gate, the label provided by the sender is ignored and replaced by the IPC gate's label where the two least significant bits are set to the [L4\\_CAP\\_FPAGE\\_S](#) and [L4\\_CAP\\_FPAGE\\_W](#) permissions of the capability used. The replaced label is only visible to the thread the IPC gate is bound to upon receive (or to the selected thread from the thread group the IPC gate is bound to). However, the configured label of an IPC gate can also be queried via [l4\\_ipc\\_gate\\_get\\_infos\(\)](#) if the capability used has the [L4\\_FPAGE\\_C\\_IPCGATE\\_SVR](#) permission.

When deleting an IPC gate or when unbinding it from a thread or thread group, the label of IPC already in flight won't be changed. To ensure that no IPC from this IPC gate is received by a thread with an unexpected label, [l4\\_thread\\_modify\\_sender\\_start\(\)](#) shall be used to change the labels of every pending IPC to that gate. This is also required if the label of an already bound IPC gate is changed. It is not necessary after binding the IPC gate to a thread or thread group for the first time.

When binding a currently bound IPC gate to a new thread or thread group, the same label should be used that was used with the old thread. Otherwise the old and the new thread need to synchronize to avoid IPC messages with unexpected labels.

## Include File

```
#include <l4/sys/ipc_gate.h>
```

For the C++ interface refer to the [L4::ipc\\_gate](#) documentation.

## See also

[Object Invocation](#)

Definition in file [ipc\\_gate.h](#).

## 17.532 ipc\_gate.h

[Go to the documentation of this file.](#)

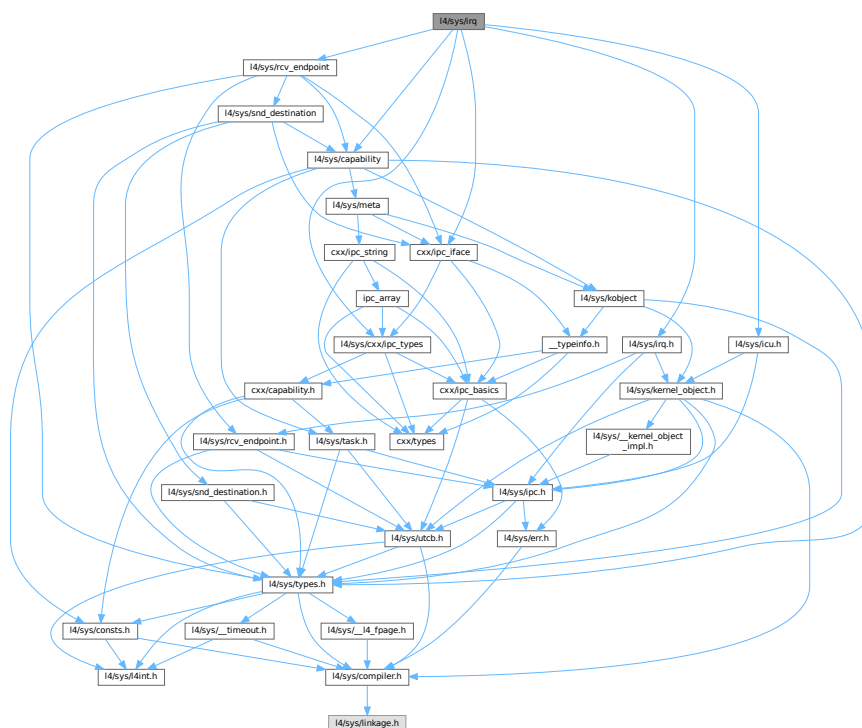
```

00001 /*
00002 * (c) 2009-2010 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00009 #pragma once
00010
00011 #include <l4/sys/utcb.h>
00012 #include <l4/sys/types.h>
00013 #include <l4/sys/rcv_endpoint.h>
00014
00015 L4_INLINE l4_msgtag_t
00016 l4_ipc_gate_get_infos(l4_cap_idx_t gate, l4_umword_t *label);
00017
00018 L4_INLINE l4_msgtag_t
00019 l4_ipc_gate_get_infos_u(l4_cap_idx_t gate, l4_umword_t *label, l4_utcb_t *utcb);
00020
00021 enum L4_ipc_gate_ops
00022 {
00023 L4_IPC_GATE_BIND_OP = 0x10,
00024 L4_IPC_GATE_GET_INFO_OP = 0x11,
00025 };
00026
00027 /* IMPLEMENTATION -----*/
00028
00029 #include <l4/sys/ipc.h>
00030
00031 L4_INLINE l4_msgtag_t
00032 l4_ipc_gate_bind_thread_u(l4_cap_idx_t gate,
00033 l4_cap_idx_t thread, l4_umword_t label,
00034 l4_utcb_t *utcb)
00035 {
00036 return l4_rcv_ep_bind_thread_u(gate, thread, label, utcb);
00037 }
00038
00039 L4_INLINE l4_msgtag_t
00040 l4_ipc_gate_bind_snd_destination_u(l4_cap_idx_t gate,
00041 l4_cap_idx_t snd_dst, l4_umword_t label,
00042 l4_utcb_t *utcb)
00043 {
00044 return l4_rcv_ep_bind_snd_destination_u(gate, snd_dst, label, utcb);
00045 }
00046
00047 L4_INLINE l4_msgtag_t
00048 l4_ipc_gate_get_infos_u(l4_cap_idx_t gate, l4_umword_t *label, l4_utcb_t *utcb)
00049 {
00050 l4_msgtag_t tag;
00051 l4_msg_regs_t *m = l4_utcb_mr_u(utcb);
00052 m->mr[0] = L4_IPC_GATE_GET_INFO_OP;
00053 tag = l4_ipc_call(gate, utcb, l4_msgtag(L4_PROTO_KOBJECT, 1, 0, 0),
00054 L4_IPC_NEVER);
00055 if (!l4_msgtag_has_error(tag) && l4_msgtag_label(tag) >= 0)
00056 *label = m->mr[0];
00057 return tag;
00058 }
00059
00060 L4_INLINE l4_msgtag_t
00061 l4_ipc_gate_bind_thread(l4_cap_idx_t gate, l4_cap_idx_t thread,
00062 l4_umword_t label)
00063 {
00064 return l4_rcv_ep_bind_thread_u(gate, thread, label, l4_utcb());
00065 }
00066
00067 L4_INLINE l4_msgtag_t
00068 l4_ipc_gate_bind_snd_destination(l4_cap_idx_t gate, l4_cap_idx_t snd_dst,
00069 l4_umword_t label)
00070 {
00071 return l4_rcv_ep_bind_snd_destination_u(gate, snd_dst, label, l4_utcb());
00072 }
00073
00074 L4_INLINE l4_msgtag_t
00075 l4_ipc_gate_get_infos(l4_cap_idx_t gate, l4_umword_t *label)
00076 {
00077 return l4_ipc_gate_get_infos_u(gate, label, l4_utcb());
00078 }

```

### 17.533 I4/sys/irq File Reference

```
#include <l4/sys/icu.h>
#include <l4/sys/irq.h>
#include <l4/sys/capability>
#include <l4/sys/rcv_endpoint>
#include <l4/sys/cxx/ipc_iface>
#include <l4/sys/cxx/ipc_types>
Include dependency graph for irq:
```



## Data Structures

- class [L4::Irq\\_eoi](#)  
*Interface for sending an unmask message to an object.*
- struct [L4::Triggerable](#)  
*Interface that allows an object to be triggered by some source.*
- class [L4::Irq](#)  
*C++ [Irq](#) interface, see [IRQs](#) for the C interface.*
- class [L4::Icu](#)  
*C++ [Icu](#) interface, see [Interrupt controller](#) for the C interface.*
- class [L4::Icu::Info](#)  
*This class encapsulates information about an ICU.*

## Namespaces

- namespace [L4](#)  
*[L4](#) low-level kernel interface.*

### 17.533.1 Detailed Description

C++ Irq interface.

Definition in file [irq](#).

## 17.534 irq

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013
00014 #pragma once
00015
00016 #include <l4/sys/icu.h>
00017 #include <l4/sys/irq.h>
00018 #include <l4/sys/capability>
00019 #include <l4/sys/rcv_endpoint>
00020 #include <l4/sys/cxx/ipc_iface>
00021 #include <l4/sys/cxx/ipc_types>
00022
00023 namespace L4 {
00024
00037 class Irq_eoi : public Kobject_0t<Irq_eoi, L4::PROTO_EMPTY>
00038 {
00039 public:
00064 l4_msgtag_t unmask(unsigned irqnum, l4_umword_t *label = 0,
00065 l4_timeout_t to = L4_IPC_NEVER,
00066 l4_utcb_t *utcb = l4_utcb()) noexcept
00067 {
00068 return l4_icu_control_u(cap(), irqnum, L4_ICU_CTL_UNMASK, label, to, utcb);
00069 }
00070 };
00071
00079 struct Triggerable : Kobject_t<Triggerable, Irq_eoi, L4_PROTO_IRQ>
00080 {
00091 l4_msgtag_t trigger(l4_utcb_t *utcb = l4_utcb()) noexcept
00092 { return l4_irq_trigger_u(cap(), utcb); }
00093 };

```

```

00094
00120 class Irq : public Kobject_2t<Irq, Triggerable, Rcv_endpoint, L4_PROTO_IRQ_SENDER>
00121 {
00122 public:
00123 using Triggerable::unmask;
00124
00158 l4_msgtag_t bind_vcpu(L4::Cap<Thread> const &thread, l4_umword_t cfg,
00159 l4_utcb_t *utcb = l4_utcb()) noexcept
00160 { return l4_irq_bind_vcpu_u(cap(), thread.cap(), cfg, utcb); }
00161
00176 l4_msgtag_t detach(l4_utcb_t *utcb = l4_utcb()) noexcept
00177 { return l4_irq_detach_u(cap(), utcb); }
00178
00179
00191 l4_msgtag_t receive(l4_timeout_t timeout = L4_IPC_NEVER,
00192 l4_utcb_t *utcb = l4_utcb()) noexcept
00193 { return l4_irq_receive_u(cap(), timeout, utcb); }
00194
00204 l4_msgtag_t wait(l4_umword_t *label, l4_timeout_t timeout = L4_IPC_NEVER,
00205 l4_utcb_t *utcb = l4_utcb()) noexcept
00206 { return unmask(-1, label, timeout, utcb); }
00207
00221 l4_msgtag_t unmask(l4_utcb_t *utcb = l4_utcb()) noexcept
00222 { return unmask(-1, 0, L4_IPC_NEVER, utcb); }
00223 };
00224
00250 class Icu :
00251 public Kobject_t<Icu, Irq_eoi, L4_PROTO_IRQ,
00252 Type_info::Demand_t<1> >
00253 {
00254 public:
00255 enum Mode
00256 {
00257 F_none = L4_IRQ_F_NONE,
00258 F_level_high = L4_IRQ_F_LEVEL_HIGH,
00259 F_level_low = L4_IRQ_F_LEVEL_LOW,
00260 F_pos_edge = L4_IRQ_F_POS_EDGE,
00261 F_neg_edge = L4_IRQ_F_NEG_EDGE,
00262 F_both_edge = L4_IRQ_F_BOTH_EDGE,
00263 F_mask = L4_IRQ_F_MASK,
00264
00265 F_set_wakeup = L4_IRQ_F_SET_WAKEUP,
00266 F_clear_wakeup = L4_IRQ_F_CLEAR_WAKEUP,
00267 };
00268
00269 enum Flags
00270 {
00271 F_msi = L4_ICU_FLAG_MSI
00272 };
00273
00277 class Info : public l4_icu_info_t
00278 {
00279 public:
00281 bool supports_msi() const noexcept { return features & F_msi; }
00282 };
00283
00310 l4_msgtag_t bind(unsigned irqnum, L4::Cap<Triggerable> irq,
00311 l4_utcb_t *utcb = l4_utcb()) noexcept
00312 { return l4_icu_bind_u(cap(), irqnum, irq.cap(), utcb); }
00313
00314 L4_RPC_NF_OP(
00315 L4_ICU_OP_BIND,
00316 l4_msgtag_t, bind, (l4_umword_t irqnum, Ipc::Cap<Irq> irq)
00317);
00318
00328 l4_msgtag_t unbind(unsigned irqnum, L4::Cap<Triggerable> irq,
00329 l4_utcb_t *utcb = l4_utcb()) noexcept
00330 { return l4_icu_unbind_u(cap(), irqnum, irq.cap(), utcb); }
00331
00332 L4_RPC_NF_OP(
00333 L4_ICU_OP_UNBIND,
00334 l4_msgtag_t, unbind, (l4_umword_t irqnum, Ipc::Cap<Irq> irq)
00335);
00336
00345 l4_msgtag_t info(l4_icu_info_t *info, l4_utcb_t *utcb = l4_utcb()) noexcept
00346 { return l4_icu_info_u(cap(), info, utcb); }
00347
00348 struct _Info { l4_umword_t features, nr_irqs, nr_msis; };
00349 L4_RPC_NF_OP(L4_ICU_OP_INFO, l4_msgtag_t, info, (_Info *info));
00350
00363 L4_INLINE_RPC_OP(L4_ICU_OP_MSI_INFO,
00364 l4_msgtag_t, msi_info, (l4_umword_t irqnum, l4_uint64_t source,
00365 l4_icu_msi_info_t *msi_info));
00366
00370 l4_msgtag_t control(unsigned irqnum, unsigned op, l4_umword_t *label,
00371 l4_timeout_t to, l4_utcb_t *utcb = l4_utcb()) noexcept
00372 { return l4_icu_control_u(cap(), irqnum, op, label, to, utcb); }

```

```

00373
00393 l4_msgtag_t mask(unsigned irqnum,
00394 l4_umword_t *label = 0,
00395 l4_timeout_t to = L4_IPC_NEVER,
00396 l4_utcb_t *utcb = l4_utcb()) noexcept
00397 { return l4_icu_mask_u(cap(), irqnum, label, to, utcb); }
00398
00399 L4_RPC_NF_OP(
00400 L4_ICU_OP_MASK,
00401 l4_msgtag_t, mask, (l4_umword_t irqnum),
00402 L4::Ipc::Send_only
00403);
00404
00405
00406 L4_RPC_NF_OP(
00407 L4_ICU_OP_UNMASK,
00408 l4_msgtag_t, unmask, (l4_umword_t irqnum),
00409 L4::Ipc::Send_only
00410);
00411
00421 l4_msgtag_t set_mode(unsigned irqnum, l4_umword_t mode,
00422 l4_utcb_t *utcb = l4_utcb()) noexcept
00423 { return l4_icu_set_mode_u(cap(), irqnum, mode, utcb); }
00424
00425 L4_RPC_NF_OP(
00426 L4_ICU_OP_SET_MODE,
00427 l4_msgtag_t, set_mode, (l4_umword_t irqnum, l4_umword_t mode)
00428);
00429
00430 typedef L4::Typeid::Rpcs_sys<
00431 bind_t, unbind_t, info_t, msi_info_t, unmask_t, mask_t, set_mode_t
00432 > Rpcs;
00433 };
00434
00435 }

```

## 17.535 l4/sys/kdebug.h File Reference

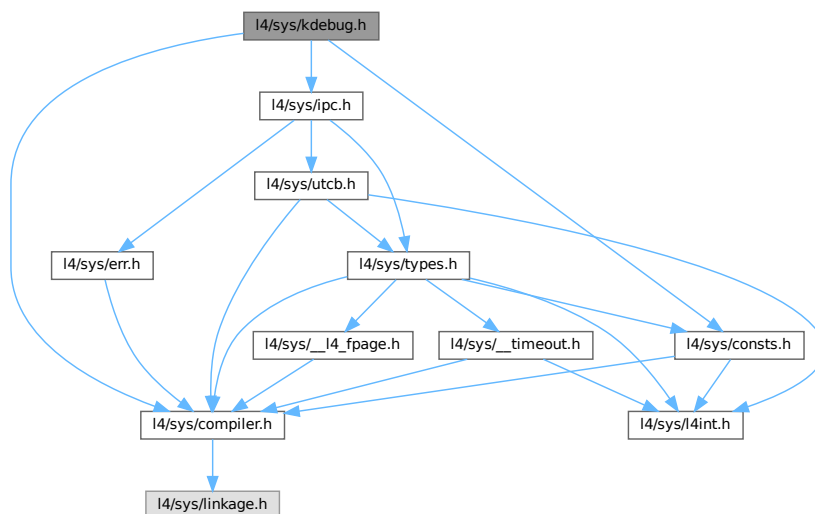
Functionality for invoking the kernel debugger.

```

#include <l4/sys/compiler.h>
#include <l4/sys/consts.h>
#include <l4/sys/ipc.h>

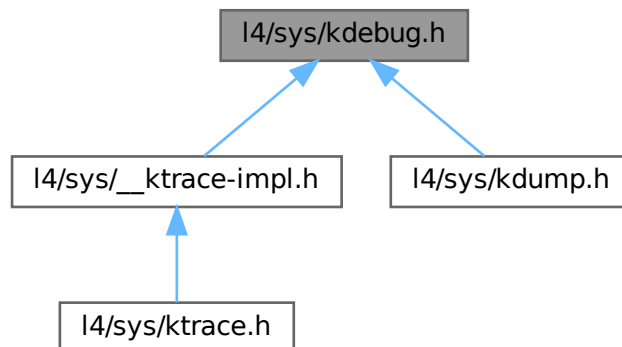
```

Include dependency graph for kdebug.h:





This graph shows which files directly or indirectly include this file:



## Enumerations

- enum [l4\\_kdebug\\_group\\_t](#)  
*Opcode groups for operations that can be invoked on the base debugger capability.*
- enum [l4\\_kdebug\\_ops\\_t](#)  
*Op-codes for operations that can be invoked on the base debugger capability.*

## Functions

- void [enter\\_kdebug](#) (char const \*text) [L4\\_NOTHROW](#)  
*Enter the kernel debugger.*
- [l4\\_msgtag\\_t \\_\\_kdebug\\_op](#) (unsigned op) [L4\\_NOTHROW](#)  
*Invoke a nullary operation on the base debugger capability.*
- [l4\\_msgtag\\_t \\_\\_kdebug\\_text](#) (unsigned op, char const \*text, unsigned len) [L4\\_NOTHROW](#)  
*Invoke a text output operation on the base debugger capability.*
- [l4\\_msgtag\\_t \\_\\_kdebug\\_3\\_text](#) (unsigned op, char const \*text, unsigned len, [l4\\_umword\\_t](#) v1, [l4\\_umword\\_t](#) v2, [l4\\_umword\\_t](#) v3) [L4\\_NOTHROW](#)  
*Invoke a text output operation with 3 additional machine word arguments on the base debugger capability.*
- [l4\\_msgtag\\_t \\_\\_kdebug\\_op\\_1](#) (unsigned op, [l4\\_mword\\_t](#) val) [L4\\_NOTHROW](#)  
*Invoke an unary operation on the base debugger capability.*
- void [outnstring](#) (char const \*text, unsigned len)  
*Output a fixed-length string via the kernel debugger.*
- void [outstring](#) (char const \*text)  
*Output a string via the kernel debugger.*
- void [outchar](#) (char c)  
*Output a single character via the kernel debugger.*
- void [outumword](#) ([l4\\_umword\\_t](#) number)  
*Output a hexadecimal unsigned machine word via the kernel debugger.*
- void [outhex64](#) ([l4\\_uint64\\_t](#) number)  
*Output a 64-bit unsigned hexadecimal number via the kernel debugger.*
- void [outhex32](#) ([l4\\_uint32\\_t](#) number)

- Output a 32-bit unsigned hexadecimal number via the kernel debugger.*
- void `outhex20` (`l4_uint32_t` number)
- Output a 20-bit unsigned hexadecimal number via the kernel debugger.*
- void `outhex16` (`l4_uint16_t` number)
- Output a 16-bit unsigned hexadecimal number via the kernel debugger.*
- void `outhex12` (`l4_uint16_t` number)
- Output a 12-bit unsigned hexadecimal number via the kernel debugger.*
- void `outhex8` (`l4_uint8_t` number)
- Output an 8-bit unsigned hexadecimal number via the kernel debugger.*
- void `outdec` (`l4_mword_t` number)
- Output a decimal unsigned machine word via the kernel debugger.*

### 17.535.1 Detailed Description

Functionality for invoking the kernel debugger.

Definition in file [kdebug.h](#).

### 17.535.2 Enumeration Type Documentation

#### 17.535.2.1 `l4_kdebug_ops_t`

enum `l4_kdebug_ops_t`

Op-codes for operations that can be invoked on the base debugger capability.

See also [\\_\\_ktrace-impl.h](#) for additional op-codes.

Definition at line 44 of file [kdebug.h](#).

### 17.535.3 Function Documentation

#### 17.535.3.1 `__kdebug_3_text()`

```
l4_msgtag_t __kdebug_3_text (
 unsigned op,
 char const * text,
 unsigned len,
 l4_umword_t v1,
 l4_umword_t v2,
 l4_umword_t v3) [inline]
```

Invoke a text output operation with 3 additional machine word arguments on the base debugger capability.

#### Parameters

---

|             |                                                                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>op</i>   | Text output operation code from <a href="#">l4_kdebug_ops_t</a> or a value above 0x200 used by the kernel trace buffer implementation ( <a href="#">__ktrace-impl.h</a> ).        |
| <i>text</i> | Output string.                                                                                                                                                                    |
| <i>len</i>  | Length of the output string. The maximum length is limited to <a href="#">L4_UTCB_GENERIC_DATA_SIZE</a> - 5 machine words. Output strings longer than this limit will be cropped. |
| <i>v1</i>   | First machine word argument.                                                                                                                                                      |
| <i>v2</i>   | Second machine word argument.                                                                                                                                                     |
| <i>v3</i>   | Third machine word argument.                                                                                                                                                      |

### Return values

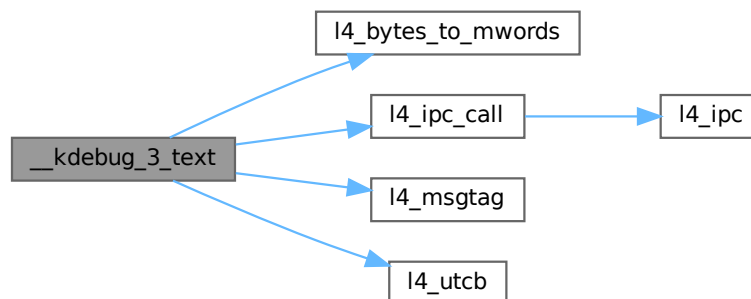
|                |                                                            |
|----------------|------------------------------------------------------------|
| <i>Message</i> | tag returned from the IPC on the base debugger capability. |
|----------------|------------------------------------------------------------|

Definition at line 139 of file [kdebug.h](#).

References [L4\\_BASE\\_DEBUGGER\\_CAP](#), [l4\\_bytes\\_to\\_mwords\(\)](#), [l4\\_ipc\\_call\(\)](#), [L4\\_IPC\\_NEVER](#), [l4\\_msgtag\(\)](#), [L4\\_NOTHROW](#), [L4\\_PROTO\\_DEBUGGER](#), [l4\\_utcb\(\)](#), and [l4\\_msg\\_regs\\_t::mr](#).

Referenced by [fiasco\\_tbuf\\_log\\_3val\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 17.535.3.2 `__kdebug_op()`

```
l4_msgtag_t __kdebug_op (
 unsigned op) [inline]
```

Invoke a nullary operation on the base debugger capability.

#### Parameters

|           |                                                                                                                                                                        |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>op</i> | Nullary operation code from <a href="#">l4_kdebug_ops_t</a> or a value above 0x200 used by the kernel trace buffer implementation ( <a href="#">__ktrace-impl.h</a> ). |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

#### Return values

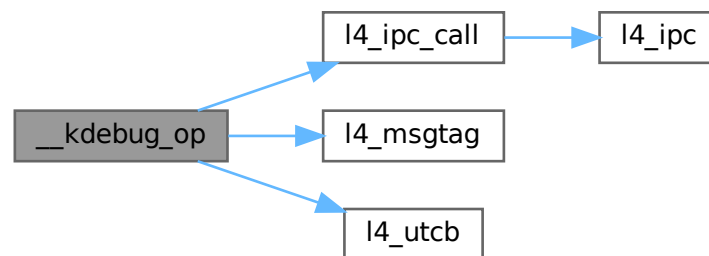
|                |                                                            |
|----------------|------------------------------------------------------------|
| <i>Message</i> | tag returned from the IPC on the base debugger capability. |
|----------------|------------------------------------------------------------|

Definition at line 68 of file [kdebug.h](#).

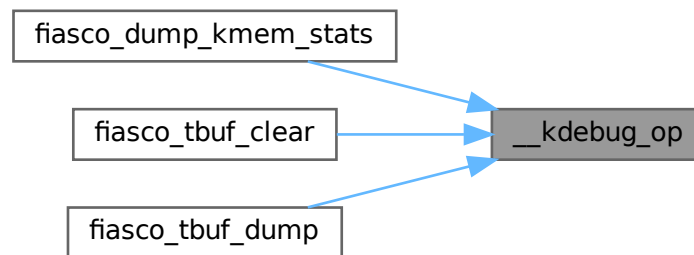
References [L4\\_BASE\\_DEBUGGER\\_CAP](#), [l4\\_ipc\\_call\(\)](#), [L4\\_IPC\\_NEVER](#), [l4\\_msgtag\(\)](#), [L4\\_NOTHROW](#), [L4\\_PROTO\\_DEBUGGER](#), [l4\\_utcb\(\)](#), and [l4\\_msg\\_regs\\_t::mr](#).

Referenced by [fiasco\\_dump\\_kmem\\_stats\(\)](#), [fiasco\\_tbuf\\_clear\(\)](#), and [fiasco\\_tbuf\\_dump\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 17.535.3.3 \_\_kdebug\_op\_1()

```

l4_msgtag_t __kdebug_op_1 (
 unsigned op,
 l4_mword_t val) [inline]

```

Invoke an unary operation on the base debugger capability.

#### Parameters

|            |                                                                                                                                                                      |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>op</i>  | Unary operation code from <a href="#">l4_kdebug_ops_t</a> or a value above 0x200 used by the kernel trace buffer implementation ( <a href="#">__ktrace-impl.h</a> ). |
| <i>val</i> | Machine word argument to the unary operation.                                                                                                                        |

#### Return values

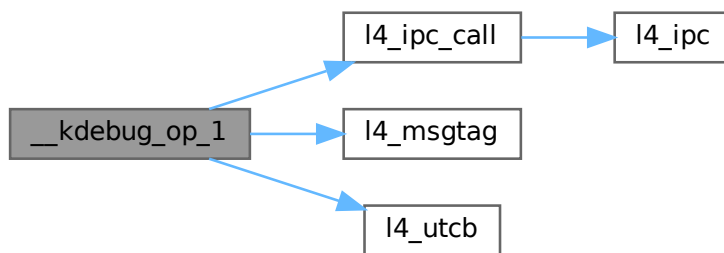
|                |                                                            |
|----------------|------------------------------------------------------------|
| <i>Message</i> | tag returned from the IPC on the base debugger capability. |
|----------------|------------------------------------------------------------|

Definition at line 176 of file [kdebug.h](#).

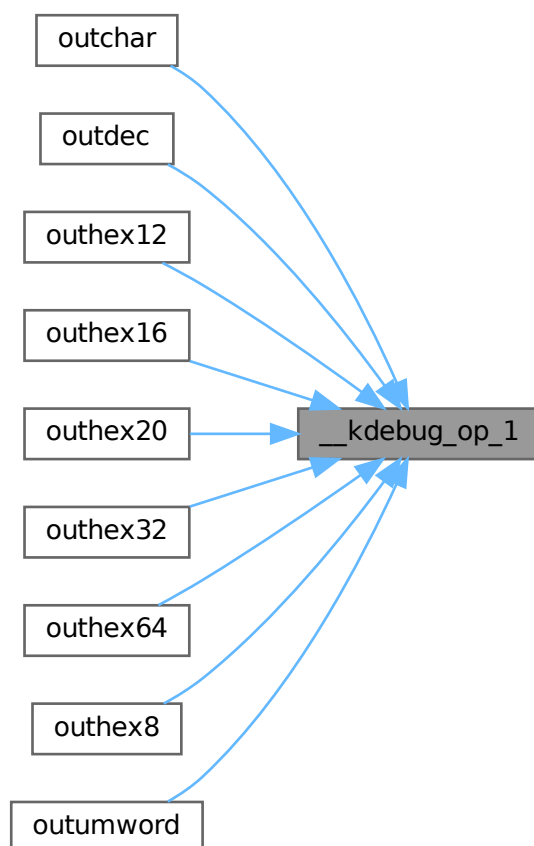
References [L4\\_BASE\\_DEBUGGER\\_CAP](#), [l4\\_ipc\\_call\(\)](#), [L4\\_IPC\\_NEVER](#), [l4\\_msgtag\(\)](#), [L4\\_NOTHROW](#), [L4\\_PROTO\\_DEBUGGER](#), [l4\\_utcb\(\)](#), and [l4\\_msg\\_regs\\_t::mr](#).

Referenced by [outchar\(\)](#), [outdec\(\)](#), [outhex12\(\)](#), [outhex16\(\)](#), [outhex20\(\)](#), [outhex32\(\)](#), [outhex64\(\)](#), [outhex8\(\)](#), and [outumword\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



17.535.3.4 `__kdebug_text()`

```
l4_msgtag_t __kdebug_text (
 unsigned op,
 char const * text,
 unsigned len) [inline]
```

Invoke a text output operation on the base debugger capability.

**Parameters**

|             |                                                                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>op</i>   | Text output operation code from <a href="#">l4_kdebug_ops_t</a> or a value above 0x200 used by the kernel trace buffer implementation ( <a href="#">__ktrace-impl.h</a> ).        |
| <i>text</i> | Output string.                                                                                                                                                                    |
| <i>len</i>  | Length of the output string. The maximum length is limited to <a href="#">L4_UTCB_GENERIC_DATA_SIZE</a> - 2 machine words. Output strings longer than this limit will be cropped. |

**Return values**

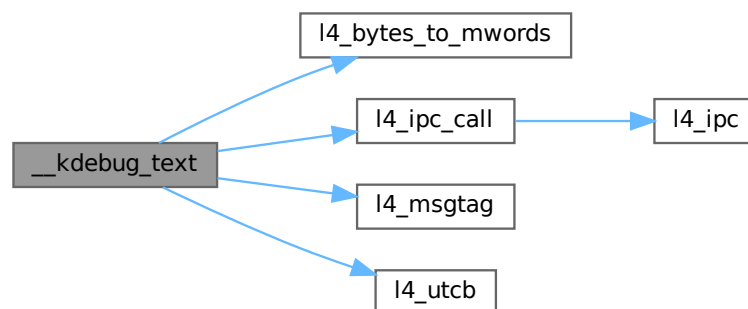
|                |                                                            |
|----------------|------------------------------------------------------------|
| <i>Message</i> | tag returned from the IPC on the base debugger capability. |
|----------------|------------------------------------------------------------|

Definition at line 98 of file [kdebug.h](#).

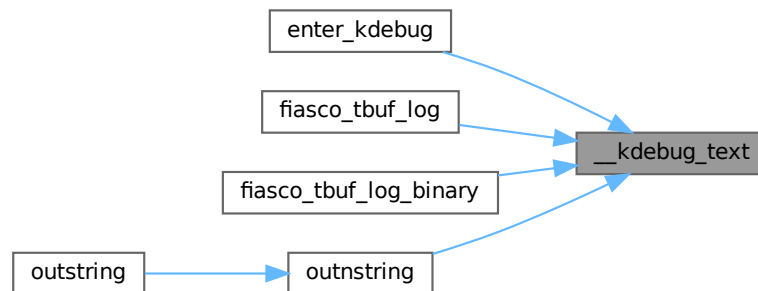
References [L4\\_BASE\\_DEBUGGER\\_CAP](#), [l4\\_bytes\\_to\\_mwords\(\)](#), [l4\\_ipc\\_call\(\)](#), [L4\\_IPC\\_NEVER](#), [l4\\_msgtag\(\)](#), [L4\\_NOTHROW](#), [L4\\_PROTO\\_DEBUGGER](#), [l4\\_utcb\(\)](#), and [l4\\_msg\\_regs\\_t::mr](#).

Referenced by [enter\\_kdebug\(\)](#), [fiasco\\_tbuf\\_log\(\)](#), [fiasco\\_tbuf\\_log\\_binary\(\)](#), and [outnstring\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 17.535.3.5 enter\_kdebug()

```
void enter_kdebug (
 char const * text) [inline]
```

Enter the kernel debugger.

#### Parameters

|             |                                                                 |
|-------------|-----------------------------------------------------------------|
| <i>text</i> | Optional message displayed by the kernel debugger when entered. |
|-------------|-----------------------------------------------------------------|

Enter the kernel debugger, if configured. An optional message can be passed to the kernel debugger which is printed upon the entering of the debugger.

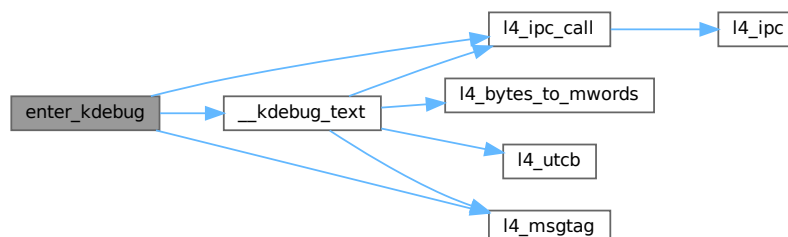
#### Examples

[examples/sys/singlestep/main.c](#).

Definition at line 204 of file [kdebug.h](#).

References [\\_\\_kdebug\\_text\(\)](#), [L4\\_BASE\\_DEBUGGER\\_CAP](#), [L4\\_INLINE](#), [l4\\_ipc\\_call\(\)](#), [L4\\_IPC\\_NEVER](#), [l4\\_msgtag\(\)](#), [L4\\_NOTHROW](#), and [L4\\_PROTO\\_DEBUGGER](#).

Here is the call graph for this function:





### 17.535.3.6 outchar()

```
void outchar (
 char c) [inline]
```

Output a single character via the kernel debugger.

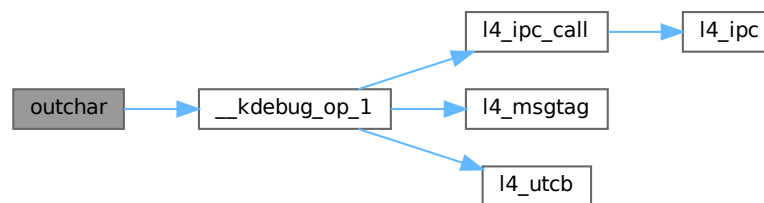
#### Parameters

|          |                   |
|----------|-------------------|
| <i>c</i> | Output character. |
|----------|-------------------|

Definition at line 245 of file [kdebug.h](#).

References [\\_\\_kdebug\\_op\\_1\(\)](#), and [L4\\_INLINE](#).

Here is the call graph for this function:



### 17.535.3.7 outdec()

```
void outdec (
 l4_mword_t number) [inline]
```

Output a decimal unsigned machine word via the kernel debugger.

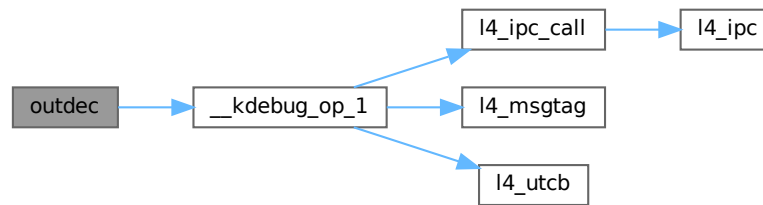
#### Parameters

|               |                      |
|---------------|----------------------|
| <i>number</i> | Output machine word. |
|---------------|----------------------|

Definition at line 334 of file [kdebug.h](#).

References [\\_\\_kdebug\\_op\\_1\(\)](#), and [L4\\_INLINE](#).

Here is the call graph for this function:



### 17.535.3.8 outhex12()

```
void outhex12 (
 14_uint16_t number) [inline]
```

Output a 12-bit unsigned hexadecimal number via the kernel debugger.

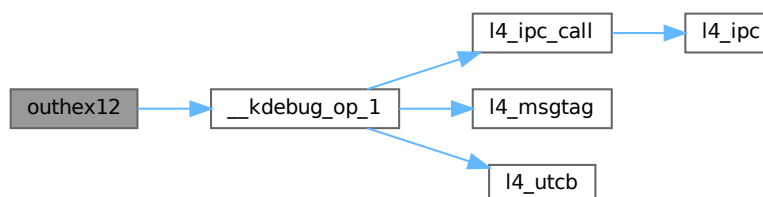
#### Parameters

|               |                                                      |
|---------------|------------------------------------------------------|
| <i>number</i> | Output 12-bit number. Only the 12 LSB bits are used. |
|---------------|------------------------------------------------------|

Definition at line 314 of file [kdebug.h](#).

References [\\_\\_kdebug\\_op\\_1\(\)](#), and [L4\\_INLINE](#).

Here is the call graph for this function:



### 17.535.3.9 outhex16()

```
void outhex16 (
 14_uint16_t number) [inline]
```

Output a 16-bit unsigned hexadecimal number via the kernel debugger.

#### Parameters

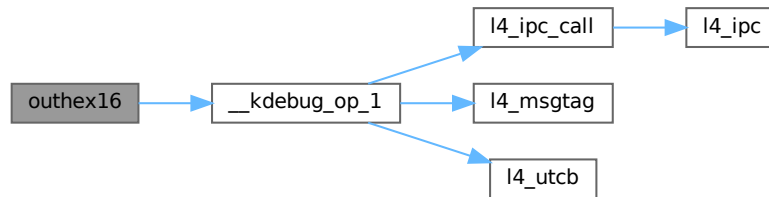
---

|               |                       |
|---------------|-----------------------|
| <i>number</i> | Output 16-bit number. |
|---------------|-----------------------|

Definition at line 304 of file [kdebug.h](#).

References [\\_\\_kdebug\\_op\\_1\(\)](#), and [L4\\_INLINE](#).

Here is the call graph for this function:



### 17.535.3.10 outhex20()

```
void outhex20 (
 l4_uint32_t number) [inline]
```

Output a 20-bit unsigned hexadecimal number via the kernel debugger.

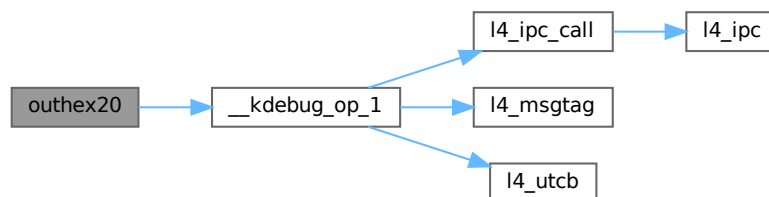
#### Parameters

|               |                                                      |
|---------------|------------------------------------------------------|
| <i>number</i> | Output 20-bit number. Only the 20 LSB bits are used. |
|---------------|------------------------------------------------------|

Definition at line 294 of file [kdebug.h](#).

References [\\_\\_kdebug\\_op\\_1\(\)](#), and [L4\\_INLINE](#).

Here is the call graph for this function:



**17.535.3.11 outhex32()**

```
void outhex32 (
 14_uint32_t number) [inline]
```

Output a 32-bit unsigned hexadecimal number via the kernel debugger.

**Parameters**

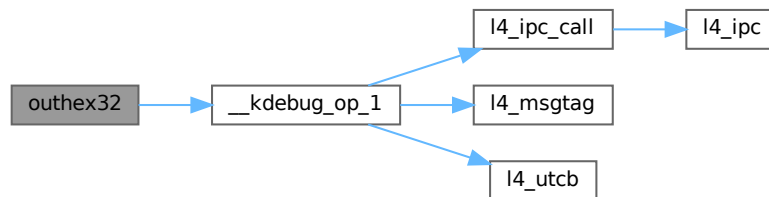
---

|               |                       |
|---------------|-----------------------|
| <i>number</i> | Output 32-bit number. |
|---------------|-----------------------|

Definition at line 284 of file [kdebug.h](#).

References [\\_\\_kdebug\\_op\\_1\(\)](#), and [L4\\_INLINE](#).

Here is the call graph for this function:



### 17.535.3.12 outhex64()

```
void outhex64 (
 l4_uint64_t number) [inline]
```

Output a 64-bit unsigned hexadecimal number via the kernel debugger.

#### Parameters

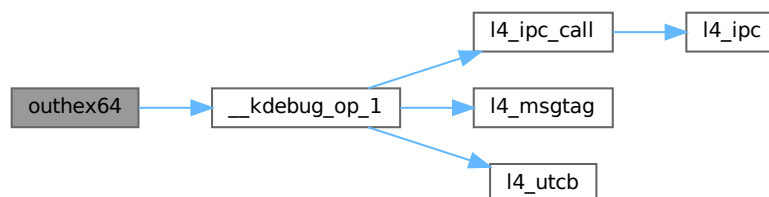
|               |                       |
|---------------|-----------------------|
| <i>number</i> | Output 64-bit number. |
|---------------|-----------------------|

The two 32-bit halves are printed non-atomically.

Definition at line 273 of file [kdebug.h](#).

References [\\_\\_kdebug\\_op\\_1\(\)](#), and [L4\\_INLINE](#).

Here is the call graph for this function:



### 17.535.3.13 outhex8()

```
void outhex8 (
 l4_uint8_t number) [inline]
```

Output an 8-bit unsigned hexadecimal number via the kernel debugger.

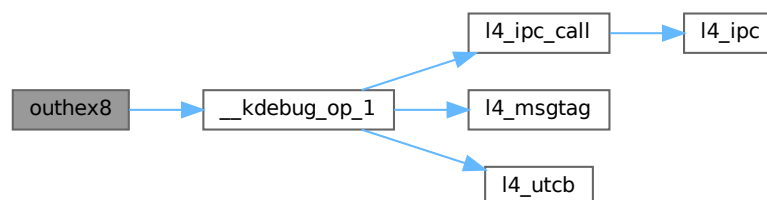
#### Parameters

|               |                      |
|---------------|----------------------|
| <i>number</i> | Output 8-bit number. |
|---------------|----------------------|

Definition at line 324 of file [kdebug.h](#).

References [\\_\\_kdebug\\_op\\_1\(\)](#), and [L4\\_INLINE](#).

Here is the call graph for this function:



### 17.535.3.14 outnstring()

```
void outnstring (
 char const * text,
 unsigned len) [inline]
```

Output a fixed-length string via the kernel debugger.

#### Parameters

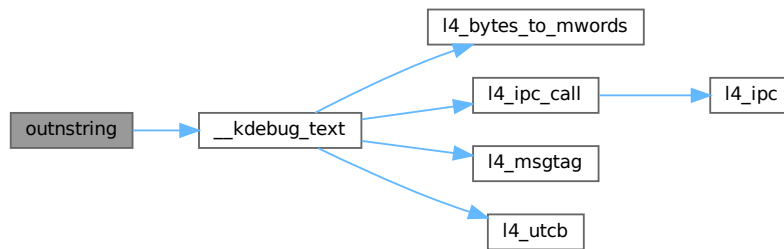
|             |                                                                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>text</i> | Beginning of the output string.                                                                                                                                                   |
| <i>len</i>  | Length of the output string. The maximum length is limited to <a href="#">L4_UTCB_GENERIC_DATA_SIZE</a> - 2 machine words. Output strings longer than this limit will be cropped. |

Definition at line 226 of file [kdebug.h](#).

References [\\_\\_kdebug\\_text\(\)](#), and [L4\\_INLINE](#).

Referenced by [outstring\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 17.535.3.15 outstring()

```
void outstring (
 char const * text) [inline]
```

Output a string via the kernel debugger.

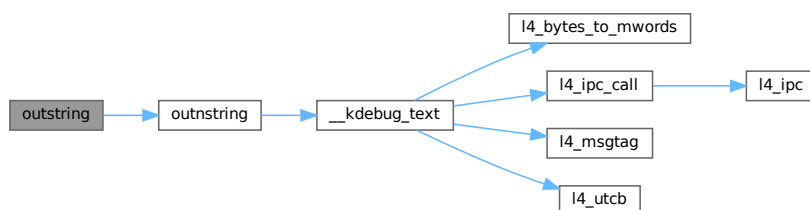
#### Parameters

|             |                                                                                                                                                                                                           |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>text</i> | Beginning of the output string. The maximum length of the output string is limited to <a href="#">L4_UTCB_GENERIC_DATA_SIZE</a> - 2 machine words. Output strings longer than this limit will be cropped. |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Definition at line 237 of file [kdebug.h](#).

References [L4\\_INLINE](#), and [outnstring\(\)](#).

Here is the call graph for this function:



### 17.535.3.16 outumword()

```
void outumword (
 l4_umword_t number) [inline]
```

Output a hexadecimal unsigned machine word via the kernel debugger.

#### Parameters

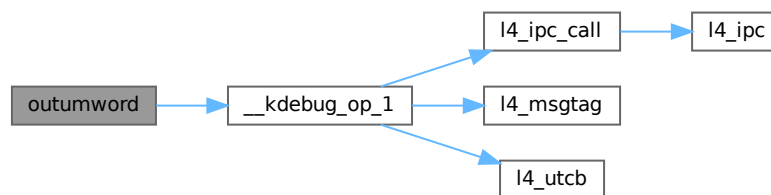
|               |                      |
|---------------|----------------------|
| <i>number</i> | Output machine word. |
|---------------|----------------------|

If the machine word is 64 bits long, it is printed non-atomically as two 32-bit numbers.

Definition at line 258 of file [kdebug.h](#).

References [\\_\\_kdebug\\_op\\_1\(\)](#), and [L4\\_INLINE](#).

Here is the call graph for this function:



## 17.536 kdebug.h

[Go to the documentation of this file.](#)

```
00001 /*
00002 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00003 * economic rights: Technische Universität Dresden (Germany)
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #pragma once
00009
00010 #ifndef __KDEBUG_H__
00011 #define __KDEBUG_H__
00012
00013 #include <l4/sys/compiler.h>
00014 #include <l4/sys/consts.h>
00015 #include <l4/sys/ipc.h>
00016
00017 L4_INLINE void
00018 enter_kdebug(char const *text) L4_NOTHROW;
00019
00020 enum l4_kdebug_group_t
00021 {
00022 L4_KDEBUG_GROUP_JDB = 0x000,
00023 L4_KDEBUG_GROUP_KOBJ = 0x100, // see __kernel_object_impl.h
00024 L4_KDEBUG_GROUP_TRACE = 0x200, // see __ktrace-impl.h
00025 L4_KDEBUG_GROUP_COV = 0x400,
00026 L4_KDEBUG_GROUP_DUMP = 0x500, // see kdump.h
00027 }
```



```

00038 };
00039
00044 enum l4_kdebug_ops_t
00045 {
00046 L4_KDEBUG_ENTER = L4_KDEBUG_GROUP_JDB + 0,
00047 L4_KDEBUG_OUTCHAR = L4_KDEBUG_GROUP_JDB + 1,
00048 L4_KDEBUG_OUTNSTRING = L4_KDEBUG_GROUP_JDB + 2,
00049 L4_KDEBUG_OUTHEX32 = L4_KDEBUG_GROUP_JDB + 3,
00050 L4_KDEBUG_OUTHEX20 = L4_KDEBUG_GROUP_JDB + 4,
00051 L4_KDEBUG_OUTHEX16 = L4_KDEBUG_GROUP_JDB + 5,
00052 L4_KDEBUG_OUTHEX12 = L4_KDEBUG_GROUP_JDB + 6,
00053 L4_KDEBUG_OUTHEX8 = L4_KDEBUG_GROUP_JDB + 7,
00054 L4_KDEBUG_OUTDEC = L4_KDEBUG_GROUP_JDB + 8,
00055 };
00056
00057
00067 L4_INLINE l4_msgtag_t
00068 __kdebug_op(unsigned op) L4_NOTHROW
00069 {
00070 l4_msgtag_t res;
00071 l4_utcb_t *u = l4_utcb();
00072 l4_msg_regs_t *mr = l4_utcb_mr_u(u);
00073 l4_umword_t mr0 = mr->mr[0];
00074
00075 mr->mr[0] = op;
00076 res = l4_ipc_call(L4_BASE_DEBUGGER_CAP, u,
00077 l4_msgtag(L4_PROTO_DEBUGGER, 1, 0, 0),
00078 L4_IPC_NEVER);
00079 mr->mr[0] = mr0;
00080 return res;
00081 }
00082
00097 L4_INLINE l4_msgtag_t
00098 __kdebug_text(unsigned op, char const *text, unsigned len) L4_NOTHROW
00099 {
00100 l4_msg_regs_t store;
00101 l4_msgtag_t res;
00102 l4_utcb_t *u = l4_utcb();
00103 l4_msg_regs_t *mr = l4_utcb_mr_u(u);
00104
00105 if (len > (sizeof(store) - (2 * sizeof(l4_umword_t))))
00106 len = sizeof(store) - (2 * sizeof(l4_umword_t));
00107
00108 __builtin_memcpy(&store, mr, sizeof(store));
00109 mr->mr[0] = op;
00110 mr->mr[1] = len;
00111 __builtin_memcpy(&mr->mr[2], text, len);
00112 res = l4_ipc_call(L4_BASE_DEBUGGER_CAP, u,
00113 l4_msgtag(L4_PROTO_DEBUGGER,
00114 l4_bytes_to_mwords(len) + 2, 0, 0),
00115 L4_IPC_NEVER);
00116 __builtin_memcpy(mr, &store, sizeof(*mr));
00117 return res;
00118 }
00119
00138 L4_INLINE l4_msgtag_t
00139 __kdebug_3_text(unsigned op, char const *text, unsigned len,
00140 l4_umword_t v1, l4_umword_t v2, l4_umword_t v3) L4_NOTHROW
00141 {
00142 l4_msg_regs_t store;
00143 l4_msgtag_t res;
00144 l4_utcb_t *u = l4_utcb();
00145 l4_msg_regs_t *mr = l4_utcb_mr_u(u);
00146
00147 if (len > (sizeof(store) - (5 * sizeof(l4_umword_t))))
00148 len = sizeof(store) - (5 * sizeof(l4_umword_t));
00149
00150 __builtin_memcpy(&store, mr, sizeof(store));
00151 mr->mr[0] = op;
00152 mr->mr[1] = v1;
00153 mr->mr[2] = v2;
00154 mr->mr[3] = v3;
00155 mr->mr[4] = len;
00156 __builtin_memcpy(&mr->mr[5], text, len);
00157 res = l4_ipc_call(L4_BASE_DEBUGGER_CAP, u,
00158 l4_msgtag(L4_PROTO_DEBUGGER,
00159 l4_bytes_to_mwords(len) + 5, 0, 0),
00160 L4_IPC_NEVER);
00161 __builtin_memcpy(mr, &store, sizeof(*mr));
00162 return res;
00163 }
00164
00175 L4_INLINE l4_msgtag_t
00176 __kdebug_op_l(unsigned op, l4_mword_t val) L4_NOTHROW
00177 {
00178 l4_umword_t m[2];
00179 l4_msgtag_t res;

```

```

00180 l4_utcb_t *u = l4_utcb();
00181 l4_msg_regs_t *mr = l4_utcb_mr_u(u);
00182
00183 m[0] = mr->mr[0];
00184 m[1] = mr->mr[1];
00185 mr->mr[0] = op;
00186 mr->mr[1] = val;
00187 res = l4_ipc_call(L4_BASE_DEBUGGER_CAP, u,
00188 l4_msgtag(L4_PROTO_DEBUGGER, 2, 0, 0),
00189 L4_IPC_NEVER);
00190 mr->mr[0] = m[0];
00191 mr->mr[1] = m[1];
00192 return res;
00193 }
00194
00204 L4_INLINE void enter_kdebug(char const *text) L4_NOTHROW
00205 {
00206 /* special case, enter without any text and use of the UTCB */
00207 if (!text)
00208 {
00209 l4_ipc_call(L4_BASE_DEBUGGER_CAP, 0,
00210 l4_msgtag(L4_PROTO_DEBUGGER, 0, 0, 0),
00211 L4_IPC_NEVER);
00212 return;
00213 }
00214 __kdebug_text(L4_KDEBUG_ENTER, text, __builtin_strlen(text));
00215 }
00216
00226 L4_INLINE void outnstring(char const *text, unsigned len)
00227 { __kdebug_text(L4_KDEBUG_OUTNSTRING, text, len); }
00228
00237 L4_INLINE void outstring(char const *text)
00238 { outnstring(text, __builtin_strlen(text)); }
00239
00245 L4_INLINE void outchar(char c)
00246 {
00247 __kdebug_op_1(L4_KDEBUG_OUTCHAR, c);
00248 }
00249
00258 L4_INLINE void outumword(l4_umword_t number)
00259 {
00260 if (sizeof(l4_umword_t) == sizeof(l4_uint64_t))
00261 __kdebug_op_1(L4_KDEBUG_OUTHEX32, (l4_uint64_t)number >> 32);
00262 __kdebug_op_1(L4_KDEBUG_OUTHEX32, number);
00263 }
00264
00273 L4_INLINE void outhex64(l4_uint64_t number)
00274 {
00275 __kdebug_op_1(L4_KDEBUG_OUTHEX32, number >> 32);
00276 __kdebug_op_1(L4_KDEBUG_OUTHEX32, number);
00277 }
00278
00284 L4_INLINE void outhex32(l4_uint32_t number)
00285 {
00286 __kdebug_op_1(L4_KDEBUG_OUTHEX32, number);
00287 }
00288
00294 L4_INLINE void outhex20(l4_uint32_t number)
00295 {
00296 __kdebug_op_1(L4_KDEBUG_OUTHEX20, number);
00297 }
00298
00304 L4_INLINE void outhex16(l4_uint16_t number)
00305 {
00306 __kdebug_op_1(L4_KDEBUG_OUTHEX16, number);
00307 }
00308
00314 L4_INLINE void outhex12(l4_uint16_t number)
00315 {
00316 __kdebug_op_1(L4_KDEBUG_OUTHEX12, number);
00317 }
00318
00324 L4_INLINE void outhex8(l4_uint8_t number)
00325 {
00326 __kdebug_op_1(L4_KDEBUG_OUTHEX8, number);
00327 }
00328
00334 L4_INLINE void outdec(l4_mword_t number)
00335 {
00336 __kdebug_op_1(L4_KDEBUG_OUTDEC, number);
00337 }
00338
00339 #endif // __KDEBUG_H__

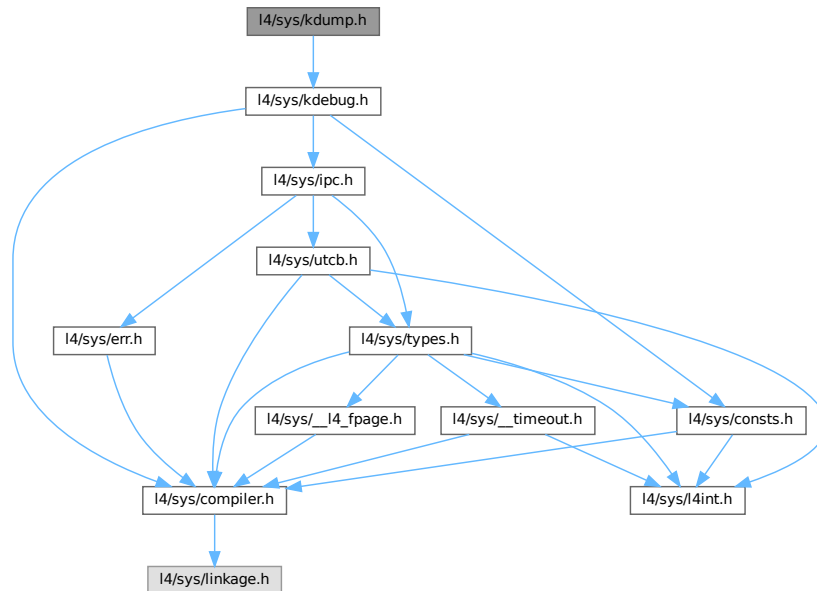
```

## 17.537 l4/sys/kdump.h File Reference

Functionality for dumping kernel information.

```
#include <l4/sys/kdebug.h>
```

Include dependency graph for kdump.h:



### Functions

- long [fiasco\\_dump\\_kmem\\_stats](#) (void)  
*Dump kernel memory statistics on console.*

### 17.537.1 Detailed Description

Functionality for dumping kernel information.

Definition in file [kdump.h](#).

### 17.537.2 Function Documentation

#### 17.537.2.1 fiasco\_dump\_kmem\_stats()

```
long fiasco_dump_kmem_stats (
 void) [inline]
```

Dump kernel memory statistics on console.

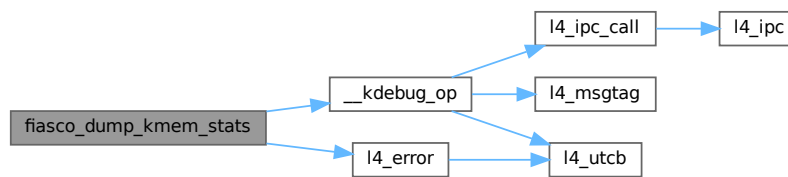
#### Return values

|            |                            |
|------------|----------------------------|
| 0          | Success.                   |
| -L4_ENOSYS | Not implemented by kernel. |

Definition at line 38 of file [kdump.h](#).

References [\\_\\_kdebug\\_op\(\)](#), and [l4\\_error\(\)](#).

Here is the call graph for this function:



## 17.538 kdump.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * Copyright (C) 2024 Kernkonzept GmbH.
00003 * Author(s): Jan Klötzke <jan.kloetzke@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #pragma once
00009
00010
00014
00025
00026 #include <l4/sys/kdebug.h>
00027
00034 L4_INLINE long
00035 fiasco_dump_kmem_stats(void);
00036
00037 L4_INLINE long
00038 fiasco_dump_kmem_stats(void)
00039 {
00040 enum { DUMP_KMEM_STATS = L4_KDEBUG_GROUP_DUMP + 0x00 };
00041 return l4_error(__kdebug_op(DUMP_KMEM_STATS));
00042 }

```

## 17.539 l4/sys/kernel\_object.h File Reference

Kernel object system calls.

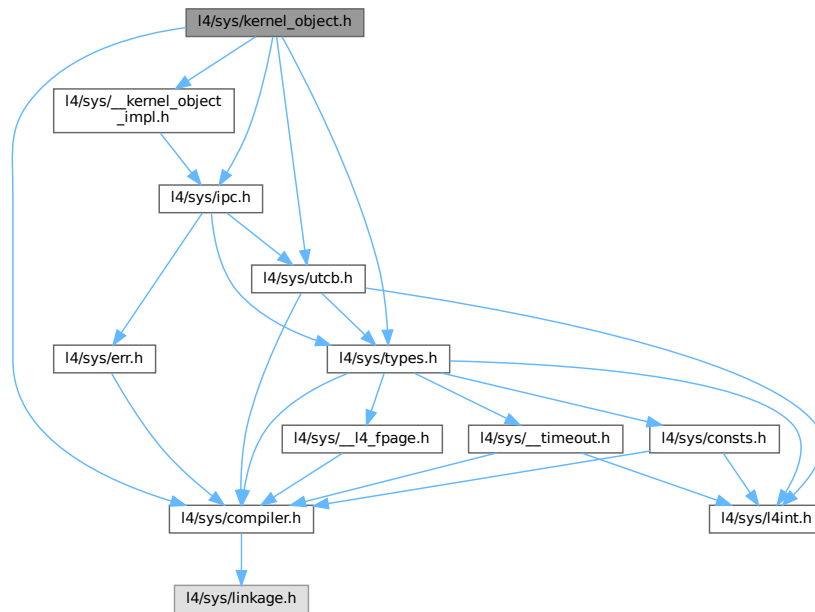
```

#include <l4/sys/types.h>
#include <l4/sys/compiler.h>
#include <l4/sys/utcb.h>
#include <l4/sys/__kernel_object_impl.h>

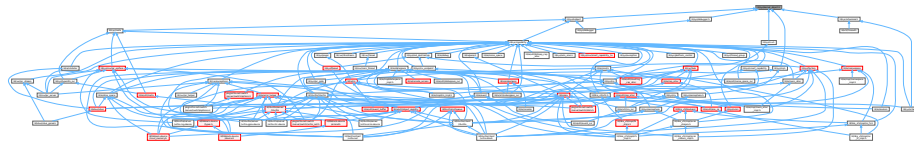
```

```
#include <l4/sys/ipc.h>
```

Include dependency graph for kernel\_object.h:



This graph shows which files directly or indirectly include this file:



## 17.539.1 Detailed Description

Kernel object system calls.

Definition in file [kernel\\_object.h](#).

## 17.540 kernel\_object.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00008 * Björn Döbel <doebel@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #ifndef __L4SYS_KERNEL_OBJECT_H__
00014 #define __L4SYS_KERNEL_OBJECT_H__

```

```

00015
00016 #include <l4/sys/types.h>
00017 #include <l4/sys/compiler.h>
00018 #include <l4/sys/utcb.h>
00019
00027
00038 L4_INLINE l4_msgtag_t
00039 l4_invoke_debugger(l4_cap_idx_t obj, l4_msgtag_t tag, l4_utcb_t *utcb) L4_NOTHROW;
00040
00041
00042 /*****
00043 * Implementation
00044 *****/
00045
00046 #include <l4/sys/__kernel_object_impl.h>
00047 #include <l4/sys/ipc.h>
00048
00049 enum L4_kobject_op {
00050 L4_KOBJECT_OP_DEC_REFCNT = 0,
00051 L4_KOBJECT_OP_REGISTER_IRQ,
00052 };
00053
00054 L4_INLINE l4_msgtag_t
00055 l4_kobject_dec_refcnt_u(l4_cap_idx_t obj, l4_mword_t diff, l4_utcb_t *u) L4_NOTHROW;
00056
00057 L4_INLINE l4_msgtag_t
00058 l4_kobject_dec_refcnt(l4_cap_idx_t obj, l4_mword_t diff) L4_NOTHROW;
00059
00060 L4_INLINE l4_msgtag_t
00061 l4_kobject_dec_refcnt_u(l4_cap_idx_t obj, l4_mword_t diff, l4_utcb_t *u) L4_NOTHROW
00062 {
00063 l4_msg_regs_t *m = l4_utcb_mr_u(u);
00064 m->mr[0] = L4_KOBJECT_OP_DEC_REFCNT;
00065 m->mr[1] = diff;
00066 return l4_ipc_call(obj, u, l4_msgtag(L4_PROTO_KOBJECT, 2, 0, 0), L4_IPC_NEVER);
00067 }
00068
00069 L4_INLINE l4_msgtag_t
00070 l4_kobject_dec_refcnt(l4_cap_idx_t obj, l4_mword_t diff) L4_NOTHROW
00071 {
00072 return l4_kobject_dec_refcnt_u(obj, diff, l4_utcb());
00073 }
00074
00075 #endif /* ! __L4SYS__KERNEL_OBJECT_H__ */

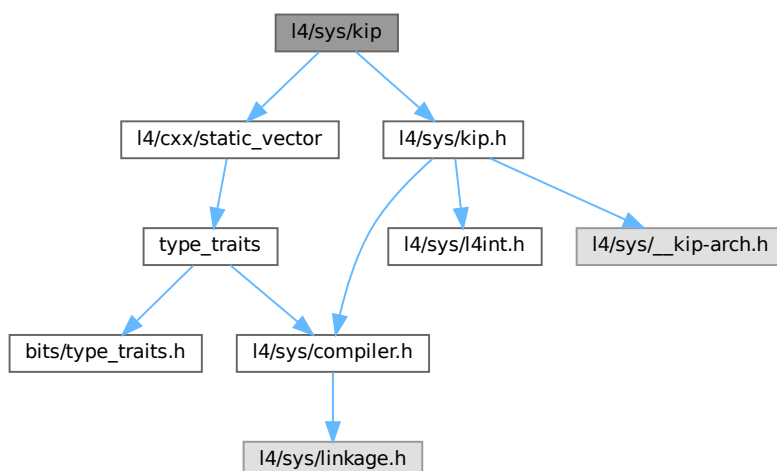
```

## 17.541 I4/sys/kip File Reference

```
#include <l4/cxx/static_vector>
```

```
#include <l4/sys/kip.h>
```

Include dependency graph for kip:



## Data Structures

- class [L4::Kip::Mem\\_desc](#)

*Memory descriptors stored in the kernel interface page.*

## Namespaces

- namespace [L4](#)

*[L4](#) low-level kernel interface.*

## 17.541.1 Detailed Description

L4::Kip class, memory descriptors.

### Author

Alexander Warg [alexander.warg@os.inf.tu-dresden.de](mailto:alexander.warg@os.inf.tu-dresden.de)

Definition in file [kip](#).

## 17.542 kip

[Go to the documentation of this file.](#)

```

00001 // vim:set ft=cpp: -*- Mode: C++ -*-
00010 /*
00011 * (c) 2008-2009 Author(s)
00012 * economic rights: Technische Universität Dresden (Germany)
00013 *
00014 * License: see LICENSE.spdx (in this directory or the directories above)
00015 */
00016 #pragma once
00017
00018 #include <l4/cxx/static_vector>
00019 #include <l4/sys/kip.h>
00020
00021 /* C++ version of memory descriptors */
00022
00031
00032 namespace L4
00033 {
00034 namespace Kip
00035 {
00042 class Mem_desc
00043 {
00044 public:
00048 enum Mem_type
00049 {
00050 Undefined = 0x0,
00051 Conventional = 0x1,
00052 Reserved = 0x2,
00053 Dedicated = 0x3,
00054 Shared = 0x4,
00055
00056 Info = 0xd,
00057 Bootloader = 0xe,
00058 Arch = 0xf
00059 };
00060
00064 enum Info_sub_type
00065 {
00066 Info_acpi_rsdp = 0
00067 };
00068
00072 enum Arch_sub_type_common
00073 {

```

```

00074 Arch_acpi_tables = 3,
00075 Arch_acpi_nvs = 4,
00076 };
00077
00078 private:
00079 unsigned long _l, _h;
00080
00081 public:
00082 static Mem_desc *first(l4_kernel_info_t *kip) noexcept
00083 {
00084 return reinterpret_cast<Mem_desc *>(reinterpret_cast<char *>(kip) + kip->mem_descs);
00085 }
00086
00087 static Mem_desc const *first(l4_kernel_info_t const *kip) noexcept
00088 {
00089 char const *addr = reinterpret_cast<char const *>(kip) + kip->mem_descs;
00090 return reinterpret_cast<Mem_desc const *>(addr);
00091 }
00092
00093 static unsigned long count(l4_kernel_info_t const *kip) noexcept
00094 {
00095 return kip->mem_descs_num;
00096 }
00097
00098 static void count(l4_kernel_info_t *kip, unsigned count) noexcept
00099 {
00100 kip->mem_descs_num = count;
00101 }
00102
00103 static inline cxx::static_vector<Mem_desc const> all(l4_kernel_info_t const *kip)
00104 {
00105 return cxx::static_vector<Mem_desc const>(Mem_desc::first(kip),
00106 Mem_desc::count(kip));
00107 }
00108
00109 static inline cxx::static_vector<Mem_desc> all(l4_kernel_info_t *kip)
00110 {
00111 return cxx::static_vector<Mem_desc>(Mem_desc::first(kip),
00112 Mem_desc::count(kip));
00113 }
00114
00115 Mem_desc(unsigned long start, unsigned long end,
00116 Mem_type t, unsigned char st = 0, bool virt = false,
00117 bool eager = false) noexcept
00118 : _l((start & ~0x3ffUL) | (t & 0x0f) | ((st < 4) & 0x0f0)
00119 | (virt ? 0x0200 : 0x0) | (eager ? 0x100 : 0x0)), _h(end | 0x3ffUL)
00119 {}
00120
00121 unsigned long start() const noexcept { return _l & ~0x3ffUL; }
00122
00123 unsigned long end() const noexcept { return _h | 0x3ffUL; }
00124
00125 unsigned long size() const noexcept { return end() + 1 - start(); }
00126
00127 Mem_type type() const noexcept
00128 {
00129 return static_cast<Mem_type>(_l & 0x0f);
00130 }
00131
00132 unsigned char sub_type() const noexcept { return (_l >> 4) & 0x0f; }
00133
00134 unsigned is_virtual() const noexcept { return _l & 0x200; }
00135
00136 unsigned eager_map() const noexcept { return _l & 0x100; }
00137
00138 void set(unsigned long start, unsigned long end,
00139 Mem_type t, unsigned char st = 0, bool virt = false,
00140 bool eager = false) noexcept
00141 {
00142 _l = (start & ~0x3ffUL) | (t & 0x0f) | ((st < 4) & 0x0f0)
00143 | (virt?0x0200:0x0) | (eager ? 0x0100 : 0x0);
00144
00145 _h = end | 0x3ffUL;
00146 }
00147 };
00148 };
00149 };

```

## 17.543 kobject

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /* \file

```



```

00003 * Kobject C++ interface.
00004 */
00005 /*
00006 * Copyright (C) 2015-2017, 2019, 2021, 2023-2024 Kernkonzept GmbH.
00007 * Author(s): Alexander Warg <alexander.warg@kernkonzept.com>
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #pragma once
00012
00013 #include "kernel_object.h"
00014 #include "types.h"
00015 #include "__typeinfo.h"
00016
00017 namespace L4 {
00018
00036 class L4_EXPORT Kobject
00037 {
00038 private:
00039 Kobject();
00040 Kobject(Kobject const &);
00041 Kobject &operator = (Kobject const &);
00042
00043 template<typename T> friend struct Kobject_typeid;
00044
00045 protected:
00046 typedef Typeid::Iface<L4_PROTO_META, Kobject> __Iface;
00047 typedef Typeid::Iface_list<__Iface> __Iface_list;
00048
00055 struct __Kobject_typeid
00056 {
00057 typedef Type_info::Demand_t<> Demand;
00058 static Type_info const _m;
00059 };
00060
00069 l4_cap_idx_t cap() const noexcept { return _c(); }
00070
00071 private:
00072
00077 l4_cap_idx_t _c() const noexcept
00078 { return reinterpret_cast<l4_cap_idx_t>(this) & L4_CAP_MASK; }
00079
00080 public:
00100 l4_msgtag_t dec_refcnt(l4_mword_t diff, l4_utcb_t *utcb = l4_utcb())
00101 { return l4_kobject_dec_refcnt_u(cap(), diff, utcb); }
00102 };
00103
00104 template<typename Derived, long PROTO = L4::PROTO_ANY,
00105 typename S_DEMAND = Type_info::Demand_t<> >
00106 struct Kobject_0t : Kobject_t<Derived, L4::Kobject, PROTO, S_DEMAND> {};
00107
00108 }
00109

```

## 17.544 l4/sys/ktrace.h File Reference

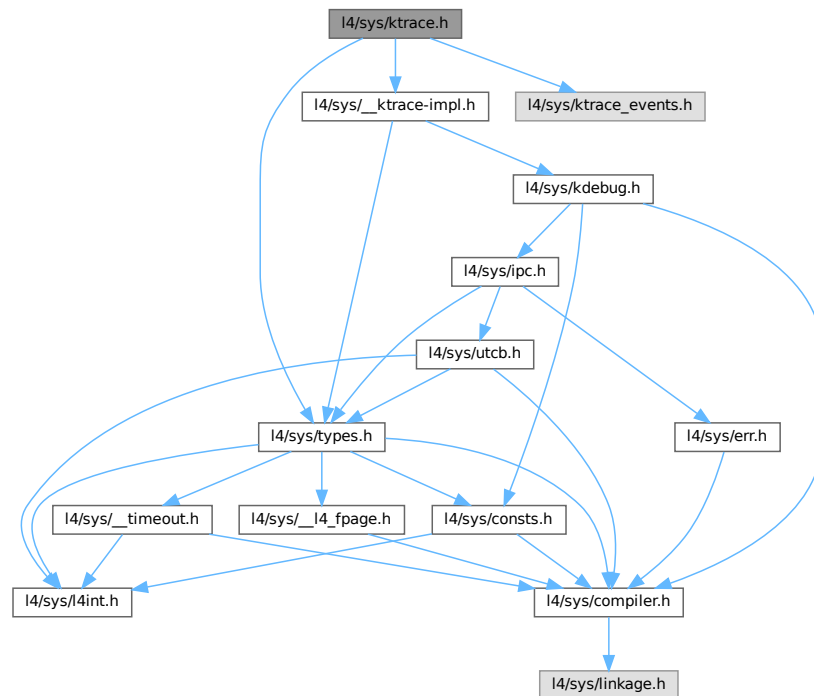
[L4](#) kernel event tracing.

```

#include <l4/sys/types.h>
#include <l4/sys/ktrace_events.h>
#include <l4/sys/__ktrace-impl.h>

```

Include dependency graph for `ktrace.h`:



## Functions

- `l4_umword_t fiasco_tbuf_log` (const char \*text)  
Create new trace-buffer entry with describing <text>.
- `l4_umword_t fiasco_tbuf_log_3val` (const char \*text, `l4_umword_t` v1, `l4_umword_t` v2, `l4_umword_t` v3)  
Create new trace-buffer entry with describing <text> and three additional values.
- `l4_umword_t fiasco_tbuf_log_binary` (const unsigned char \*data)  
Create new trace-buffer entry with binary data.
- void `fiasco_tbuf_clear` (void)  
Clear trace-buffer.
- void `fiasco_tbuf_dump` (void)  
Dump trace-buffer to kernel console.

### 17.544.1 Detailed Description

[L4](#) kernel event tracing.

Definition in file [ktrace.h](#).

## 17.545 ktrace.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Björn Döbel <doebel@os.inf.tu-dresden.de>,
00009 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00010 * 2015 Adam Lackorzynski <adam@l4re.org>
00011 * economic rights: Technische Universität Dresden (Germany)
00012 *
00013 * License: see LICENSE.spdx (in this directory or the directories above)
00014 */
00015 /*****
00016 #ifndef __L4_KTRACE_H__
00017 #define __L4_KTRACE_H__
00018
00019 #include <l4/sys/types.h>
00020 #include <l4/sys/ktrace_events.h>
00021
00033
00041 L4_INLINE l4_umword_t
00042 fiasco_tbuf_log(const char *text);
00043
00055 L4_INLINE l4_umword_t
00056 fiasco_tbuf_log_3val(const char *text, l4_umword_t v1, l4_umword_t v2, l4_umword_t v3);
00057
00065 L4_INLINE l4_umword_t
00066 fiasco_tbuf_log_binary(const unsigned char *data);
00067
00072 L4_INLINE void
00073 fiasco_tbuf_clear(void);
00074
00079 L4_INLINE void
00080 fiasco_tbuf_dump(void);
00081
00082 #include <l4/sys/__ktrace-impl.h>
00083
00084 #endif

```

## 17.546 amd64/l4/sys/l4int.h File Reference

Fixed sized integer types, AMD64 version.

### Macros

- `#define L4_MWORD_BITS 64`  
*Size of machine words in bits.*

### Typedefs

- `typedef unsigned long l4_size_t`  
*Unsigned size type.*
- `typedef signed long l4_ssize_t`  
*Signed size type.*

### 17.546.1 Detailed Description

Fixed sized integer types, AMD64 version.

Definition in file [l4int.h](#).

## 17.547 l4int.h

[Go to the documentation of this file.](#)

```

00001 /*****
00007 */
00008 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00009 * Alexander Warg <warg@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #pragma once
00015
00016 #include_next <l4/sys/l4int.h>
00017
00022
00023 #define L4_MWORD_BITS 64
00024
00025 typedef unsigned long l4_size_t;
00026 typedef signed long l4_ssize_t;
00028

```

## 17.548 arm/l4/sys/l4int.h File Reference

Fixed sized integer types, arm version.

### Macros

- `#define L4_MWORD_BITS 32`  
*Size of machine words in bits.*

### Typedefs

- `typedef unsigned int l4_size_t`  
*Unsigned size type.*
- `typedef signed int l4_ssize_t`  
*Signed size type.*

### 17.548.1 Detailed Description

Fixed sized integer types, arm version.

Definition in file [l4int.h](#).

## 17.549 l4int.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #pragma once
00014
00015 #include_next <l4/sys/l4int.h>
00016
00021
00022 #define L4_MWORD_BITS 32
00023
00024 typedef unsigned int l4_size_t;
00025 typedef signed int l4_ssize_t;
00027

```

## 17.550 arm64/l4/sys/l4int.h File Reference

Fixed sized integer types, arm version.

### Macros

- `#define L4_MWORD_BITS 64`  
*Size of machine words in bits.*

### Typedefs

- `typedef unsigned long l4_size_t`  
*Unsigned size type.*
- `typedef signed long l4_ssize_t`  
*Signed size type.*

### 17.550.1 Detailed Description

Fixed sized integer types, arm version.

Definition in file [l4int.h](#).

## 17.551 l4int.h

[Go to the documentation of this file.](#)

```

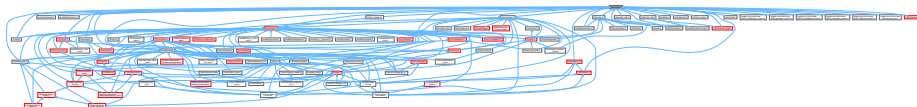
00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #pragma once
00014
00015 #include_next <l4/sys/l4int.h>
00016
00021
00022 #define L4_MWORD_BITS 64
00023
00024 typedef unsigned long l4_size_t;
00025 typedef signed long l4_ssize_t;
00027

```

## 17.552 l4/sys/l4int.h File Reference

Fixed sized integer types, generic version.

This graph shows which files directly or indirectly include this file:



## Typedefs

- typedef signed char **I4\_int8\_t**  
*Signed 8bit value.*
- typedef unsigned char **I4\_uint8\_t**  
*Unsigned 8bit value.*
- typedef signed short int **I4\_int16\_t**  
*Signed 16bit value.*
- typedef unsigned short int **I4\_uint16\_t**  
*Unsigned 16bit value.*
- typedef signed int **I4\_int32\_t**  
*Signed 32bit value.*
- typedef unsigned int **I4\_uint32\_t**  
*Unsigned 32bit value.*
- typedef signed long long **I4\_int64\_t**  
*Signed 64bit value.*
- typedef unsigned long long **I4\_uint64\_t**  
*Unsigned 64bit value.*
- typedef unsigned long **I4\_addr\_t**  
*Address type.*
- typedef signed long **I4\_mword\_t**  
*Signed machine word.*
- typedef unsigned long **I4\_umword\_t**  
*Unsigned machine word.*
- typedef [I4\\_uint64\\_t](#) **I4\_cpu\_time\_t**  
*CPU clock type.*
- typedef [I4\\_uint64\\_t](#) **I4\_kernel\_clock\_t**  
*Kernel clock type.*

### 17.552.1 Detailed Description

Fixed sized integer types, generic version.

Definition in file [I4int.h](#).

## 17.553 I4int.h

[Go to the documentation of this file.](#)

```

00001
00007
00013 /*
00014 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00015 * Alexander Warg <warg@os.inf.tu-dresden.de>
00016 * economic rights: Technische Universität Dresden (Germany)
00017 *
00018 * License: see LICENSE.spdx (in this directory or the directories above)
00019 */
00020 #ifndef __L4_SYS_L4INT_H__
00021 #define __L4_SYS_L4INT_H__
00022
00023 /* fixed sized data types */
00024 typedef signed char I4_int8_t;
00025 typedef unsigned char I4_uint8_t;
00026 typedef signed short int I4_int16_t;
00027 typedef unsigned short int I4_uint16_t;
00028 typedef signed int I4_int32_t;
```

```

00029 typedef unsigned int l4_uint32_t;
00030 typedef signed long long l4_int64_t;
00031 typedef unsigned long long l4_uint64_t;
00032
00033 /* some common data types */
00034 typedef unsigned long l4_addr_t;
00035
00036
00037 typedef signed long l4_mword_t;
00040 typedef unsigned long l4_umword_t;
00047 typedef l4_uint64_t l4_cpu_time_t;
00048
00053 typedef l4_uint64_t l4_kernel_clock_t;
00054
00055 #endif /* !__L4_SYS_L4INT_H__ */

```

## 17.554 x86/l4/sys/l4int.h File Reference

Fixed sized integer types, x86 version.

### Macros

- **#define L4\_MWORD\_BITS 32**  
*Size of machine words in bits.*

### Typedefs

- typedef unsigned int **l4\_size\_t**  
*Unsigned size type.*
- typedef signed int **l4\_ssize\_t**  
*Signed size type.*

### 17.554.1 Detailed Description

Fixed sized integer types, x86 version.

Definition in file [l4int.h](#).

## 17.555 l4int.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #pragma once
00014
00015 #include_next <l4/sys/l4int.h>
00016
00021
00022 #define L4_MWORD_BITS 32
00023
00024 typedef unsigned int l4_size_t;
00025 typedef signed int l4_ssize_t;
00027

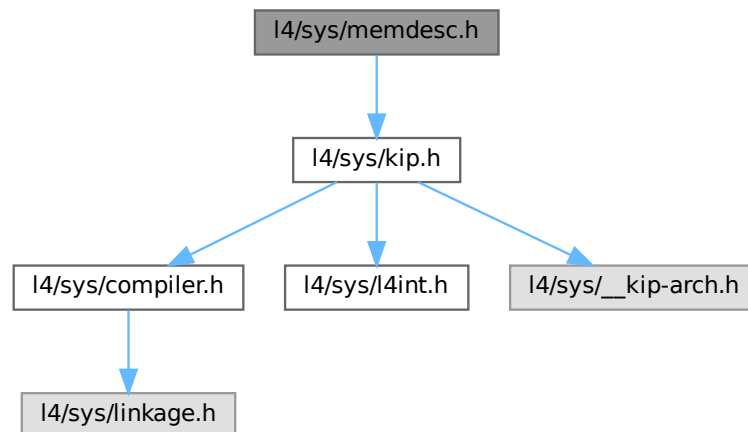
```

## 17.556 I4/sys/memdesc.h File Reference

Memory description functions.

```
#include <l4/sys/kip.h>
```

Include dependency graph for memdesc.h:



### Data Structures

- struct [l4\\_kernel\\_info\\_mem\\_desc\\_t](#)  
*Memory descriptor data structure.*

### Typedefs

- typedef struct `l4_kernel_info_mem_desc_t` [l4\\_kernel\\_info\\_mem\\_desc\\_t](#)  
*Memory descriptor data structure.*

### Enumerations

- enum [l4\\_mem\\_type\\_t](#) {  
[l4\\_mem\\_type\\_undefined](#) = 0x0 , [l4\\_mem\\_type\\_conventional](#) = 0x1 , [l4\\_mem\\_type\\_reserved](#) = 0x2 ,  
[l4\\_mem\\_type\\_dedicated](#) = 0x3 ,  
[l4\\_mem\\_type\\_shared](#) = 0x4 , [l4\\_mem\\_type\\_info](#) = 0xd , [l4\\_mem\\_type\\_bootloader](#) = 0xe , [l4\\_mem\\_type\\_archspecific](#)  
= 0xf }  
*Type of a memory descriptor.*
- enum [l4\\_mem\\_info\\_sub\\_type\\_t](#) { [l4\\_mem\\_info\\_acpi\\_rsd](#) = 0 }  
*Memory sub types for l4\_mem\_type\_info descriptors.*
- enum [l4\\_mem\\_archspecific\\_sub\\_type\\_common\\_t](#) { [l4\\_mem\\_archspecific\\_acpi\\_tables](#) = 3 , [l4\\_mem\\_archspecific\\_acpi\\_nvs](#)  
= 4 }  
*Memory sub types for l4\_mem\_type\_archspecific descriptors.*



## Functions

- [l4\\_kernel\\_info\\_mem\\_desc\\_t \\* l4\\_kernel\\_info\\_get\\_mem\\_descs](#) ([l4\\_kernel\\_info\\_t](#) \*kip) [L4\\_NOTHROW](#)  
*Get pointer to memory descriptors from KIP.*
- [unsigned l4\\_kernel\\_info\\_get\\_num\\_mem\\_descs](#) ([l4\\_kernel\\_info\\_t](#) \*kip) [L4\\_NOTHROW](#)  
*Get number of memory descriptors in KIP.*
- [void l4\\_kernel\\_info\\_set\\_mem\\_desc](#) ([l4\\_kernel\\_info\\_mem\\_desc\\_t](#) \*md, [l4\\_addr\\_t](#) start, [l4\\_addr\\_t](#) end, unsigned type, unsigned virt, unsigned sub\_type) [L4\\_NOTHROW](#)  
*Populate a memory descriptor.*
- [l4\\_umword\\_t l4\\_kernel\\_info\\_get\\_mem\\_desc\\_start](#) ([l4\\_kernel\\_info\\_mem\\_desc\\_t](#) \*md) [L4\\_NOTHROW](#)  
*Get start address of the region described by the memory descriptor.*
- [l4\\_umword\\_t l4\\_kernel\\_info\\_get\\_mem\\_desc\\_end](#) ([l4\\_kernel\\_info\\_mem\\_desc\\_t](#) \*md) [L4\\_NOTHROW](#)  
*Get end address of the region described by the memory descriptor.*
- [l4\\_umword\\_t l4\\_kernel\\_info\\_get\\_mem\\_desc\\_type](#) ([l4\\_kernel\\_info\\_mem\\_desc\\_t](#) \*md) [L4\\_NOTHROW](#)  
*Get type of the memory region.*
- [l4\\_umword\\_t l4\\_kernel\\_info\\_get\\_mem\\_desc\\_subtype](#) ([l4\\_kernel\\_info\\_mem\\_desc\\_t](#) \*md) [L4\\_NOTHROW](#)  
*Get sub-type of memory region.*
- [l4\\_umword\\_t l4\\_kernel\\_info\\_get\\_mem\\_desc\\_is\\_virtual](#) ([l4\\_kernel\\_info\\_mem\\_desc\\_t](#) \*md) [L4\\_NOTHROW](#)  
*Get virtual flag of the memory descriptor.*

### 17.556.1 Detailed Description

Memory description functions.

Definition in file [memdesc.h](#).

## 17.557 memdesc.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2007-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #ifndef __L4SYS_MEMDESC_H__
00014 #define __L4SYS_MEMDESC_H__
00015
00016 #include <l4/sys/kip.h>
00017
00028
00033 enum l4_mem_type_t
00034 {
00035 l4_mem_type_undefined = 0x0,
00036 l4_mem_type_conventional = 0x1,
00037 l4_mem_type_reserved = 0x2,
00038 l4_mem_type_dedicated = 0x3,
00039 l4_mem_type_shared = 0x4,
00040
00041 l4_mem_type_info = 0xd,
00042 l4_mem_type_bootloader = 0xe,
00043 l4_mem_type_archspecific = 0xf,
00044 };
00045
00050 enum l4_mem_info_sub_type_t
00051 {
00052 l4_mem_info_acpi_rsdp = 0
00053 };
00054
00059 enum l4_mem_archspecific_sub_type_common_t

```

```

00060 {
00061 l4_mem_archspecific_acpi_tables = 3,
00062 l4_mem_archspecific_acpi_nvs = 4,
00063 };
00064
00065
00073 typedef struct l4_kernel_info_mem_desc_t
00074 {
00076 l4_umword_t l;
00078 l4_umword_t h;
00079 } l4_kernel_info_mem_desc_t;
00080
00081
00086 L4_INLINE
00087 l4_kernel_info_mem_desc_t *
00088 l4_kernel_info_get_mem_descs(l4_kernel_info_t *kip) L4_NOTHROW;
00089
00096 L4_INLINE
00097 unsigned
00098 l4_kernel_info_get_num_mem_descs(l4_kernel_info_t *kip) L4_NOTHROW;
00099
00111 L4_INLINE
00112 void
00113 l4_kernel_info_set_mem_desc(l4_kernel_info_mem_desc_t *md,
00114 l4_addr_t start,
00115 l4_addr_t end,
00116 unsigned type,
00117 unsigned virt,
00118 unsigned sub_type) L4_NOTHROW;
00119
00126 L4_INLINE
00127 l4_umword_t
00128 l4_kernel_info_get_mem_desc_start(l4_kernel_info_mem_desc_t *md) L4_NOTHROW;
00129
00136 L4_INLINE
00137 l4_umword_t
00138 l4_kernel_info_get_mem_desc_end(l4_kernel_info_mem_desc_t *md) L4_NOTHROW;
00139
00146 L4_INLINE
00147 l4_umword_t
00148 l4_kernel_info_get_mem_desc_type(l4_kernel_info_mem_desc_t *md) L4_NOTHROW;
00149
00159 L4_INLINE
00160 l4_umword_t
00161 l4_kernel_info_get_mem_desc_subtype(l4_kernel_info_mem_desc_t *md) L4_NOTHROW;
00162
00169 L4_INLINE
00170 l4_umword_t
00171 l4_kernel_info_get_mem_desc_is_virtual(l4_kernel_info_mem_desc_t *md) L4_NOTHROW;
00172
00173 /*****
00174 * Implementations
00175 *****/
00176
00177 L4_INLINE
00178 l4_kernel_info_mem_desc_t *
00179 l4_kernel_info_get_mem_descs(l4_kernel_info_t *kip) L4_NOTHROW
00180 {
00181 return (l4_kernel_info_mem_desc_t *) ((l4_addr_t)kip + kip->mem_descs);
00182 }
00183
00184 L4_INLINE
00185 unsigned
00186 l4_kernel_info_get_num_mem_descs(l4_kernel_info_t *kip) L4_NOTHROW
00187 {
00188 return kip->mem_descs_num;
00189 }
00190
00191 L4_INLINE
00192 void
00193 l4_kernel_info_set_mem_desc(l4_kernel_info_mem_desc_t *md,
00194 l4_addr_t start,
00195 l4_addr_t end,
00196 unsigned type,
00197 unsigned virt,
00198 unsigned sub_type) L4_NOTHROW
00199 {
00200 md->l = (start & ~0x3ffUL) | (type & 0x0f) | ((sub_type << 4) & 0x0f0)
00201 | (virt ? 0x200 : 0x0);
00202 md->h = end;
00203 }
00204
00205
00206 L4_INLINE
00207 l4_umword_t
00208 l4_kernel_info_get_mem_desc_start(l4_kernel_info_mem_desc_t *md) L4_NOTHROW
00209 {

```

```

00210 return md->l & ~0x3ffUL;
00211 }
00212
00213 L4_INLINE
00214 l4_umword_t
00215 l4_kernel_info_get_mem_desc_end(l4_kernel_info_mem_desc_t *md) L4_NOTHROW
00216 {
00217 return md->h | 0x3ffUL;
00218 }
00219
00220 L4_INLINE
00221 l4_umword_t
00222 l4_kernel_info_get_mem_desc_type(l4_kernel_info_mem_desc_t *md) L4_NOTHROW
00223 {
00224 return md->l & 0xf;
00225 }
00226
00227 L4_INLINE
00228 l4_umword_t
00229 l4_kernel_info_get_mem_desc_subtype(l4_kernel_info_mem_desc_t *md) L4_NOTHROW
00230 {
00231 return (md->l & 0xf0) >> 4;
00232 }
00233
00234 L4_INLINE
00235 l4_umword_t
00236 l4_kernel_info_get_mem_desc_is_virtual(l4_kernel_info_mem_desc_t *md) L4_NOTHROW
00237 {
00238 return md->l & 0x200;
00239 }
00240
00241 #endif /* ! __L4SYS__MEMDESC_H__ */

```

## 17.558 meta

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002
00003 #pragma once
00004
00005 #include <l4/sys/meta>
00006 #include <l4/sys/typeinfo_svr>
00007
00008 namespace L4Re { namespace Util {
00009 using L4::Util::handle_meta_request;
00010 }}

```

## 17.559 l4/sys/meta File Reference

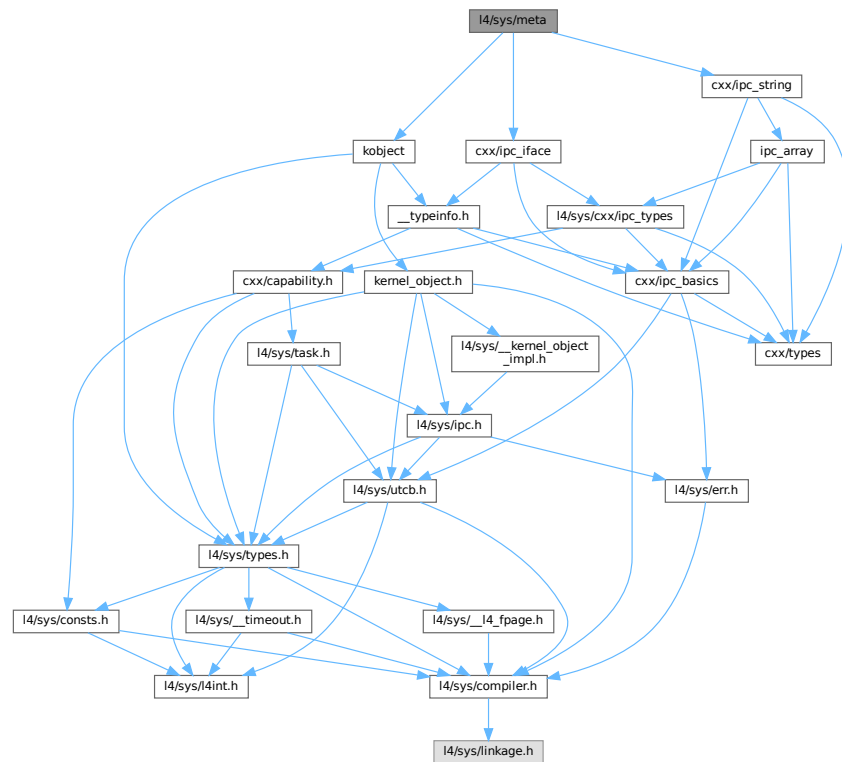
Meta interface for getting dynamic type information about objects behind capabilities.

```

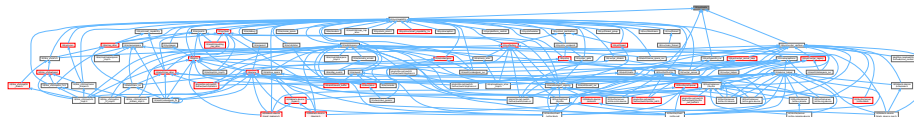
#include "kobject"
#include "cxx/ipc_iface"
#include "cxx/ipc_string"

```

Include dependency graph for meta:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class [L4::Meta](#)  
*Meta* interface that shall be implemented by each [L4Re](#) object and gives access to the dynamic type information for [L4Re](#) objects.

## Namespaces

- namespace [L4](#)  
*L4* low-level kernel interface.

## 17.559.1 Detailed Description

Meta interface for getting dynamic type information about objects behind capabilities.

Definition in file [meta](#).

## 17.560 meta

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008
00009 #pragma once
00010
00011 #include "kobject"
00012 #include "cxx/ipc_iface"
00013 #include "cxx/ipc_string"
00014
00015 namespace L4 {
00016
00017 class Meta : public Kobject_t<Meta, Kobject, L4_PROTO_META>
00018 {
00019 public:
00020 L4_INLINE_RPC(l4_msgtag_t, num_interfaces, ());
00021 L4_INLINE_RPC(l4_msgtag_t, interface, (l4_umword_t idx, long *proto,
00022 L4::Ipc::String<char> *name));
00023 L4_INLINE_RPC(l4_msgtag_t, supports, (l4_mword_t protocol));
00024 typedef L4::Typeid::Rpc<num_interfaces_t, interface_t, supports_t> Rpc;
00025 };
00026
00027 }

```

## 17.561 l4/sys/obj\_info.h File Reference

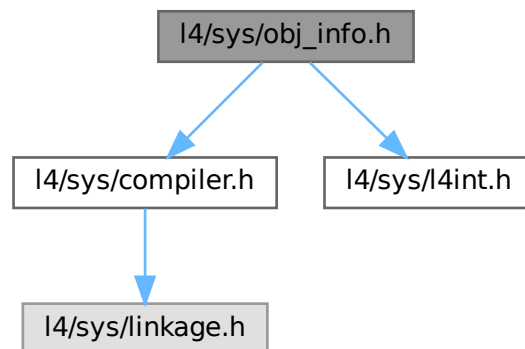
Debugger related functions.

```

#include <l4/sys/compiler.h>
#include <l4/sys/l4int.h>

```

Include dependency graph for obj\_info.h:



## Functions

- [l4\\_msgtag\\_t](#) [l4\\_debugger\\_query\\_obj\\_infos](#) ([l4\\_cap\\_idx\\_t](#) cap, [l4\\_addr\\_t](#) ku\_mem\_addr, [l4\\_size\\_t](#) ku\_mem\_size, [l4\\_umword\\_t](#) skip, [l4\\_umword\\_t](#) \*result\_cnt, [l4\\_umword\\_t](#) \*result\_all) [L4\\_NOTHROW](#)

*Retrieve information from the kernel about all objects in the mapping database and write data to the passed KU memory.*

### 17.561.1 Detailed Description

Debugger related functions.

#### Attention

This API is subject to change!

Definition in file [obj\\_info.h](#).

### 17.561.2 Function Documentation

#### 17.561.2.1 l4\_debugger\_query\_obj\_infos()

```
l4_msgtag_t l4_debugger_query_obj_infos (
 l4_cap_idx_t cap,
 l4_addr_t ku_mem_addr,
 l4_size_t ku_mem_size,
 l4_umword_t skip,
 l4_umword_t * result_cnt,
 l4_umword_t * result_all) [inline]
```

Retrieve information from the kernel about all objects in the mapping database and write data to the passed KU memory.

#### Parameters

|     |                    |                                                       |
|-----|--------------------|-------------------------------------------------------|
|     | <i>cap</i>         | Capability of the debugger object.                    |
|     | <i>ku_mem_addr</i> | Address of the KU memory for writing the information. |
|     | <i>ku_mem_size</i> | Size of the KU memory to writing the information.     |
|     | <i>skip</i>        | Number of objects to skip.                            |
| out | <i>result_cnt</i>  | Number of objects in the mapping database.            |
| out | <i>result_all</i>  | Number of objects written to the KU memory.           |

**Note**

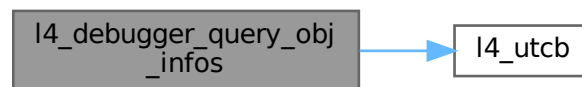
The kernel will only write a number of object information which fits to the passed KU memory. To retrieve missing object information, repeat the call and adapt the `skip` parameter accordingly.

If this system call is performed several times, the number of kernel objects might have changed in the meantime.

Definition at line 176 of file [obj\\_info.h](#).

References [L4\\_NOTHROW](#), and [l4\\_utcb\(\)](#).

Here is the call graph for this function:



## 17.562 obj\_info.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * Copyright (C) 2023 Kernkonzept GmbH.
00003 * Author(s): Frank Mehnert <frank.mehnert@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00014
00015 #pragma once
00016
00017 #include <l4/sys/compiler.h>
00018 #include <l4/sys/l4int.h>
00019
00020 struct L4_kobj_info
00021 {
00022 // Type_mapping: See Jdb_mapdb::info_obj_mapping().
00023 struct Mapping
00024 {
00025 enum { Type = 0 };
00026 l4_uint64_t mapping_ptr;
00027 char space_name[16];
00028 l4_uint32_t cap_idx;
00029 l4_uint16_t entry_rights;
00030 l4_uint16_t entry_flags;
00031 l4_uint64_t entry_ptr;
00032 };
00033
00034 // Type_thread: See Jdb_tcb::info_kobject().
00035 struct Thread
00036 {
00037 enum { Type = 1 };
00038 bool is_kernel;
00039 bool is_current;
00040 bool in_ready_list;
00041 bool is_kernel_task;
00042 l4_uint32_t home_cpu;
00043 l4_uint32_t current_cpu;
00044 l4_int64_t ref_cnt;
00045 l4_uint64_t space_id;
00046 };
00047
00048 // Type_space: See Jdb_space::info_kobject().

```

```

00049 struct Space
00050 {
00051 enum { Type = 2 };
00052 bool is_kernel;
00053 l4_int64_t ref_cnt;
00054 };
00055
00056 // Type_vm: See Jdb_vm::info_kobject().
00057 struct Vm
00058 {
00059 enum { Type = 3 };
00060 l4_uint64_t utcb;
00061 l4_uint64_t pc;
00062 };
00063
00064 // Type_ipc_gate: See Jdb_ipc_gate::info_kobject().
00065 struct Ipc_gate
00066 {
00067 enum { Type = 4 };
00068 l4_uint64_t label;
00069 l4_uint64_t thread_id;
00070 };
00071
00072 // Type_irq: See Jdb_kobject_irq::info_kobject().
00073 struct Irq_sender
00074 {
00075 enum { Type = 5 };
00076 char chip_type[10];
00077 l4_uint16_t flags;
00078 l4_uint32_t pin;
00079 l4_uint64_t label;
00080 l4_uint64_t target_id;
00081 l4_int64_t queued;
00082 };
00083
00084 // Type_irq: See Jdb_kobject_irq::info_kobject().
00085 struct Irq_semaphore
00086 {
00087 enum { Type = 6 };
00088 char chip_type[10];
00089 l4_uint16_t flags;
00090 l4_uint32_t pin;
00091 l4_uint64_t sender_id;
00092 l4_uint64_t target_id;
00093 l4_int64_t queued;
00094 };
00095
00096 // Type_factory: See Jdb_factory::info_kobject().
00097 struct Factory
00098 {
00099 enum { Type = 7 };
00100 l4_uint64_t current;
00101 l4_uint64_t limit;
00102 };
00103
00104 struct Jdb { enum { Type = 8 }; };
00105 struct Scheduler { enum { Type = 9 }; };
00106 struct Vlog { enum { Type = 10 }; };
00107 struct Pfc { enum { Type = 11 }; };
00108 struct Dmar_space { enum { Type = 12 }; };
00109 struct Iommu { enum { Type = 13 }; };
00110 struct Smmu { enum { Type = 14 }; };
00111
00112 l4_uint64_t type:5;
00113 l4_uint64_t id:59;
00114 l4_uint64_t mapping_ptr;
00115 l4_uint64_t ref_cnt;
00116 union
00117 {
00118 Thread thread;
00119 Space space;
00120 Vm vm;
00121 Ipc_gate ipc_gate;
00122 Irq_sender irq_sender;
00123 Irq_semaphore irq_semaphore;
00124 Factory factory;
00125 Mapping mapping;
00126 l4_uint64_t raw[5];
00127 };
00128 };
00129
00130 static_assert(sizeof(L4_kobj_info) == 64, "Size of Jobb_info");
00131
00150 L4_INLINE l4_msgtag_t
00151 l4_debugger_query_obj_infos(l4_cap_idx_t cap, l4_addr_t ku_mem_addr,
00152 l4_size_t ku_mem_size, l4_umword_t skip,
00153 l4_umword_t *result_cnt, l4_umword_t *result_all)

```



```

00154 L4_NOTHROW;
00155
00156 L4_INLINE l4_msgtag_t
00157 l4_debugger_query_obj_infos_u(l4_cap_idx_t cap, l4_addr_t ku_mem_addr,
00158 l4_size_t ku_mem_size, l4_umword_t skip,
00159 l4_umword_t *result_cnt, l4_umword_t *result_all,
00160 l4_utcb_t *utcb) L4_NOTHROW
00161 {
00162 l4_utcb_mr()->mr[0] = 16;
00163 l4_utcb_mr()->mr[1] = ku_mem_addr;
00164 l4_utcb_mr()->mr[2] = ku_mem_size;
00165 l4_utcb_mr()->mr[3] = skip;
00166
00167 l4_msgtag_t tag = l4_invoke_debugger(cap, l4_msgtag(0, 4, 0, 0), utcb);
00168
00169 *result_cnt = l4_utcb_mr()->mr[0];
00170 *result_all = l4_utcb_mr()->mr[1];
00171
00172 return tag;
00173 }
00174
00175 L4_INLINE l4_msgtag_t
00176 l4_debugger_query_obj_infos(l4_cap_idx_t cap, l4_addr_t ku_mem_addr,
00177 l4_size_t ku_mem_size, l4_umword_t skip,
00178 l4_umword_t *result_cnt, l4_umword_t *result_all)
00179 L4_NOTHROW
00180 {
00181 return l4_debugger_query_obj_infos_u(cap, ku_mem_addr, ku_mem_size, skip,
00182 result_cnt, result_all, l4_utcb());
00183 }

```

## 17.563 l4/sys/pager File Reference

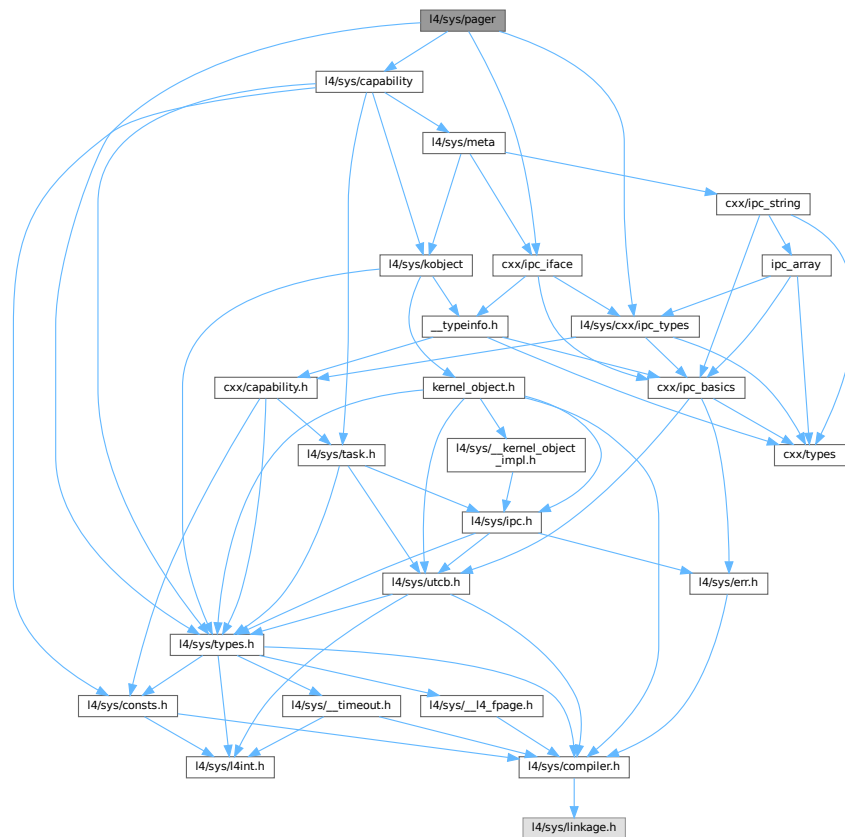
Pager and lo\_pager C++ interface.

```

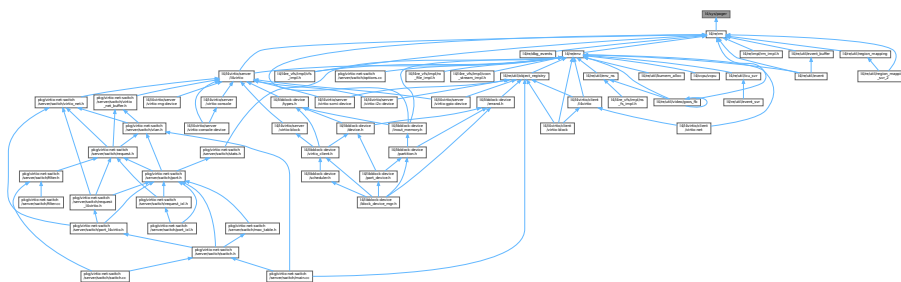
#include <l4/sys/capability>
#include <l4/sys/types.h>
#include <l4/sys/cxx/ipc_types>
#include <l4/sys/cxx/ipc_iface>

```

Include dependency graph for pager:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class [L4::lo\\_pager](#)  
[lo\\_pager](#) interface.
- class [L4::Pager](#)  
[Pager](#) interface including the [lo\\_pager](#) interface.

## Namespaces

- namespace [L4](#)  
[L4](#) low-level kernel interface.

## 17.563.1 Detailed Description

Pager and Io\_pager C++ interface.

Definition in file [pager](#).

## 17.564 pager

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2014 Alexander Warg <alexander.warg@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #pragma once
00009
00010 #include <l4/sys/capability>
00011 #include <l4/sys/types.h>
00012 #include <l4/sys/cxx/ipc_types>
00013 #include <l4/sys/cxx/ipc_iface>
00014
00015 namespace L4 {
00016
00017 class L4_EXPORT Io_pager :
00018 public Kobject_0t<Io_pager, L4_PROTO_IO_PAGE_FAULT>
00019 {
00020 public:
00021 L4_INLINE_RPC(
00022 l4_msgtag_t, io_page_fault, (l4_fpage_t io_pfa, l4_umword_t pc,
00023 L4::Ipc::Rcv_fpage rwin,
00024 L4::Ipc::Opt<L4::Ipc::Snd_fpage &> fp));
00025
00026 typedef L4::Typeid::Rpc_nocode<io_page_fault_t> Rpcs;
00027 };
00028
00029 class L4_EXPORT Pager :
00030 public Kobject_t<Pager, Io_pager, L4_PROTO_PAGE_FAULT>
00031 {
00032 public:
00033 L4_INLINE_RPC(
00034 l4_msgtag_t, page_fault, (l4_umword_t pfa, l4_umword_t pc,
00035 L4::Ipc::Rcv_fpage rwin,
00036 L4::Ipc::Opt<L4::Ipc::Snd_fpage &> fp));
00037
00038 typedef L4::Typeid::Rpc_nocode<page_fault_t> Rpcs;
00039 };
00040
00041 }
00042

```

## 17.565 l4/sys/platform\_control File Reference

Platform control object.

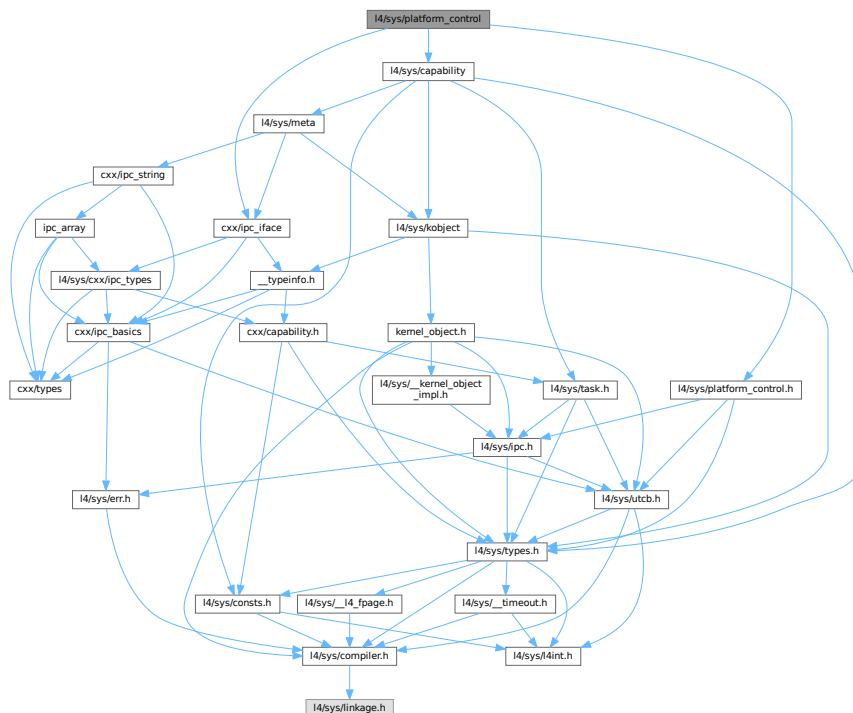
```

#include <l4/sys/capability>
#include <l4/sys/platform_control.h>

```

```
#include <l4/sys/cxx/ipc_iface>
```

Include dependency graph for platform\_control:



## Data Structures

- class [L4::Platform\\_control](#)  
[L4](#) C++ interface for controlling platform-wide properties, see [Platform Control C API](#) for the C interface.

## Namespaces

- namespace [L4](#)  
[L4](#) low-level kernel interface.

### 17.565.1 Detailed Description

Platform control object.

Definition in file [platform\\_control](#).

## 17.566 platform\_control

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2014 Steffen Liebergeld <steffen.liebergeld@kernkonzept.com>
00004 * Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00005 * Alexander Warg <warg@os.inf.tu-dresden.de>
00006 * economic rights: Technische Universität Dresden (Germany)
00007 *
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010
00011 #pragma once
00012
00013 #include <l4/sys/capability>
00014 #include <l4/sys/platform_control.h>
00015 #include <l4/sys/cxx/ipc_iface>
00016
00017 namespace L4 {
00018
00019 class L4_EXPORT Platform_control
00020 : public Kobject_t<Platform_control, Kobject, L4_PROTO_PLATFORM_CTL>
00021 {
00022 public:
00023 L4_INLINE_RPC_OP(L4_PLATFORM_CTL_SYS_SUSPEND_OP,
00024 l4_msgtag_t, system_suspend, (l4_umword_t extras));
00025
00026 L4_INLINE_RPC_OP(L4_PLATFORM_CTL_SYS_SHUTDOWN_OP,
00027 l4_msgtag_t, system_shutdown, (l4_umword_t reboot));
00028
00029 L4_INLINE_RPC_OP(L4_PLATFORM_CTL_CPU_ALLOW_SHUTDOWN_OP,
00030 l4_msgtag_t, cpu_allow_shutdown,
00031 (l4_umword_t phys_id, l4_umword_t enable));
00032
00033 L4_INLINE_RPC_OP(L4_PLATFORM_CTL_CPU_ENABLE_OP,
00034 l4_msgtag_t, cpu_enable, (l4_umword_t phys_id));
00035
00036 L4_INLINE_RPC_OP(L4_PLATFORM_CTL_CPU_DISABLE_OP,
00037 l4_msgtag_t, cpu_disable, (l4_umword_t phys_id));
00038
00039 typedef L4::Typeid::Rpcs_sys<system_suspend_t, system_shutdown_t,
00040 cpu_allow_shutdown_t, cpu_enable_t,
00041 cpu_disable_t> Rpcs;
00042 };
00043
00044 }
00045
00046
00047
00048

```

## 17.567 platform\_control.h

```

00001 /*
00002 * Copyright (C) 2024 Kernkonzept GmbH.
00003 * Author(s): Jan Klötzke <jan.kloetzke@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include_next <l4/sys/platform_control.h>
00010 #include <l4/sys/__platform_control-arm.h>

```

## 17.568 platform\_control.h

```

00001 /*
00002 * Copyright (C) 2024 Kernkonzept GmbH.
00003 * Author(s): Jan Klötzke <jan.kloetzke@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include_next <l4/sys/platform_control.h>
00010 #include <l4/sys/__platform_control-arm.h>

```

## 17.569 I4/sys/platform\_control.h File Reference

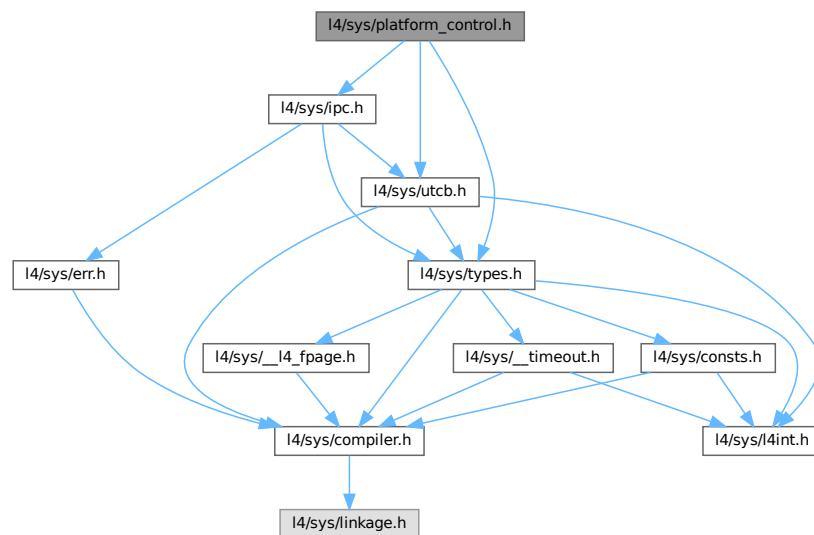
Platform control object.

```
#include <l4/sys/types.h>
```

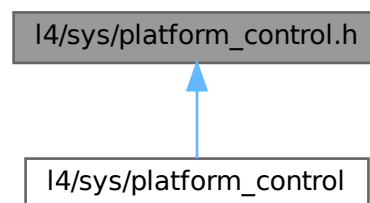
```
#include <l4/sys/utcb.h>
```

```
#include <l4/sys/ipc.h>
```

Include dependency graph for platform\_control.h:



This graph shows which files directly or indirectly include this file:



### Enumerations

- enum [L4\\_platform\\_ctl\\_ops](#) {  
[L4\\_PLATFORM\\_CTL\\_SYS\\_SUSPEND\\_OP](#) = 0UL , [L4\\_PLATFORM\\_CTL\\_SYS\\_SHUTDOWN\\_OP](#) = 1UL ,  
[L4\\_PLATFORM\\_CTL\\_CPU\\_ALLOW\\_SHUTDOWN\\_OP](#) = 2UL , [L4\\_PLATFORM\\_CTL\\_CPU\\_ENABLE\\_OP](#) = 3UL ,  
[L4\\_PLATFORM\\_CTL\\_CPU\\_DISABLE\\_OP](#) = 4UL , [L4\\_PLATFORM\\_CTL\\_SET\\_TASK\\_ASID\\_OP](#) = 0x10UL }

*Operations on platform-control objects.*

- enum [L4\\_platform\\_ctl\\_proto](#) { [L4\\_PROTO\\_PLATFORM\\_CTL](#) = 0 }

*Predefined protocol type for messages to platform-control objects.*

## Functions

- [l4\\_msgtag\\_t l4\\_platform\\_ctl\\_system\\_suspend \(l4\\_cap\\_idx\\_t pfc, l4\\_umword\\_t extras\) L4\\_NOTHROW](#)  
*Enter suspend to RAM.*
- [l4\\_msgtag\\_t l4\\_platform\\_ctl\\_system\\_shutdown \(l4\\_cap\\_idx\\_t pfc, l4\\_umword\\_t reboot\) L4\\_NOTHROW](#)  
*Shutdown or reboot the system.*
- [l4\\_msgtag\\_t l4\\_platform\\_ctl\\_cpu\\_allow\\_shutdown \(l4\\_cap\\_idx\\_t pfc, l4\\_umword\\_t phys\\_id, l4\\_umword\\_t enable\) L4\\_NOTHROW](#)  
*Allow a CPU to be shut down.*
- [l4\\_msgtag\\_t l4\\_platform\\_ctl\\_cpu\\_enable \(l4\\_cap\\_idx\\_t pfc, l4\\_umword\\_t phys\\_id\) L4\\_NOTHROW](#)  
*Enable an offline CPU.*
- [l4\\_msgtag\\_t l4\\_platform\\_ctl\\_cpu\\_disable \(l4\\_cap\\_idx\\_t pfc, l4\\_umword\\_t phys\\_id\) L4\\_NOTHROW](#)  
*Disable an online CPU.*

## 17.569.1 Detailed Description

Platform control object.

Definition in file [platform\\_control.h](#).

## 17.570 platform\_control.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2014 Alexander Warg <alexander.warg@kernkonzept.com>
00007 *
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010
00011 #pragma once
00012
00013 #include <l4/sys/types.h>
00014 #include <l4/sys/utcb.h>
00015
00030
00031
00049 L4_INLINE l4_msgtag_t
00050 l4_platform_ctl_system_suspend(l4_cap_idx_t pfc,
00051 l4_umword_t extras) L4_NOTHROW;
00052
00056 L4_INLINE l4_msgtag_t
00057 l4_platform_ctl_system_suspend_u(l4_cap_idx_t pfc,
00058 l4_umword_t extras,
00059 l4_utcb_t *utcb) L4_NOTHROW;
00060
00061
00070 L4_INLINE l4_msgtag_t
00071 l4_platform_ctl_system_shutdown(l4_cap_idx_t pfc,
00072 l4_umword_t reboot) L4_NOTHROW;
00073
00077 L4_INLINE l4_msgtag_t
00078 l4_platform_ctl_system_shutdown_u(l4_cap_idx_t pfc,
00079 l4_umword_t reboot,
00080 l4_utcb_t *utcb) L4_NOTHROW;
00081
00091 L4_INLINE l4_msgtag_t
00092 l4_platform_ctl_cpu_allow_shutdown(l4_cap_idx_t pfc,
00093 l4_umword_t phys_id,
00094 l4_umword_t enable) L4_NOTHROW;
00095
00099 L4_INLINE l4_msgtag_t
00100 l4_platform_ctl_cpu_allow_shutdown_u(l4_cap_idx_t pfc,
00101 l4_umword_t phys_id,
00102 l4_umword_t enable,
00103 l4_utcb_t *utcb) L4_NOTHROW;
00104
00114 L4_INLINE l4_msgtag_t

```

```

00115 l4_platform_ctl_cpu_enable(l4_cap_idx_t pfc,
00116 l4_umword_t phys_id) L4_NOTHROW;
00117
00121 L4_INLINE l4_msgtag_t
00122 l4_platform_ctl_cpu_enable_u(l4_cap_idx_t pfc,
00123 l4_umword_t phys_id,
00124 l4_utcb_t *utcb) L4_NOTHROW;
00125
00136 L4_INLINE l4_msgtag_t
00137 l4_platform_ctl_cpu_disable(l4_cap_idx_t pfc,
00138 l4_umword_t phys_id) L4_NOTHROW;
00139
00143 L4_INLINE l4_msgtag_t
00144 l4_platform_ctl_cpu_disable_u(l4_cap_idx_t pfc,
00145 l4_umword_t phys_id,
00146 l4_utcb_t *utcb) L4_NOTHROW;
00147 /* ends l4_platform_control_api group */
00149
00150
00159 enum L4_platform_ctl_ops
00160 {
00161 L4_PLATFORM_CTL_SYS_SUSPEND_OP = 0UL,
00162 L4_PLATFORM_CTL_SYS_SHUTDOWN_OP = 1UL,
00163 L4_PLATFORM_CTL_CPU_ALLOW_SHUTDOWN_OP = 2UL,
00164 L4_PLATFORM_CTL_CPU_ENABLE_OP = 3UL,
00165 L4_PLATFORM_CTL_CPU_DISABLE_OP = 4UL,
00166
00167 L4_PLATFORM_CTL_SET_TASK_ASID_OP = 0x10UL,
00168 };
00169
00174 enum L4_platform_ctl_proto
00175 {
00181 L4_PROTO_PLATFORM_CTL = 0
00182 };
00183
00184 /* IMPLEMENTATION -----*/
00185
00186 #include <l4/sys/ipc.h>
00187
00188 L4_INLINE l4_msgtag_t
00189 l4_platform_ctl_system_suspend_u(l4_cap_idx_t pfc,
00190 l4_umword_t extras,
00191 l4_utcb_t *utcb) L4_NOTHROW
00192 {
00193 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00194 v->mr[0] = L4_PLATFORM_CTL_SYS_SUSPEND_OP;
00195 v->mr[1] = extras;
00196 return l4_ipc_call(pfc, utcb, l4_msgtag(L4_PROTO_PLATFORM_CTL, 2, 0, 0),
00197 L4_IPC_NEVER);
00198 }
00199
00200 L4_INLINE l4_msgtag_t
00201 l4_platform_ctl_system_shutdown_u(l4_cap_idx_t pfc,
00202 l4_umword_t reboot,
00203 l4_utcb_t *utcb) L4_NOTHROW
00204 {
00205 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00206 v->mr[0] = L4_PLATFORM_CTL_SYS_SHUTDOWN_OP;
00207 v->mr[1] = reboot;
00208 return l4_ipc_call(pfc, utcb, l4_msgtag(L4_PROTO_PLATFORM_CTL, 2, 0, 0),
00209 L4_IPC_NEVER);
00210 }
00211
00212
00213 L4_INLINE l4_msgtag_t
00214 l4_platform_ctl_system_suspend(l4_cap_idx_t pfc,
00215 l4_umword_t extras) L4_NOTHROW
00216 {
00217 return l4_platform_ctl_system_suspend_u(pfc, extras, l4_utcb());
00218 }
00219
00220 L4_INLINE l4_msgtag_t
00221 l4_platform_ctl_system_shutdown(l4_cap_idx_t pfc,
00222 l4_umword_t reboot) L4_NOTHROW
00223 {
00224 return l4_platform_ctl_system_shutdown_u(pfc, reboot, l4_utcb());
00225 }
00226
00227 L4_INLINE l4_msgtag_t
00228 l4_platform_ctl_cpu_allow_shutdown_u(l4_cap_idx_t pfc,
00229 l4_umword_t phys_id,
00230 l4_umword_t enable,
00231 l4_utcb_t *utcb) L4_NOTHROW
00232 {
00233 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00234 v->mr[0] = L4_PLATFORM_CTL_CPU_ALLOW_SHUTDOWN_OP;
00235 v->mr[1] = phys_id;

```



```

00236 v->mr[2] = enable;
00237 return l4_ipc_call(pfc, utcb, l4_msgtag(L4_PROTO_PLATFORM_CTL, 3, 0, 0),
00238 L4_IPC_NEVER);
00239 }
00240
00241 L4_INLINE l4_msgtag_t
00242 l4_platform_ctl_cpu_allow_shutdown(l4_cap_idx_t pfc,
00243 l4_umword_t phys_id,
00244 l4_umword_t enable) L4_NOTHROW
00245 {
00246 return l4_platform_ctl_cpu_allow_shutdown_u(pfc, phys_id, enable, l4_utcb());
00247 }
00248
00249 L4_INLINE l4_msgtag_t
00250 l4_platform_ctl_cpu_enable_u(l4_cap_idx_t pfc,
00251 l4_umword_t phys_id,
00252 l4_utcb_t *utcb) L4_NOTHROW
00253 {
00254 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00255 v->mr[0] = L4_PLATFORM_CTL_CPU_ENABLE_OP;
00256 v->mr[1] = phys_id;
00257 return l4_ipc_call(pfc, utcb, l4_msgtag(L4_PROTO_PLATFORM_CTL, 2, 0, 0),
00258 L4_IPC_NEVER);
00259 }
00260
00261 L4_INLINE l4_msgtag_t
00262 l4_platform_ctl_cpu_disable_u(l4_cap_idx_t pfc,
00263 l4_umword_t phys_id,
00264 l4_utcb_t *utcb) L4_NOTHROW
00265 {
00266 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00267 v->mr[0] = L4_PLATFORM_CTL_CPU_DISABLE_OP;
00268 v->mr[1] = phys_id;
00269 return l4_ipc_call(pfc, utcb, l4_msgtag(L4_PROTO_PLATFORM_CTL, 2, 0, 0),
00270 L4_IPC_NEVER);
00271 }
00272
00273 L4_INLINE l4_msgtag_t
00274 l4_platform_ctl_cpu_enable(l4_cap_idx_t pfc,
00275 l4_umword_t phys_id) L4_NOTHROW
00276 {
00277 return l4_platform_ctl_cpu_enable_u(pfc, phys_id, l4_utcb());
00278 }
00279
00280 L4_INLINE l4_msgtag_t
00281 l4_platform_ctl_cpu_disable(l4_cap_idx_t pfc,
00282 l4_umword_t phys_id) L4_NOTHROW
00283 {
00284 return l4_platform_ctl_cpu_disable_u(pfc, phys_id, l4_utcb());
00285 }

```

## 17.571 l4/sys/rcv\_endpoint File Reference

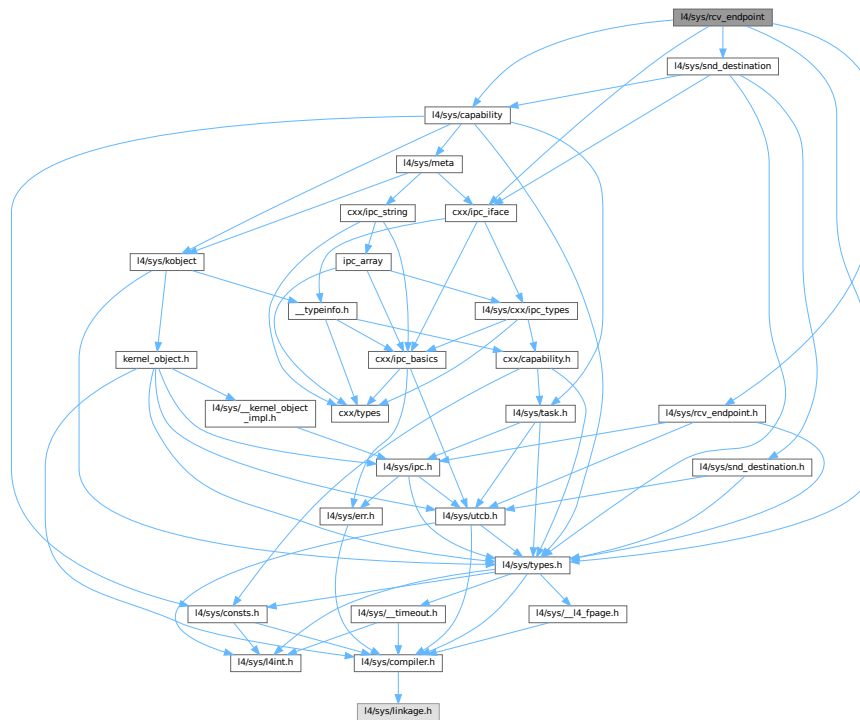
The C++ Receive endpoint interface.

```

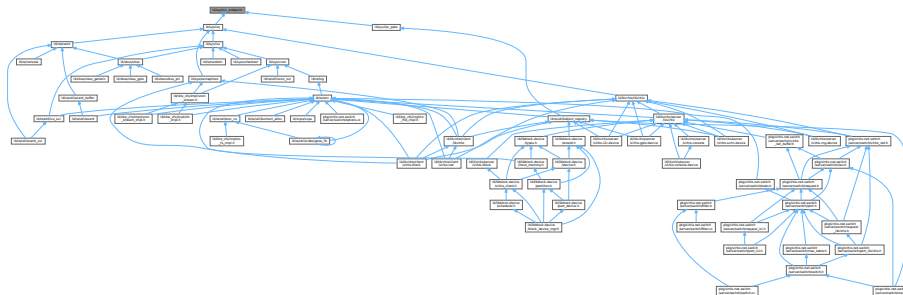
#include <l4/sys/rcv_endpoint.h>
#include <l4/sys/snd_destination>
#include <l4/sys/types.h>
#include <l4/sys/capability>
#include <l4/sys/cxx/ipc_iface>

```

Include dependency graph for `rcv_endpoint`:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class [L4::Rcv\\_endpoint](#)  
*Interface for kernel objects that allow to receive IPC from them.*

## Namespaces

- namespace [L4](#)  
*L4 low-level kernel interface.*

## 17.571.1 Detailed Description

The C++ Receive endpoint interface.

Definition in file [rcv\\_endpoint](#).

## 17.572 rcv\_endpoint

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2017 Alexander Warg <alexander.warg@kernkonzept.com>
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #pragma once
00012
00013 #include <l4/sys/rcv_endpoint.h>
00014 #include <l4/sys/snd_destination>
00015 #include <l4/sys/types.h>
00016 #include <l4/sys/capability>
00017 #include <l4/sys/cxx/ipc_iface>
00018
00019 namespace L4 {
00020
00021 class Thread;
00022
00030 class L4_EXPORT Rcv_endpoint :
00031 public Kobject_t<Rcv_endpoint, Kobject, L4_PROTO_KOBJECT,
00032 Type_info::Demand_t<1> >
00033 {
00034 public:
00066 L4_INLINE_RPC_OP(L4_RCV_EP_BIND_OP,
00067 l4_msgtag_t, bind_thread, (Ipc::Cap<Thread> t, l4_umword_t label));
00068
00101 l4_msgtag_t bind_snd_destination(Cap<Snd_destination> snd_dst, l4_umword_t label)
00102 {
00103 return l4_rcv_ep_bind_snd_destination(cap(), snd_dst.cap(), label);
00104 }
00105
00106 typedef L4::Typeid::Rpc_sys<bind_thread_t> Rpc;
00107 };
00108
00109 }
```

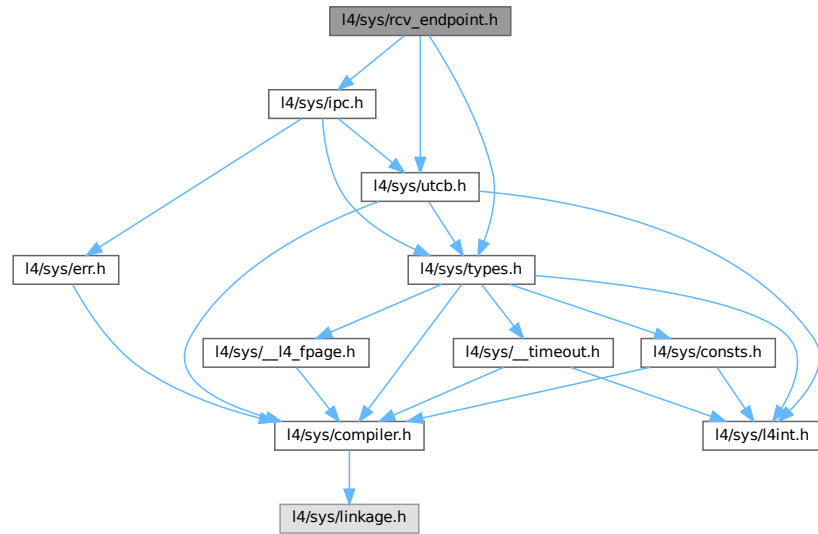
## 17.573 l4/sys/rcv\_endpoint.h File Reference

Receive endpoint C interface.

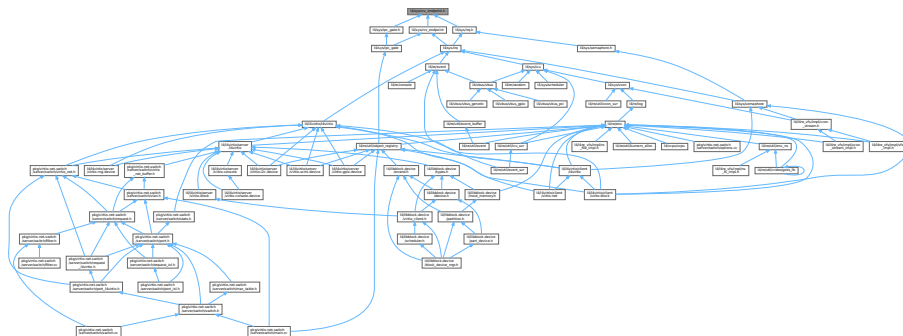
```

#include <l4/sys/utcb.h>
#include <l4/sys/types.h>
#include <l4/sys/ipc.h>
```

Include dependency graph for rcv\_endpoint.h:



This graph shows which files directly or indirectly include this file:



## Enumerations

- enum [L4\\_rcv\\_ep\\_ops](#) { [L4\\_RCV\\_EP\\_BIND\\_OP](#) = 0x10 }  
Receive endpoint operations.

## Functions

- [l4\\_msgtag\\_t l4\\_rcv\\_ep\\_bind\\_thread](#) ([l4\\_cap\\_idx\\_t](#) ep, [l4\\_cap\\_idx\\_t](#) thread, [l4\\_umword\\_t](#) label)  
Bind the IPC receive endpoint to a thread.
- [l4\\_msgtag\\_t l4\\_rcv\\_ep\\_bind\\_snd\\_destination](#) ([l4\\_cap\\_idx\\_t](#) ep, [l4\\_cap\\_idx\\_t](#) snd\_dst, [l4\\_umword\\_t](#) label)  
Bind the IPC receive endpoint to a send destination (a thread).

## 17.573.1 Detailed Description

Receive endpoint C interface.

Definition in file [rcv\\_endpoint.h](#).

## 17.573.2 Enumeration Type Documentation

### 17.573.2.1 L4\_rcv\_ep\_ops

enum [L4\\_rcv\\_ep\\_ops](#)

Receive endpoint operations.

#### Enumerator

|                   |                                 |
|-------------------|---------------------------------|
| L4_RCV_EP_BIND_OP | Bind to thread or thread group. |
|-------------------|---------------------------------|

Definition at line 93 of file [rcv\\_endpoint.h](#).

## 17.574 rcv\_endpoint.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2017 Alexander Warg <alexander.warg@kernkonzept.com>
00007 *
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010 #pragma once
00011
00012 #include <l4/sys/utcb.h>
00013 #include <l4/sys/types.h>
00014
00045 L4_INLINE l4_msgtag_t
00046 l4_rcv_ep_bind_thread(l4_cap_idx_t ep, l4_cap_idx_t thread,
00047 l4_umword_t label);
00048
00076 L4_INLINE l4_msgtag_t
00077 l4_rcv_ep_bind_snd_destination(l4_cap_idx_t ep, l4_cap_idx_t snd_dst,
00078 l4_umword_t label);
00079
00084 L4_INLINE l4_msgtag_t
00085 l4_rcv_ep_bind_thread_u(l4_cap_idx_t ep, l4_cap_idx_t thread,
00086 l4_umword_t label, l4_utcb_t *utcb);
00087
00088 L4_INLINE l4_msgtag_t
00089 l4_rcv_ep_bind_snd_destination_u(l4_cap_idx_t ep, l4_cap_idx_t snd_dst,
00090 l4_umword_t label, l4_utcb_t *utcb);
00091
00093 enum L4_rcv_ep_ops
00094 {
00095 L4_RCV_EP_BIND_OP = 0x10,
00096 };
00097
00098 /* IMPLEMENTATION -----*/
00099
00100 #include <l4/sys/ipc.h>
00101
00102 L4_INLINE l4_msgtag_t
00103 l4_rcv_ep_bind_thread_u(l4_cap_idx_t ep,
00104 l4_cap_idx_t thread, l4_umword_t label,
00105 l4_utcb_t *utcb)
00106 {
00107 l4_msg_regs_t *m = l4_utcb_mr_u(utcb);

```

```

00108 m->mr[0] = L4_RCV_EP_BIND_OP;
00109 m->mr[1] = label;
00110 m->mr[2] = l4_map_obj_control(0, 0);
00111 m->mr[3] = l4_obj_fpage(thread, 0, L4_CAP_FPAGE_RWS).raw;
00112 return l4_ipc_call(ep, utcb, l4_msgtag(L4_PROTO_KOBJECT, 2, 1, 0),
00113 L4_IPC_NEVER);
00114 }
00115
00116 L4_INLINE l4_msgtag_t
00117 l4_rcv_ep_bind_snd_destination_u(l4_cap_idx_t ep,
00118 l4_cap_idx_t snd_dst, l4_umword_t label,
00119 l4_utcb_t *utcb)
00120 {
00121 l4_msg_regs_t *m = l4_utcb_mr_u(utcb);
00122 m->mr[0] = L4_RCV_EP_BIND_OP;
00123 m->mr[1] = label;
00124 m->mr[2] = l4_map_obj_control(0, 0);
00125 m->mr[3] = l4_obj_fpage(snd_dst, 0, L4_CAP_FPAGE_RWS).raw;
00126 return l4_ipc_call(ep, utcb, l4_msgtag(L4_PROTO_KOBJECT, 2, 1, 0),
00127 L4_IPC_NEVER);
00128 }
00129
00130 L4_INLINE l4_msgtag_t
00131 l4_rcv_ep_bind_thread(l4_cap_idx_t ep, l4_cap_idx_t thread,
00132 l4_umword_t label)
00133 {
00134 return l4_rcv_ep_bind_thread_u(ep, thread, label, l4_utcb());
00135 }
00136
00137 L4_INLINE l4_msgtag_t
00138 l4_rcv_ep_bind_snd_destination(l4_cap_idx_t ep, l4_cap_idx_t snd_dst,
00139 l4_umword_t label)
00140 {
00141 return l4_rcv_ep_bind_snd_destination_u(ep, snd_dst, label, l4_utcb());
00142 }

```

## 17.575 l4/sys/scheduler File Reference

Scheduler object functions.

```

#include <l4/sys/icu>
#include <l4/sys/scheduler.h>
#include <l4/sys/capability>
#include <l4/sys/cxx/ipc_iface>

```



## 17.576 scheduler

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 #pragma once
00010
00011 #include <l4/sys/icu>
00012 #include <l4/sys/scheduler.h>
00013 #include <l4/sys/capability>
00014 #include <l4/sys/cxx/ipc_iface>
00015
00016 namespace L4 {
00017
00018 class L4_EXPORT Scheduler :
00019 public Kobject_t<Scheduler, Icu, L4_PROTO_SCHEDULER,
00020 Type_info::Demand_t<1> >
00021 {
00022 public:
00023 // ABI function for 'info' call
00024 L4_INLINE_RPC_NF_OP(L4_SCHEDULER_INFO_OP,
00025 l4_msgtag_t, info, (l4_umword_t gran_offset, l4_umword_t *map,
00026 l4_umword_t *cpu_max, l4_umword_t *sched_classes));
00027
00028 l4_msgtag_t info(l4_umword_t *cpu_max, l4_sched_cpu_set_t *cpus,
00029 l4_umword_t *sched_classes = nullptr,
00030 l4_utcb_t *utcb = l4_utcb()) const noexcept
00031 {
00032 l4_umword_t max = 0;
00033 l4_umword_t sc = 0;
00034 l4_msgtag_t t =
00035 info_t::call(c(), cpus->gran_offset, &cpus->map, &max, &sc, utcb);
00036 if (cpu_max)
00037 *cpu_max = max;
00038 if (sched_classes)
00039 *sched_classes = sc;
00040 return t;
00041 }
00042
00043 L4_INLINE_RPC_OP(L4_SCHEDULER_RUN_THREAD_OP,
00044 l4_msgtag_t, run_thread, (Ipc::Cap<Thread> thread, l4_sched_param_t const &sp));
00045
00046 L4_INLINE_RPC_OP(L4_SCHEDULER_IDLE_TIME_OP,
00047 l4_msgtag_t, idle_time, (l4_sched_cpu_set_t const &cpus,
00048 l4_kernel_clock_t *us));
00049
00050 bool is_online(l4_umword_t cpu, l4_utcb_t *utcb = l4_utcb()) const noexcept
00051 { return l4_scheduler_is_online_u(cap(), cpu, utcb); }
00052
00053 typedef L4::Typeid::Rpcsys<info_t, run_thread_t, idle_time_t> Rpcsys;
00054 };
00055
00056 }
```

## 17.577 l4/sys/semaphore File Reference

Semaphore class definition.

```

#include <l4/sys/irq>
#include <l4/sys/semaphore.h>
```



```

graph BT
 l4_sys_semaphore[l4/sys/semaphore]
 l4_l4re_vfs_impl_vcon_stream_h[l4/l4re_vfs/impl/vcon_stream.h]
 l4_l4virtio_client_l4virtio[l4/l4virtio/client/l4virtio]
 l4_l4re_vfs_impl_vcon_stream_impl_h[l4/l4re_vfs/impl/vcon_stream_impl.h]
 l4_l4re_vfs_impl_vfs_impl_h[l4/l4re_vfs/impl/vfs_impl.h]
 l4_l4virtio_client_virtio_block[l4/l4virtio/client/virtio-block]
 l4_l4virtio_client_virtio_net[l4/l4virtio/client/virtio-net]

 l4_l4re_vfs_impl_vcon_stream_impl_h --> l4_l4re_vfs_impl_vcon_stream_h
 l4_l4re_vfs_impl_vfs_impl_h --> l4_l4re_vfs_impl_vcon_stream_h
 l4_l4re_vfs_impl_vfs_impl_h --> l4_sys_semaphore
 l4_l4virtio_client_virtio_block --> l4_l4virtio_client_l4virtio
 l4_l4virtio_client_virtio_net --> l4_l4virtio_client_l4virtio
 l4_l4virtio_client_l4virtio --> l4_sys_semaphore
 l4_l4re_vfs_impl_vcon_stream_h --> l4_sys_semaphore

```

- struct [L4::Semaphore](#)  
C++ Kernel-provided semaphore interface, see [Kernel-provided semaphore](#) for the C interface.

- namespace **L4**  
*L4 low-level kernel interface.*

## 17.577.1 Detailed Description

Semaphore class definition.

Definition in file [semaphore](#).

## 17.578 semaphore

[Go to the documentation of this file.](#)

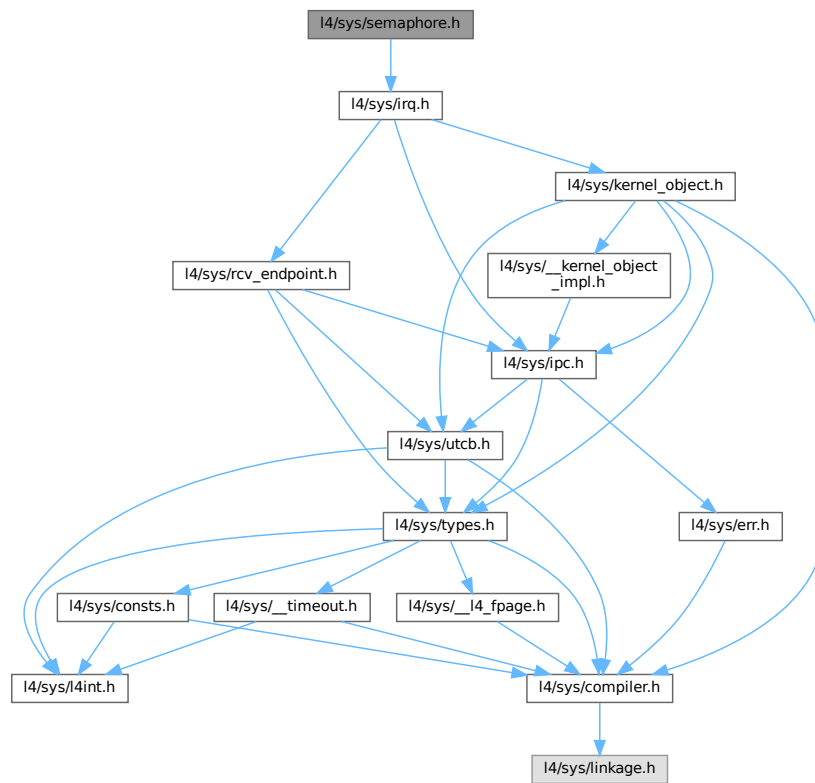
```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2015 Alexander Warg <alexander.warg@kernkonzept.com>
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011
00012 #pragma once
00013
00014 #include <l4/sys/irq>
00015 #include <l4/sys/semaphore.h>
00016
00017 namespace L4 {
00018
00051 struct Semaphore : Kobject_t<Semaphore, Triggerable, L4_PROTO_SEMAPHORE>
00052 {
00067 l4_msgtag_t up(l4_utcb_t *utcb = l4_utcb()) noexcept
00068 { return trigger(utcb); }
00069
00089 l4_msgtag_t down(l4_timeout_t timeout = L4_IPC_NEVER,
00090 l4_utcb_t *utcb = l4_utcb()) noexcept
00091 { return l4_semaphore_down_u(cap(), timeout, utcb); }
00092 };
00093
00094 }
```

## 17.579 l4/sys/semaphore.h File Reference

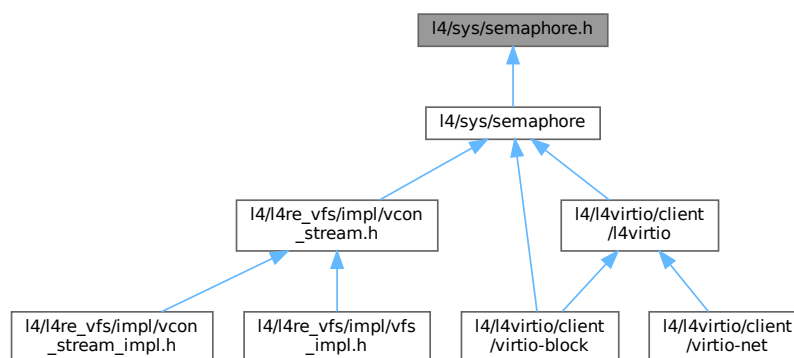
C semaphore interface.

```
#include <l4/sys/irq.h>
```

Include dependency graph for semaphore.h:



This graph shows which files directly or indirectly include this file:



## Functions

- [l4\\_msgtag\\_t l4\\_semaphore\\_up\(l4\\_cap\\_idx\\_t sem\)](#) [L4\\_NOTHROW](#)  
Semaphore up operation (wrapper for trigger()).
- [l4\\_msgtag\\_t l4\\_semaphore\\_down\(l4\\_cap\\_idx\\_t sem, l4\\_timeout\\_t timeout\)](#) [L4\\_NOTHROW](#)  
Semaphore down operation.

## 17.579.1 Detailed Description

C semaphore interface.

Definition in file [semaphore.h](#).

## 17.580 semaphore.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2015 Alexander Warg <alexander.warg@kernkonzept.com>
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011
00012 #pragma once
00013
00014 #include <l4/sys/irq.h>
00015
00024
00025 enum L4_semaphore_op
00026 {
00027 L4_SEMAPHORE_OP_DOWN = 0,
00028 // semaphore up is IRQ_OP_TRIGGER with IRQ/Triggerable protocol
00029 };
00030
00044 L4_INLINE l4_msgtag_t
00045 l4_semaphore_up(l4_cap_idx_t sem) L4_NOTHROW
00046 {
00047 return l4_irq_trigger(sem);
00048 }
00049
00053 L4_INLINE l4_msgtag_t
00054 l4_semaphore_up_u(l4_cap_idx_t sem, l4_utcb_t *utcb) L4_NOTHROW
00055 {
00056 return l4_irq_trigger_u(sem, utcb);
00057 }
00058
00078 L4_INLINE l4_msgtag_t
00079 l4_semaphore_down(l4_cap_idx_t sem, l4_timeout_t timeout) L4_NOTHROW;
00080
00084 L4_INLINE l4_msgtag_t
00085 l4_semaphore_down_u(l4_cap_idx_t sem, l4_timeout_t to,
00086 l4_utcb_t *utcb) L4_NOTHROW;
00087
00088
00089 L4_INLINE l4_msgtag_t
00090 l4_semaphore_down_u(l4_cap_idx_t sem, l4_timeout_t to,
00091 l4_utcb_t *utcb) L4_NOTHROW
00092 {
00093 l4_msg_regs_t *m = l4_utcb_mr_u(utcb);
00094 m->mr[0] = L4_SEMAPHORE_OP_DOWN;
00095 return l4_ipc_call(sem, utcb, l4_msgtag(L4_PROTO_SEMAPHORE, 1, 0, 0), to);
00096 }
00097
00098
00099 L4_INLINE l4_msgtag_t
00100 l4_semaphore_down(l4_cap_idx_t sem, l4_timeout_t to) L4_NOTHROW
00101 {
00102 return l4_semaphore_down_u(sem, to, l4_utcb());
00103 }
00104

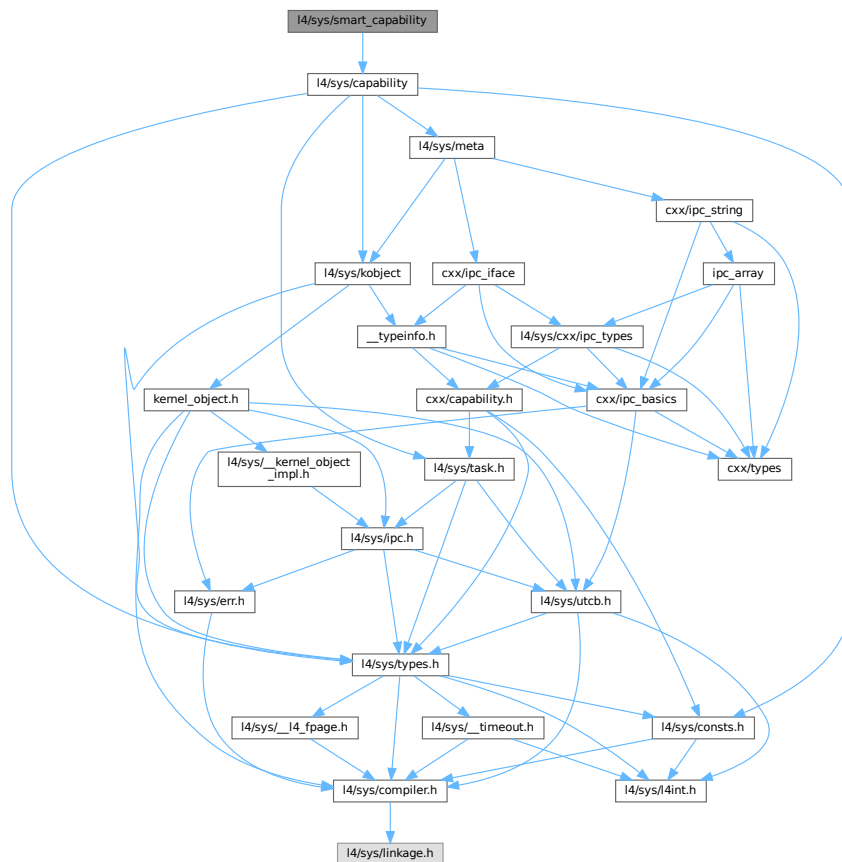
```

## 17.581 l4/sys/smart\_capability File Reference

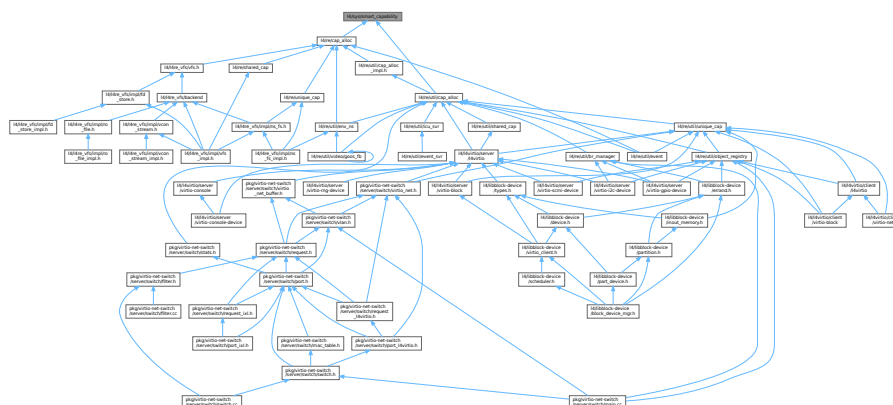
L4::Capability class.

```
#include <l4/sys/capability>
```

Include dependency graph for smart\_capability:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class [L4::Smart\\_cap< T, SMART >](#)  
*Smart capability class.*

## Namespaces

- namespace [L4](#)  
*[L4](#) low-level kernel interface.*

## Functions

- `template<typename T, typename F, typename SMART>`  
`Smart_cap< T, SMART > L4::cap\_cast (Smart_cap< F, SMART > const &c) noexcept`  
*static\_cast for (smart) capabilities.*
- `template<typename T, typename F, typename SMART>`  
`Smart_cap< T, SMART > L4::cap\_reinterpret\_cast (Smart_cap< F, SMART > const &c) noexcept`  
*reinterpret\_cast for (smart) capabilities.*

## 17.581.1 Detailed Description

L4::Capability class.

### Author

Alexander Warg [alexander.warg@os.inf.tu-dresden.de](mailto:alexander.warg@os.inf.tu-dresden.de)

Definition in file [smart\\_capability](#).

## 17.582 smart\_capability

[Go to the documentation of this file.](#)

```
00001 // vim:set ft=cpp: -*- Mode: C++ -*-
00009 /*
00010 * (c) 2008-2009 Author(s)
00011 * economic rights: Technische Universität Dresden (Germany)
00012 *
00013 * License: see LICENSE.spdx (in this directory or the directories above)
00014 */
00015 #pragma once
00016
00017 #include <l4/sys/capability>
00018
00019 namespace L4 {
00020
00021 template< typename T, typename SMART >
00022 class Smart_cap : public Cap_base, private SMART
00023 {
00024 public:
00025 SMART const &smart() const noexcept { return *this; }
00026
00027 void _delete() noexcept
00028 {
00029 SMART::free(const_cast<Smart_cap<T, SMART>&>(*this));
00030 }
00031
00032 Cap<T> release() const noexcept
00033 {
00034 l4_cap_idx_t r = cap();
00035 SMART::invalidate(const_cast<Smart_cap<T, SMART>&>(*this));
00036
00037 return Cap<T>(r);
00038 }
00039
00040 void reset() noexcept
00041 {
00042 _c = L4_INVALID_CAP;
00043 }
00044 }
```

```

00047 }
00048
00049 Smart_cap() noexcept : Cap_base(Invalid) {}
00050
00051 Smart_cap(Cap_base::Cap_type t) noexcept : Cap_base(t) {}
00052
00061 template< typename O >
00062 Smart_cap(Cap<O> const &p) noexcept : Cap_base(p.cap())
00063 { Cap<T>::template check_convertible_from<O>(); }
00064
00065 template< typename O >
00066 Smart_cap(Cap<O> const &p, SMART const &smart) noexcept
00067 : Cap_base(p.cap()), SMART(smart)
00068 { Cap<T>::template check_convertible_from<O>(); }
00069
00070 template< typename O >
00071 Smart_cap(Smart_cap<O, SMART> const &o) noexcept
00072 : Cap_base(SMART::copy(o)), SMART(o.smart())
00073 { Cap<T>::template check_convertible_from<O>(); }
00074
00075 Smart_cap(Smart_cap const &o) noexcept
00076 : Cap_base(SMART::copy(o)), SMART(o.smart())
00077 { }
00078
00079 template< typename O >
00080 Smart_cap(typename Cap<O>::Cap_type cap) noexcept : Cap_base(cap)
00081 { Cap<T>::template check_convertible_from<O>(); }
00082
00083 void operator = (typename Cap<T>::Cap_type cap) noexcept
00084 {
00085 _delete();
00086 _c = cap;
00087 }
00088
00089 template< typename O >
00090 void operator = (Smart_cap<O, SMART> const &o) noexcept
00091 {
00092 _delete();
00093 _c = this->SMART::copy(o).cap();
00094 this->SMART::operator = (o.smart());
00095 // return *this;
00096 }
00097
00098 Smart_cap const &operator = (Smart_cap const &o) noexcept
00099 {
00100 if (&o == this)
00101 return *this;
00102
00103 _delete();
00104 _c = this->SMART::copy(o).cap();
00105 this->SMART::operator = (o.smart());
00106 return *this;
00107 }
00108
00109 #if __cplusplus >= 201103L
00110 template< typename O >
00111 Smart_cap(Smart_cap<O, SMART> &&o) noexcept
00112 : Cap_base(o.release()), SMART(o.smart())
00113 { Cap<T>::template check_convertible_from<O>(); }
00114
00115 Smart_cap(Smart_cap &&o) noexcept
00116 : Cap_base(o.release()), SMART(o.smart())
00117 { }
00118
00119 template< typename O >
00120 void operator = (Smart_cap<O, SMART> &&o) noexcept
00121 {
00122 _delete();
00123 _c = o.release().cap();
00124 this->SMART::operator = (o.smart());
00125 // return *this;
00126 }
00127
00128 Smart_cap const &operator = (Smart_cap &&o) noexcept
00129 {
00130 if (&o == this)
00131 return *this;
00132
00133 _delete();
00134 _c = o.release().cap();
00135 this->SMART::operator = (o.smart());
00136 return *this;
00137 }
00138 #endif
00139
00143 Cap<T> operator -> () const noexcept { return Cap<T>(_c); }
00144

```

```

00145 Cap<T> get() const noexcept { return Cap<T>(_c); }
00146
00147 ~Smart_cap() noexcept { _delete(); }
00148 };
00149
00150 template< typename T >
00151 class Weak_cap : public Cap_base
00152 {
00153 public:
00154 Weak_cap() noexcept : Cap_base(Invalid) {}
00155
00156 template< typename O >
00157 Weak_cap(typename Cap<O>::Cap_type t) noexcept : Cap_base(t)
00158 { Cap<T>::template check_convertible_from<O>(); }
00159
00160 template< typename O, typename S >
00161 Weak_cap(Smart_cap<O, S> const &c) noexcept : Cap_base(c.cap())
00162 { Cap<T>::template check_convertible_from<O>(); }
00163
00164 Weak_cap(Weak_cap const &o) noexcept : Cap_base(o) {}
00165
00166 template< typename O >
00167 Weak_cap(Weak_cap<O> const &o) noexcept : Cap_base(o)
00168 { Cap<T>::template check_convertible_from<O>(); }
00169 };
00170
00171 namespace Cap_traits {
00172 template< typename T1, typename T2 >
00173 struct Type { enum { Equal = false }; };
00174
00175 template< typename T1 >
00176 struct Type<T1, T1> { enum { Equal = true }; };
00177 };
00178
00179 template< typename T, typename F, typename SMART >
00180 inline
00181 Smart_cap<T, SMART> cap_cast(Smart_cap<F, SMART> const &c) noexcept
00182 {
00183 Cap<T>::template check_castable_from<F>();
00184 return Smart_cap<T, SMART>(Cap<T>(SMART::copy(c).cap()));
00185 }
00186
00187 template< typename T, typename F, typename SMART >
00188 inline
00189 Smart_cap<T, SMART> cap_reinterpret_cast(Smart_cap<F, SMART> const &c) noexcept
00190 {
00191 return Smart_cap<T, SMART>(Cap<T>(SMART::copy(c).cap()));
00192 }
00193
00194 template< typename T, typename F, typename SMART >
00195 inline
00196 Smart_cap<T, SMART> cap_reinterpret_cast(Smart_cap<F, SMART> const &c) noexcept
00197 {
00198 return Smart_cap<T, SMART>(Cap<T>(SMART::copy(c).cap()));
00199 }
00200
00201 template< typename T, typename F, typename SMART >
00202 inline
00203 Smart_cap<T, SMART> cap_reinterpret_cast(Smart_cap<F, SMART> const &c) noexcept
00204 {
00205 return Smart_cap<T, SMART>(Cap<T>(SMART::copy(c).cap()));
00206 }
00207 }

```

## 17.583 l4/sys/snd\_destination File Reference

The C++ Sender destination interface.

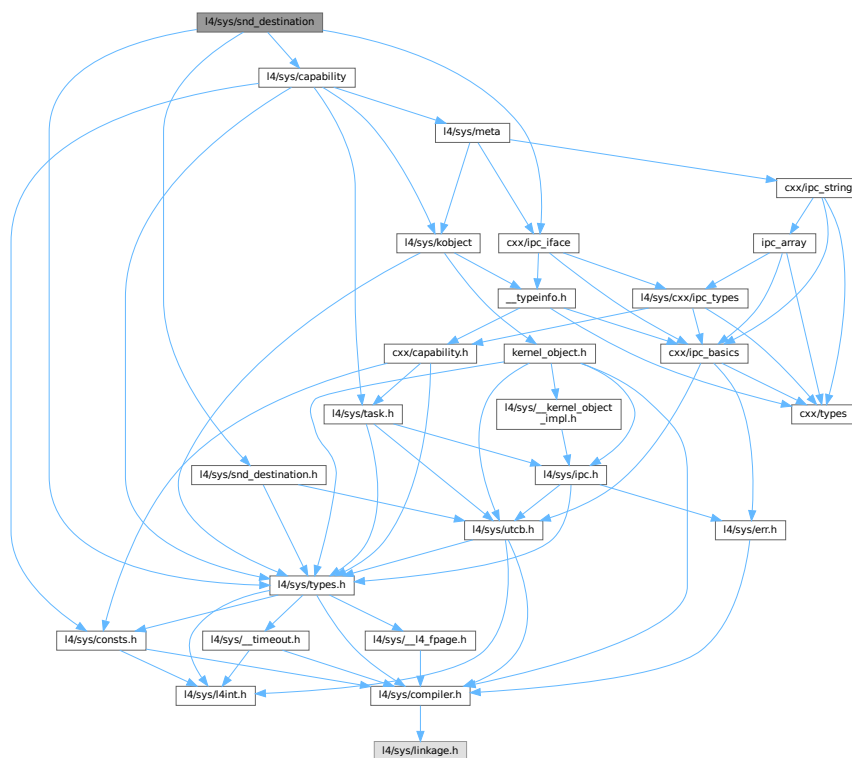
```

#include <l4/sys/snd_destination.h>
#include <l4/sys/types.h>
#include <l4/sys/capability>
#include <l4/sys/cxx/ipc_iface>

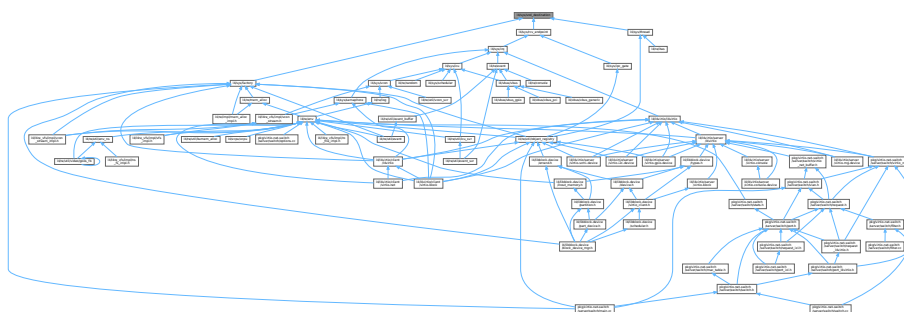
```



Include dependency graph for `snd_destination`:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [L4](#)  
*L4 low-level kernel interface.*

## 17.583.1 Detailed Description

The C++ Sender destination interface.

Definition in file [snd\\_destination](#).

## 17.584 snd\_destination

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2025 Frank Mehnert <frank.mehnert@kernkonzept.com>
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #pragma once
00012
00013 #include <l4/sys/snd_destination.h>
00014 #include <l4/sys/types.h>
00015 #include <l4/sys/capability>
00016 #include <l4/sys/cxx/ipc_iface>
00017
00018 namespace L4 {
00019
00020 class L4_EXPORT Snd_destination :
00021 public Kobject_t<Snd_destination, Kobject, L4_PROTO_KOBJECT>
00022 {
00023 };
00024
00025 }

```

## 17.585 l4/sys/snd\_destination.h File Reference

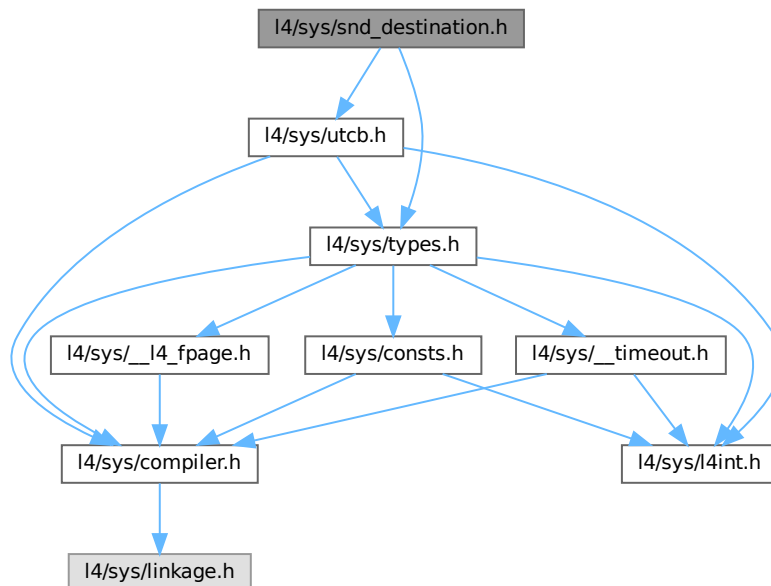
Sender destination endpoint C interface.

```

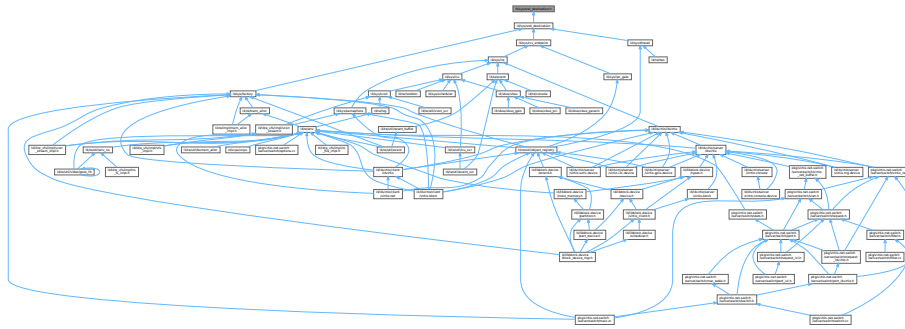
#include <l4/sys/utcb.h>
#include <l4/sys/types.h>

```

Include dependency graph for snd\_destination.h:



This graph shows which files directly or indirectly include this file:



## 17.585.1 Detailed Description

Sender destination endpoint C interface.

Definition in file [snd\\_destination.h](#).

## 17.586 snd\_destination.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2025 Frank Mehnert <frank.mehnert@kernkonzept.com>
00007 *
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010 #pragma once
00011
00012 #include <l4/sys/utcb.h>
00013 #include <l4/sys/types.h>

```

## 17.587 l4/sys/task File Reference

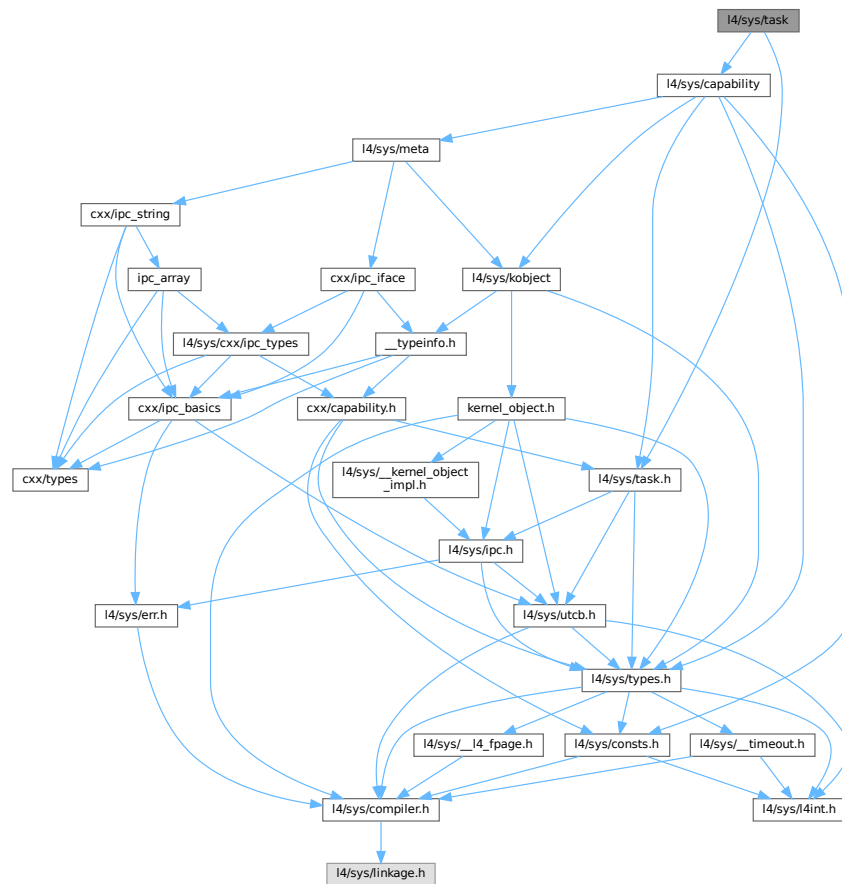
Common task related definitions.

```

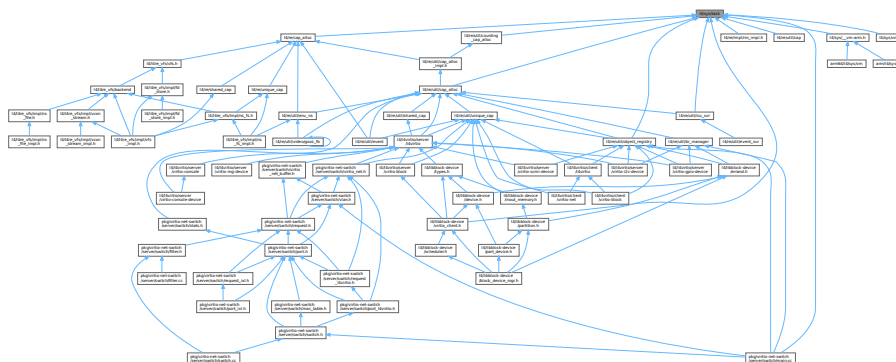
#include <l4/sys/task.h>
#include <l4/sys/capability>

```

Include dependency graph for task:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class [L4::Task](#)

*C++ interface of the [Task](#) kernel object, see [Task](#) for the C interface.*

## Namespaces

- namespace [L4](#)  
*[L4](#) low-level kernel interface.*

### 17.587.1 Detailed Description

Common task related definitions.

Definition in file [task](#).

## 17.588 task

[Go to the documentation of this file.](#)

```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009
00010 #pragma once
00011
00012 #include <l4/sys/task.h>
00013 #include <l4/sys/capability>
00014
00015 namespace L4 {
00016
00017 class Task :
00018 public Kobject<Task, Kobject, L4_PROTO_TASK,
00019 Type_info::Demand_t<2> >
00020 {
00021 public:
00022 l4_msgtag_t map(Cap<Task> const &src_task,
00023 l4_fpage_t const &snd_fpage, l4_umword_t snd_base,
00024 l4_utcb_t *utcb = l4_utcb()) noexcept
00025 { return l4_task_map_u(cap(), src_task.cap(), snd_fpage, snd_base, utcb); }
00026
00027 l4_msgtag_t unmap(l4_fpage_t const &fpage,
00028 l4_umword_t map_mask,
00029 l4_utcb_t *utcb = l4_utcb()) noexcept
00030 { return l4_task_unmap_u(cap(), fpage, map_mask, utcb); }
00031
00032 l4_msgtag_t unmap_batch(l4_fpage_t const *fpages,
00033 unsigned num_fpages,
00034 l4_umword_t map_mask,
00035 l4_utcb_t *utcb = l4_utcb()) noexcept
00036 { return l4_task_unmap_batch_u(cap(), fpages, num_fpages, map_mask, utcb); }
00037
00038 l4_msgtag_t delete_obj(L4::Cap<void> obj,
00039 l4_utcb_t *utcb = l4_utcb()) noexcept
00040 { return l4_task_delete_obj_u(cap(), obj.cap(), utcb); }
00041
00042 l4_msgtag_t release_cap(L4::Cap<void> cap,
00043 l4_utcb_t *utcb = l4_utcb()) noexcept
00044 { return l4_task_release_cap_u(this->cap(), cap.cap(), utcb); }
00045
00046 l4_msgtag_t cap_valid(Cap<void> const &cap,
00047 l4_utcb_t *utcb = l4_utcb()) noexcept
00048 { return l4_task_cap_valid_u(this->cap(), cap.cap(), utcb); }
00049
00050 l4_msgtag_t cap_equal(Cap<void> const &cap_a,
00051 Cap<void> const &cap_b,
00052 l4_utcb_t *utcb = l4_utcb()) noexcept
00053 { return l4_task_cap_equal_u(cap(), cap_a.cap(), cap_b.cap(), utcb); }
00054
00055 l4_msgtag_t add_ku_mem(l4_fpage_t *fpage,
00056 l4_utcb_t *utcb = l4_utcb()) noexcept
00057 { return l4_task_add_ku_mem_u(cap(), fpage, utcb); }
00058 };
00059 }
```

## 17.589 task.h

```

00001 /*
00002 * (c) 2018 Adam Lackorzynski <adam@l4re.org>
00003 *
00004 * License: see LICENSE.spdx (in this directory or the directories above)
00005 */
00006 #pragma once
00007
00008 #include_next <l4/sys/task.h>
00009 #include <l4/sys/__task-arm.h>

```

## 17.590 task.h

```

00001 /*
00002 * (c) 2018 Adam Lackorzynski <adam@l4re.org>
00003 *
00004 * License: see LICENSE.spdx (in this directory or the directories above)
00005 */
00006 #pragma once
00007
00008 #include_next <l4/sys/task.h>
00009 #include <l4/sys/__task-arm.h>

```

## 17.591 l4/sys/task.h File Reference

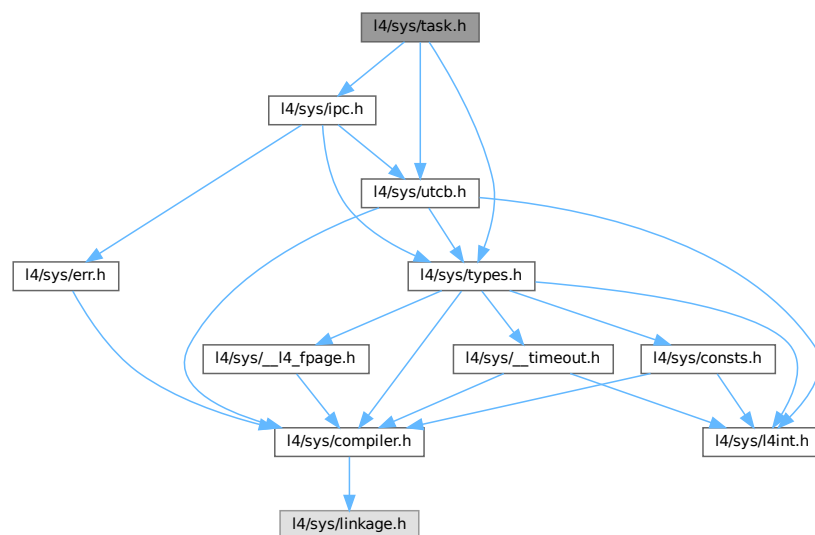
Common task related definitions.

```

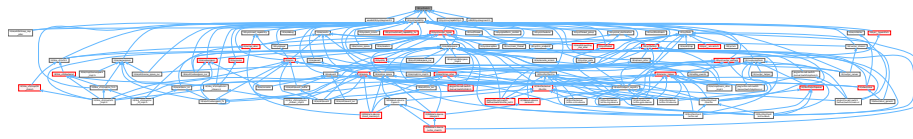
#include <l4/sys/types.h>
#include <l4/sys/utcb.h>
#include <l4/sys/ipc.h>

```

Include dependency graph for task.h:



This graph shows which files directly or indirectly include this file:



## Enumerations

- enum `L4_task_ops` {  
`L4_TASK_MAP_OP` = 0UL , `L4_TASK_UNMAP_OP` = 1UL , `L4_TASK_CAP_INFO_OP` = 2UL ,  
`L4_TASK_ADD_KU_MEM_OP` = 3UL ,  
`L4_TASK_LDT_SET_X86_OP` = 0x11UL , `L4_TASK_MAP_VGICC_ARM_OP` = 0x12UL }

*Operations on task objects.*

## Functions

- `l4_msgtag_t l4_task_map` (`l4_cap_idx_t` dst\_task, `l4_cap_idx_t` src\_task, `l4_fpage_t` snd\_fpage, `l4_umword_t` snd\_base) `L4_NOTHROW`  
*Map resources available in the source task to a destination task.*
- `l4_msgtag_t l4_task_unmap` (`l4_cap_idx_t` task, `l4_fpage_t` fpage, `l4_umword_t` map\_mask) `L4_NOTHROW`  
*Revoke rights from the task.*
- `l4_msgtag_t l4_task_unmap_batch` (`l4_cap_idx_t` task, `l4_fpage_t` const \*fpages, unsigned num\_fpages, `l4_umword_t` map\_mask) `L4_NOTHROW`  
*Revoke rights from a task.*
- `l4_msgtag_t l4_task_delete_obj` (`l4_cap_idx_t` task, `l4_cap_idx_t` obj) `L4_NOTHROW`  
*Release capability and delete object.*
- `l4_msgtag_t l4_task_release_cap` (`l4_cap_idx_t` task, `l4_cap_idx_t` cap) `L4_NOTHROW`  
*Release object capability.*
- `l4_msgtag_t l4_task_cap_valid` (`l4_cap_idx_t` task, `l4_cap_idx_t` cap) `L4_NOTHROW`  
*Check whether a capability is present (refers to an object).*
- `l4_msgtag_t l4_task_cap_equal` (`l4_cap_idx_t` task, `l4_cap_idx_t` cap\_a, `l4_cap_idx_t` cap\_b) `L4_NOTHROW`  
*Test whether two capabilities point to the same object with the same permissions (only considering selected permissions).*
- `l4_msgtag_t l4_task_add_ku_mem` (`l4_cap_idx_t` task, `l4_fpage_t` \*ku\_mem) `L4_NOTHROW`  
*Add kernel-user memory.*

### 17.591.1 Detailed Description

Common task related definitions.

Definition in file [task.h](#).

## 17.592 task.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00008 * Björn Döbel <doebel@os.inf.tu-dresden.de>,
00009 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #pragma once
00015 #include <l4/sys/types.h>
00016 #include <l4/sys/utcb.h>
00017
00031
00079 L4_INLINE l4_msgtag_t
00080 l4_task_map(l4_cap_idx_t dst_task, l4_cap_idx_t src_task,
00081 l4_fpage_t snd_fpage, l4_umword_t snd_base) L4_NOTHROW;
00082
00086 L4_INLINE l4_msgtag_t
00087 l4_task_map_u(l4_cap_idx_t dst_task, l4_cap_idx_t src_task,
00088 l4_fpage_t snd_fpage, l4_umword_t snd_base, l4_utcb_t *utcb) L4_NOTHROW;
00089
00134 L4_INLINE l4_msgtag_t
00135 l4_task_unmap(l4_cap_idx_t task, l4_fpage_t fpage,
00136 l4_umword_t map_mask) L4_NOTHROW;
00137
00141 L4_INLINE l4_msgtag_t
00142 l4_task_unmap_u(l4_cap_idx_t task, l4_fpage_t fpage,
00143 l4_umword_t map_mask, l4_utcb_t *utcb) L4_NOTHROW;
00144
00164 L4_INLINE l4_msgtag_t
00165 l4_task_unmap_batch(l4_cap_idx_t task, l4_fpage_t const *fpages,
00166 unsigned num_fpages, l4_umword_t map_mask) L4_NOTHROW;
00167
00171 L4_INLINE l4_msgtag_t
00172 l4_task_unmap_batch_u(l4_cap_idx_t task, l4_fpage_t const *fpages,
00173 unsigned num_fpages, l4_umword_t map_mask,
00174 l4_utcb_t *u) L4_NOTHROW;
00175
00197 L4_INLINE l4_msgtag_t
00198 l4_task_delete_obj(l4_cap_idx_t task, l4_cap_idx_t obj) L4_NOTHROW;
00199
00203 L4_INLINE l4_msgtag_t
00204 l4_task_delete_obj_u(l4_cap_idx_t task, l4_cap_idx_t obj,
00205 l4_utcb_t *u) L4_NOTHROW;
00206
00225 L4_INLINE l4_msgtag_t
00226 l4_task_release_cap(l4_cap_idx_t task, l4_cap_idx_t cap) L4_NOTHROW;
00227
00231 L4_INLINE l4_msgtag_t
00232 l4_task_release_cap_u(l4_cap_idx_t task, l4_cap_idx_t cap,
00233 l4_utcb_t *u) L4_NOTHROW;
00234
00235
00253 L4_INLINE l4_msgtag_t
00254 l4_task_cap_valid(l4_cap_idx_t task, l4_cap_idx_t cap) L4_NOTHROW;
00255
00259 L4_INLINE l4_msgtag_t
00260 l4_task_cap_valid_u(l4_cap_idx_t task, l4_cap_idx_t cap, l4_utcb_t *utcb) L4_NOTHROW;
00261
00289 L4_INLINE l4_msgtag_t
00290 l4_task_cap_equal(l4_cap_idx_t task, l4_cap_idx_t cap_a,
00291 l4_cap_idx_t cap_b) L4_NOTHROW;
00292
00296 L4_INLINE l4_msgtag_t
00297 l4_task_add_ku_mem_u(l4_cap_idx_t task, l4_fpage_t *ku_mem,
00298 l4_utcb_t *u) L4_NOTHROW;
00299
00326 L4_INLINE l4_msgtag_t
00327 l4_task_add_ku_mem(l4_cap_idx_t task, l4_fpage_t *ku_mem) L4_NOTHROW;
00328
00329
00333 L4_INLINE l4_msgtag_t
00334 l4_task_cap_equal_u(l4_cap_idx_t task, l4_cap_idx_t cap_a,
00335 l4_cap_idx_t cap_b, l4_utcb_t *utcb) L4_NOTHROW;
00336
00341 enum L4_task_ops
00342 {
00343 L4_TASK_MAP_OP = 0UL,
00344 L4_TASK_UNMAP_OP = 1UL,
00345 L4_TASK_CAP_INFO_OP = 2UL,

```



```

00346 L4_TASK_ADD_KU_MEM_OP = 3UL,
00347 L4_TASK_LDT_SET_X86_OP = 0x11UL,
00348 L4_TASK_MAP_VGICC_ARM_OP = 0x12UL,
00349 };
00350
00351
00352 /* IMPLEMENTATION ----- */
00353
00354 #include <l4/sys/ipc.h>
00355
00356
00357 L4_INLINE l4_msgtag_t
00358 l4_task_map_u(l4_cap_idx_t dst_task, l4_cap_idx_t src_task,
00359 l4_fpage_t snd_fpage, l4_umword_t snd_base, l4_utcb_t *u) L4_NOTHROW
00360 {
00361 l4_msg_regs_t *v = l4_utcb_mr_u(u);
00362 v->mr[0] = L4_TASK_MAP_OP;
00363 v->mr[3] = l4_map_obj_control(0,0);
00364 v->mr[4] = l4_obj_fpage(src_task, 0, L4_CAP_FPAGE_RWS).raw;
00365 v->mr[1] = snd_base;
00366 v->mr[2] = snd_fpage.raw;
00367 return l4_ipc_call(dst_task, u, l4_msgtag(L4_PROTO_TASK, 3, 1, 0), L4_IPC_NEVER);
00368 }
00369
00370 L4_INLINE l4_msgtag_t
00371 l4_task_unmap_u(l4_cap_idx_t task, l4_fpage_t fpage,
00372 l4_umword_t map_mask, l4_utcb_t *u) L4_NOTHROW
00373 {
00374 l4_msg_regs_t *v = l4_utcb_mr_u(u);
00375 v->mr[0] = L4_TASK_UNMAP_OP;
00376 v->mr[1] = map_mask;
00377 v->mr[2] = fpage.raw;
00378 return l4_ipc_call(task, u, l4_msgtag(L4_PROTO_TASK, 3, 0, 0), L4_IPC_NEVER);
00379 }
00380
00381 L4_INLINE l4_msgtag_t
00382 l4_task_unmap_batch_u(l4_cap_idx_t task, l4_fpage_t const *fpages,
00383 unsigned num_fpages, l4_umword_t map_mask,
00384 l4_utcb_t *u) L4_NOTHROW
00385 {
00386 l4_msg_regs_t *v = l4_utcb_mr_u(u);
00387 v->mr[0] = L4_TASK_UNMAP_OP;
00388 v->mr[1] = map_mask;
00389 __builtin_memcpy(&v->mr[2], fpages, num_fpages * sizeof(l4_fpage_t));
00390 return l4_ipc_call(task, u, l4_msgtag(L4_PROTO_TASK, 2 + num_fpages, 0, 0), L4_IPC_NEVER);
00391 }
00392
00393 L4_INLINE l4_msgtag_t
00394 l4_task_cap_valid_u(l4_cap_idx_t task, l4_cap_idx_t cap, l4_utcb_t *u) L4_NOTHROW
00395 {
00396 l4_msg_regs_t *v = l4_utcb_mr_u(u);
00397 v->mr[0] = L4_TASK_CAP_INFO_OP;
00398 v->mr[1] = cap & ~1UL;
00399 return l4_ipc_call(task, u, l4_msgtag(L4_PROTO_TASK, 2, 0, 0), L4_IPC_NEVER);
00400 }
00401
00402 L4_INLINE l4_msgtag_t
00403 l4_task_cap_equal_u(l4_cap_idx_t task, l4_cap_idx_t cap_a,
00404 l4_cap_idx_t cap_b, l4_utcb_t *u) L4_NOTHROW
00405 {
00406 l4_msg_regs_t *v = l4_utcb_mr_u(u);
00407 v->mr[0] = L4_TASK_CAP_INFO_OP;
00408 v->mr[1] = cap_a;
00409 v->mr[2] = cap_b;
00410 return l4_ipc_call(task, u, l4_msgtag(L4_PROTO_TASK, 3, 0, 0), L4_IPC_NEVER);
00411 }
00412
00413 L4_INLINE l4_msgtag_t
00414 l4_task_add_ku_mem_u(l4_cap_idx_t task, l4_fpage_t *ku_mem,
00415 l4_utcb_t *u) L4_NOTHROW
00416 {
00417 l4_msg_regs_t *v = l4_utcb_mr_u(u);
00418 l4_msgtag_t ret;
00419 v->mr[0] = L4_TASK_ADD_KU_MEM_OP;
00420 v->mr[1] = ku_mem->raw;
00421 ret = l4_ipc_call(task, u, l4_msgtag(L4_PROTO_TASK, 2, 0, 0), L4_IPC_NEVER);
00422 if (!l4_msgtag_has_error(ret))
00423 {
00424 l4_msg_regs_t *v = l4_utcb_mr_u(u);
00425 ku_mem->raw = v->mr[0];
00426 }
00427 return ret;
00428 }
00429
00430
00431
00432 L4_INLINE l4_msgtag_t

```

```

00433 l4_task_map(l4_cap_idx_t dst_task, l4_cap_idx_t src_task,
00434 l4_fpage_t snd_fpage, l4_umword_t snd_base) L4_NOTHROW
00435 {
00436 return l4_task_map_u(dst_task, src_task, snd_fpage, snd_base, l4_utcb());
00437 }
00438
00439 L4_INLINE l4_msgtag_t
00440 l4_task_unmap(l4_cap_idx_t task, l4_fpage_t fpage,
00441 l4_umword_t map_mask) L4_NOTHROW
00442 {
00443 return l4_task_unmap_u(task, fpage, map_mask, l4_utcb());
00444 }
00445
00446 L4_INLINE l4_msgtag_t
00447 l4_task_unmap_batch(l4_cap_idx_t task, l4_fpage_t const *fpages,
00448 unsigned num_fpages, l4_umword_t map_mask) L4_NOTHROW
00449 {
00450 return l4_task_unmap_batch_u(task, fpages, num_fpages, map_mask,
00451 l4_utcb());
00452 }
00453
00454 L4_INLINE l4_msgtag_t
00455 l4_task_delete_obj_u(l4_cap_idx_t task, l4_cap_idx_t obj,
00456 l4_utcb_t *u) L4_NOTHROW
00457 {
00458 return l4_task_unmap_u(task, l4_obj_fpage(obj, 0, L4_CAP_FPAGE_RWSD),
00459 L4_FP_DELETE_OBJ, u);
00460 }
00461
00462 L4_INLINE l4_msgtag_t
00463 l4_task_delete_obj(l4_cap_idx_t task, l4_cap_idx_t obj) L4_NOTHROW
00464 {
00465 return l4_task_delete_obj_u(task, obj, l4_utcb());
00466 }
00467
00468 L4_INLINE l4_msgtag_t
00469 l4_task_release_cap_u(l4_cap_idx_t task, l4_cap_idx_t cap,
00470 l4_utcb_t *u) L4_NOTHROW
00471 {
00472 return l4_task_unmap_u(task, l4_obj_fpage(cap, 0, L4_CAP_FPAGE_RWSD),
00473 L4_FP_ALL_SPACES, u);
00474 }
00475
00476 L4_INLINE l4_msgtag_t
00477 l4_task_release_cap(l4_cap_idx_t task, l4_cap_idx_t cap) L4_NOTHROW
00478 {
00479 return l4_task_release_cap_u(task, cap, l4_utcb());
00480 }
00481
00482 L4_INLINE l4_msgtag_t
00483 l4_task_cap_valid(l4_cap_idx_t task, l4_cap_idx_t cap) L4_NOTHROW
00484 {
00485 return l4_task_cap_valid_u(task, cap, l4_utcb());
00486 }
00487
00488 L4_INLINE l4_msgtag_t
00489 l4_task_cap_equal(l4_cap_idx_t task, l4_cap_idx_t cap_a,
00490 l4_cap_idx_t cap_b) L4_NOTHROW
00491 {
00492 return l4_task_cap_equal_u(task, cap_a, cap_b, l4_utcb());
00493 }
00494
00495 L4_INLINE l4_msgtag_t
00496 l4_task_add_ku_mem(l4_cap_idx_t task, l4_fpage_t *ku_mem) L4_NOTHROW
00497 {
00498 return l4_task_add_ku_mem_u(task, ku_mem, l4_utcb());
00499 }
00500

```

## 17.593 l4/cxx/thread File Reference

Thread implementation.

```

#include <l4/sys/capability>
#include <l4/sys/types.h>

```

```

graph BT
 A[l4/cxx/thread] --> B[l4/cxx/main_thread]

```

- namespace `cxx`  
*Our C++ library.*

Thread implementation.  
Definition in file [thread](#).

## 17.594 thread

[Go to the documentation of this file.](#)

```

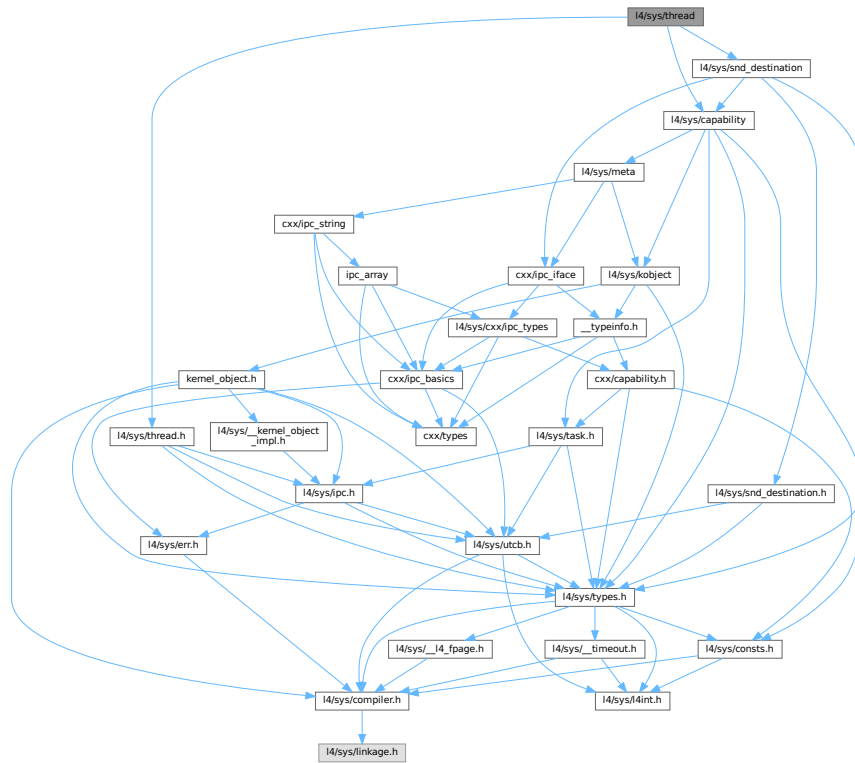
00001 // vim:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2004-2009 Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 * This file is part of TUD:OS and distributed under the terms of the
00006 * GNU Lesser General Public License 2.1.
00007 * Please see the COPYING-LGPL-2.1 file for details.
00008 */
00009
00010 #ifndef CXX_THREAD_H__
00011 #define CXX_THREAD_H__
00012
00013 #include <l4/sys/capability>
00014 #include <l4/sys/types.h>
00015
00016 namespace cxx {
00017
00018 class Thread
00019 {
00020 public:
00021
00022 enum State
00023 {
00024 Dead = 0,
00025 Running = 1,
00026 Stopped = 2,
00027 };
00028
00029 Thread(bool initiate);
00030 Thread(void *stack);
00031 Thread(void *stack, L4::Cap<L4::Thread> const &cap);
00032 virtual ~Thread();
00033 void execute() asm ("L4_Thread_execute");
00034 virtual void run() = 0;
00035 virtual void shutdown() asm ("L4_Thread_shutdown");
00036 void start();
00037 void stop();
00038
00039 L4::Cap<L4::Thread> self() const throw()
00040 { return _cap; }
00041
00042 State state() const
00043 { return _state; }
00044
00045 static void start_cxx_thread(Thread *_this)
00046 asm ("L4_Thread_start_cxx_thread");
00047
00048 static void kill_cxx_thread(Thread *_this)
00049 asm ("L4_Thread_kill_cxx_thread");
00050
00051 static void set_pager(L4::Cap<void>const &p) throw()
00052 { _pager = p; }
00053
00054 private:
00055 int create();
00056
00057 L4::Cap<L4::Thread> _cap;
00058 State _state;
00059
00060 protected:
00061 void *_stack;
00062
00063 private:
00064 static L4::Cap<void> _pager;
00065 static L4::Cap<void> _master;
00066 };
00067
00068 };
00069
00070 #endif /* CXX_THREAD_H__ */
00071

```

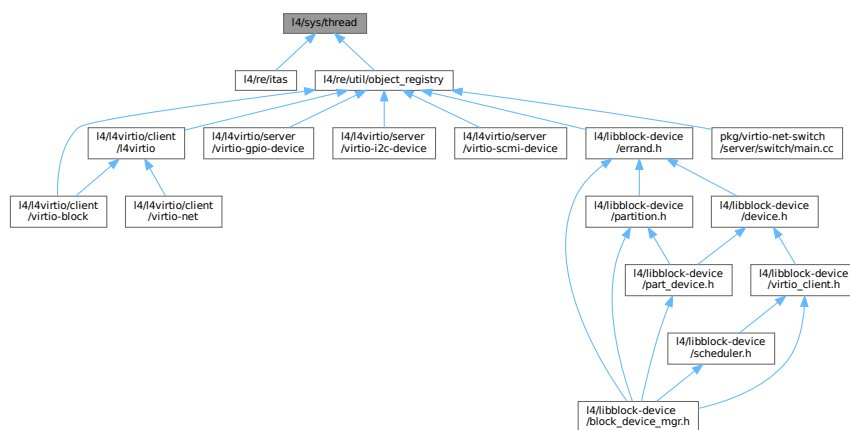
## 17.595 l4/sys/thread File Reference

Common thread related definitions.

Include dependency graph for thread:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class L4::Thread

C++ [L4](#) kernel thread interface, see [Thread](#) for the C interface.

- class [L4::Thread::Attr](#)

[Thread](#) attributes used for [control\(\)](#).

- class [L4::Thread::Modify\\_senders](#)

[Class](#) wrapping a list of rules which modify the sender label of IPC messages inbound to this thread.

## Namespaces

- namespace [L4](#)

[L4](#) low-level kernel interface.

## 17.595.1 Detailed Description

Common thread related definitions.

Definition in file [thread](#).

## 17.596 thread

[Go to the documentation of this file.](#)

```

00001 // vi:set ft=c++: -- Mode: C++ --
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013
00014 #pragma once
00015
00016 #include <l4/sys/capability>
00017 #include <l4/sys/snd_destination>
00018 #include <l4/sys/thread.h>
00019
00020 namespace L4 {
00021
00022 class Thread :
00023 public Kobject_t<Thread, Snd_destination, L4_PROTO_THREAD,
00024 Type_info::Demand_t<1> >
00025 {
00026 public:
00027 l4_msgtag_t ex_regs(l4_addr_t ip, l4_addr_t sp,
00028 l4_umword_t flags,
00029 l4_utcb_t *utcb = l4_utcb()) noexcept
00030 { return l4_thread_ex_regs_u(cap(), ip, sp, flags, utcb); }
00031
00032 l4_msgtag_t ex_regs(l4_addr_t *ip, l4_addr_t *sp,
00033 l4_umword_t *flags,
00034 l4_utcb_t *utcb = l4_utcb()) noexcept
00035 { return l4_thread_ex_regs_ret_u(cap(), ip, sp, flags, utcb); }
00036
00037 class Attr
00038 {
00039 private:
00040 friend class L4::Thread;
00041 l4_utcb_t *_u;
00042
00043 public:
00044 explicit Attr(l4_utcb_t *utcb = l4_utcb()) noexcept : _u(utcb)
00045 { l4_thread_control_start_u(utcb); }
00046
00047 void pager(Cap<void> const &pager) noexcept
00048 { l4_thread_control_pager_u(pager.cap(), _u); }
00049
00050 Cap<void> pager() noexcept

```

```

00171 { return Cap<void>(l4_utcb_mr_u(_u)->mr[1]); }
00172
00180 void exc_handler(Cap<void> const &exc_handler) noexcept
00181 { l4_thread_control_exc_handler_u(exc_handler.cap(), _u); }
00182
00189 Cap<void> exc_handler() noexcept
00190 { return Cap<void>(l4_utcb_mr_u(_u)->mr[2]); }
00191
00218 void bind(l4_utcb_t *thread_utcb, Cap<Task> const &task) noexcept
00219 { l4_thread_control_bind_u(thread_utcb, task.cap(), _u); }
00220
00224 void alien(int on) noexcept
00225 { l4_thread_control_alien_u(_u, on); }
00226 };
00227
00243 l4_msgtag_t control(Attr const &attr) noexcept
00244 { return l4_thread_control_commit_u(cap(), attr._u); }
00245
00253 l4_msgtag_t switch_to(l4_utcb_t *utcb = l4_utcb()) noexcept
00254 { return l4_thread_switch_u(cap(), utcb); }
00255
00264 l4_msgtag_t stats_time(l4_kernel_clock_t *us,
00265 l4_utcb_t *utcb = l4_utcb()) noexcept
00266 { return l4_thread_stats_time_u(cap(), us, utcb); }
00267
00283 l4_msgtag_t vcpu_resume_start(l4_utcb_t *utcb = l4_utcb()) noexcept
00284 { return l4_thread_vcpu_resume_start_u(utcb); }
00285
00334 l4_msgtag_t vcpu_resume_commit(l4_msgtag_t tag,
00335 l4_utcb_t *utcb = l4_utcb()) noexcept
00336 { return l4_thread_vcpu_resume_commit_u(cap(), tag, utcb); }
00337
00358 l4_msgtag_t vcpu_control(l4_addr_t vcpu_state, l4_utcb_t *utcb = l4_utcb())
00359 noexcept
00360 { return l4_thread_vcpu_control_u(cap(), vcpu_state, utcb); }
00361
00398 l4_msgtag_t vcpu_control_ext(l4_addr_t ext_vcpu_state,
00399 l4_utcb_t *utcb = l4_utcb()) noexcept
00400 { return l4_thread_vcpu_control_ext_u(cap(), ext_vcpu_state, utcb); }
00401
00427 l4_msgtag_t register_del_irq(Cap<Irq> irq, l4_utcb_t *u = l4_utcb()) noexcept
00428 { return l4_thread_register_del_irq_u(cap(), irq.cap(), u); }
00429
00448 class Modify_senders
00449 {
00450 private:
00451 friend class Thread;
00452 l4_utcb_t *utcb;
00453 unsigned cnt;
00454
00455 public:
00456 explicit Modify_senders(l4_utcb_t *u = l4_utcb()) noexcept
00457 : utcb(u), cnt(1)
00458 {
00459 l4_utcb_mr_u(utcb)->mr[0] = L4_THREAD_MODIFY_SENDER_OP;
00460 }
00461
00481 int add(l4_umword_t match_mask, l4_umword_t match,
00482 l4_umword_t del_bits, l4_umword_t add_bits) noexcept
00483 {
00484 l4_msg_regs_t *m = l4_utcb_mr_u(utcb);
00485 if (cnt >= L4_UTCB_GENERIC_DATA_SIZE - 4)
00486 return -L4_ENOMEM;
00487 m->mr[cnt++] = match_mask;
00488 m->mr[cnt++] = match;
00489 m->mr[cnt++] = del_bits;
00490 m->mr[cnt++] = add_bits;
00491 return 0;
00492 }
00493 };
00494
00525 l4_msgtag_t modify_senders(Modify_senders const &todo) noexcept
00526 {
00527 return l4_ipc_call(cap(), todo.utcb, l4_msgtag(L4_PROTO_THREAD, todo.cnt, 0, 0), L4_IPC_NEVER);
00528 }
00529
00553 l4_msgtag_t register_doorbell_irq(Cap<Irq> irq, l4_utcb_t *u = l4_utcb()) noexcept
00554 { return l4_thread_register_doorbell_irq_u(cap(), irq.cap(), u); }
00555 };
00556 }

```





```

00016
00017 namespace L4 {
00018
00034 class L4_EXPORT Thread_group :
00035 public Kobject_t<Thread_group, Snd_destination, L4_PROTO_THREAD_GROUP,
00036 Type_info::Demand_t<1>
00037 {
00038 public:
00053 l4_msgtag_t add(Cap<Thread> thread, l4_utcb_t *utcb = l4_utcb()) noexcept
00054 { return l4_thread_group_add_u(cap(), thread.cap(), utcb); }
00055
00067 l4_msgtag_t remove(Cap<Thread> thread, l4_utcb_t *utcb = l4_utcb()) noexcept
00068 { return l4_thread_group_remove_u(cap(), thread.cap(), utcb); }
00069 };
00070
00071 }

```

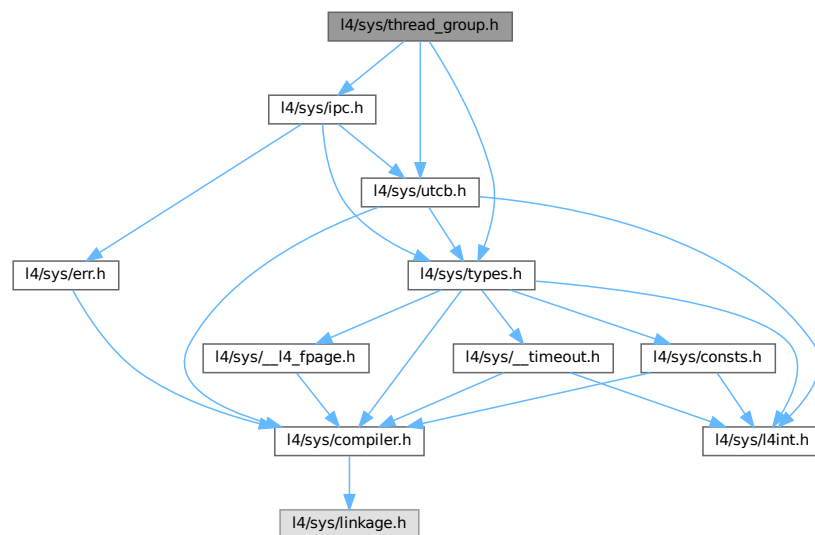
## 17.599 l4/sys/thread\_group.h File Reference

```
#include <l4/sys/types.h>
```

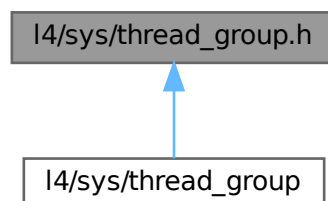
```
#include <l4/sys/utcb.h>
```

```
#include <l4/sys/ipc.h>
```

Include dependency graph for thread\_group.h:



This graph shows which files directly or indirectly include this file:



## Functions

- [l4\\_msgtag\\_t l4\\_thread\\_group\\_add \(l4\\_cap\\_idx\\_t tg, l4\\_cap\\_idx\\_t thread\) L4\\_NOTHROW](#)  
*Add thread to a thread group.*
- [l4\\_msgtag\\_t l4\\_thread\\_group\\_remove \(l4\\_cap\\_idx\\_t tg, l4\\_cap\\_idx\\_t thread\) L4\\_NOTHROW](#)  
*Remove thread from a thread group.*

## 17.600 thread\_group.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * Copyright (C) 2022-2025 Kernkonzept GmbH.
00003 * Author(s): Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00004 * Frank Mehnert <frank.mehnert@kernkonzept.com>
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00011 #pragma once
00012
00013 #include <l4/sys/types.h>
00014 #include <l4/sys/utcb.h>
00015
00033 enum l4_thread_group_ops
00034 {
00035 L4_THREAD_GROUP_ADD_OP = 2UL,
00036 L4_THREAD_GROUP_REMOVE_OP = 3UL,
00037 };
00039 enum l4_thread_group_policy
00040 {
00041 L4_THREAD_GROUP_POLICY_STRICT_CORE_LOCAL = 0,
00042 L4_THREAD_GROUP_POLICY_SOFT_CORE_LOCAL = 1,
00043 };
00045
00052 L4_INLINE l4_msgtag_t
00053 l4_thread_group_add(l4_cap_idx_t tg,
00054 l4_cap_idx_t thread) L4_NOTHROW;
00055
00059 L4_INLINE l4_msgtag_t
00060 l4_thread_group_add_u(l4_cap_idx_t tg,
00061 l4_cap_idx_t thread,
00062 l4_utcb_t *utcb) L4_NOTHROW;
00063
00070 L4_INLINE l4_msgtag_t
00071 l4_thread_group_remove(l4_cap_idx_t tg,
00072 l4_cap_idx_t thread) L4_NOTHROW;
00073
00077 L4_INLINE l4_msgtag_t
00078 l4_thread_group_remove_u(l4_cap_idx_t tg,
00079 l4_cap_idx_t thread,
00080 l4_utcb_t *utcb) L4_NOTHROW;
00081
00082 /* IMPLEMENTATION ----- */
00083
00084 #include <l4/sys/ipc.h>
00085
00087 L4_INLINE l4_msgtag_t
00088 l4_thread_group_add_u(l4_cap_idx_t tg,
00089 l4_cap_idx_t thread,
00090 l4_utcb_t *utcb) L4_NOTHROW
00091 {
00092 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00093 v->mr[0] = L4_THREAD_GROUP_ADD_OP;
00094 v->mr[1] = l4_map_obj_control(0, 0);
00095 v->mr[2] = l4_obj_fpage(thread, 0, L4_CAP_FPAGE_RWS).raw;
00096 return l4_ipc_call(tg, utcb,
00097 l4_msgtag(L4_PROTO_THREAD_GROUP, 1, 1, 0), L4_IPC_NEVER);
00098 }
00099
00100 L4_INLINE l4_msgtag_t
00101 l4_thread_group_remove_u(l4_cap_idx_t tg,
00102 l4_cap_idx_t thread,
00103 l4_utcb_t *utcb) L4_NOTHROW
00104 {
00105 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);

```

```

00106 v->mr[0] = L4_THREAD_GROUP_REMOVE_OP;
00107 v->mr[1] = l4_map_obj_control(0, 0);
00108 v->mr[2] = l4_obj_fpage(thread, 0, L4_CAP_FPAGE_RWS).raw;
00109 return l4_ipc_call(tg, utcb,
00110 l4_msgtag(L4_PROTO_THREAD_GROUP, 1, 1, 0), L4_IPC_NEVER);
00111 }
00112
00113 L4_INLINE l4_msgtag_t
00114 l4_thread_group_add(l4_cap_idx_t tg,
00115 l4_cap_idx_t thread) L4_NOTHROW
00116 {
00117 return l4_thread_group_add_u(tg, thread, l4_utcb());
00118 }
00119
00120 L4_INLINE l4_msgtag_t
00121 l4_thread_group_remove(l4_cap_idx_t tg,
00122 l4_cap_idx_t thread) L4_NOTHROW
00123 {
00124 return l4_thread_group_remove_u(tg, thread, l4_utcb());
00125 }

```

## 17.601 l4/sys/typeinfo\_svr File Reference

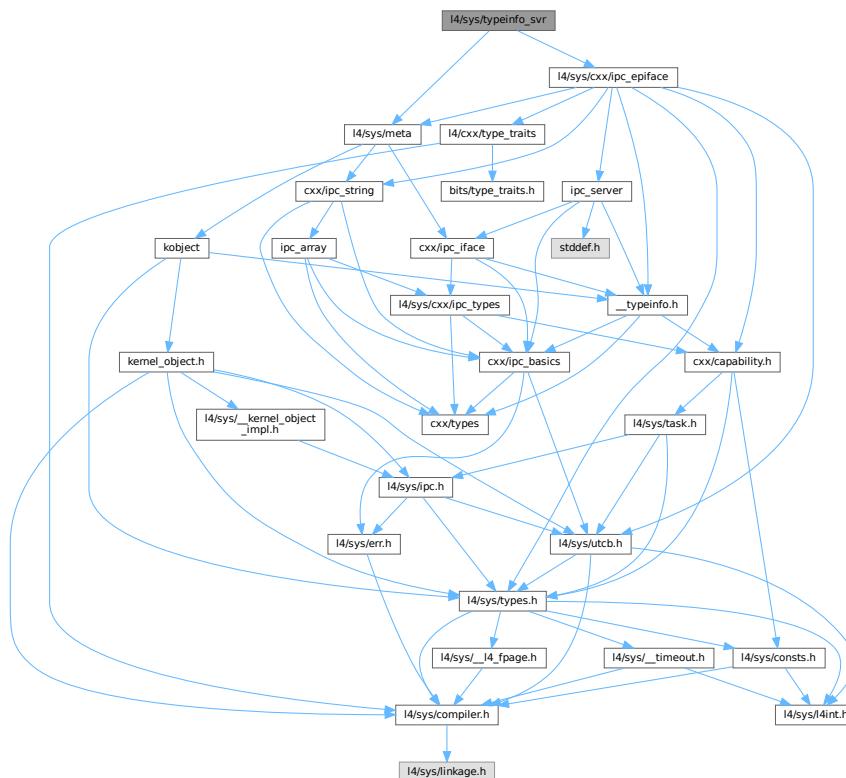
Type information server template.

```

#include <l4/sys/meta>
#include <l4/sys/cxx/ipc_epiface>

```

Include dependency graph for typeinfo\_svr:



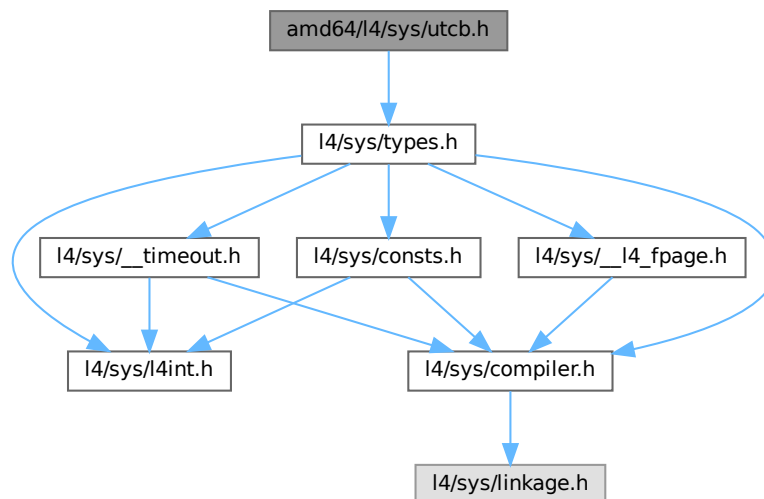


## 17.603 amd64/l4/sys/utcb.h File Reference

UTCB definitions for AMD64.

```
#include <l4/sys/types.h>
```

Include dependency graph for utcb.h:



### Data Structures

- struct [l4\\_exc\\_regs\\_t](#)  
*UTCB structure for exceptions.*

### Typedefs

- typedef struct `l4_exc_regs_t` [l4\\_exc\\_regs\\_t](#)  
*UTCB structure for exceptions.*

### Enumerations

- enum [L4\\_utcb\\_consts\\_amd64](#)  
*UTCB constants for AMD64.*

## Functions

- [l4\\_umword\\_t l4\\_utcb\\_exc\\_pc \(l4\\_exc\\_regs\\_t const \\*u\) L4\\_NOTHROW](#)  
*Access function to get the program counter of the exception state.*
- [void l4\\_utcb\\_exc\\_pc\\_set \(l4\\_exc\\_regs\\_t \\*u, l4\\_addr\\_t pc\) L4\\_NOTHROW](#)  
*Set the program counter register in the exception state.*
- [l4\\_umword\\_t l4\\_utcb\\_exc\\_typeval \(l4\\_exc\\_regs\\_t const \\*u\) L4\\_NOTHROW](#)  
*Get the value out of an exception UTCB that describes the type of exception.*
- [int l4\\_utcb\\_exc\\_is\\_pf \(l4\\_exc\\_regs\\_t const \\*u\) L4\\_NOTHROW](#)  
*Check whether an exception IPC is a page fault.*
- [l4\\_addr\\_t l4\\_utcb\\_exc\\_pfa \(l4\\_exc\\_regs\\_t const \\*u\) L4\\_NOTHROW](#)  
*Function to get the L4 style page fault address out of an exception.*
- [int l4\\_utcb\\_exc\\_is\\_ex\\_regs\\_exception \(l4\\_exc\\_regs\\_t const \\*u\) L4\\_NOTHROW](#)  
*Check whether an exception IPC was triggered via [l4\\_thread\\_ex\\_regs\(\)](#).*

### 17.603.1 Detailed Description

UTCB definitions for AMD64.

Definition in file [utcb.h](#).

## 17.604 utcb.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 /*****
00014 #ifndef __L4_SYS__INCLUDE__ARCH_AMD64__UTCB_H__
00015 #define __L4_SYS__INCLUDE__ARCH_AMD64__UTCB_H__
00016
00017 #include <l4/sys/types.h>
00018
00023
00028 enum l4_utcb_consts_amd64
00029 {
00030 L4_UTCB_EXCEPTION_REGS_SIZE = 26,
00031 L4_UTCB_GENERIC_DATA_SIZE = 63,
00032 L4_UTCB_GENERIC_BUFFERS_SIZE = 58,
00033
00034 L4_UTCB_MSG_REGS_OFFSET = 0,
00035 L4_UTCB_BUF_REGS_OFFSET = 64 * sizeof(l4_umword_t),
00036 L4_UTCB_THREAD_REGS_OFFSET = 123 * sizeof(l4_umword_t),
00037
00038 L4_UTCB_INHERIT_FPU = 1UL < 24,
00039 L4_UTCB_OFFSET = 1024,
00040 };
00041
00046 typedef struct l4_exc_regs_t
00047 {
00048 l4_umword_t r15;
00049 l4_umword_t r14;
00050 l4_umword_t r13;
00051 l4_umword_t r12;
00052 l4_umword_t r11;
00053 l4_umword_t r10;
00054 l4_umword_t r9;
00055 l4_umword_t r8;
00056 l4_umword_t rdi;
00057 l4_umword_t rsi;
00058 l4_umword_t rbp;

```

```

00059 l4_umword_t pfa;
00060 l4_umword_t rbx;
00061 l4_umword_t rdx;
00062 l4_umword_t rcx;
00063 l4_umword_t rax;
00064
00065 l4_umword_t trapno;
00066 l4_umword_t err;
00067 l4_umword_t ip;
00068 l4_umword_t dummy1;
00069 l4_umword_t flags;
00070 l4_umword_t sp;
00071 l4_umword_t ss;
00072 l4_umword_t fs_base;
00073 l4_umword_t gs_base;
00074 l4_uint16_t ds, es, fs, gs;
00075 } l4_exc_regs_t;
00076
00077
00078 #include_next <l4/sys/utcb.h>
00079
00080 /*
00081 * =====
00082 * Implementations.
00083 */
00084
00085 L4_INLINE l4_utcb_t *l4_utcb_direct(void) L4_NOTHROW
00086 {
00087 l4_utcb_t *res;
00088 __asm__ ("mov %%gs:0, %0 \n" : "=r"(res));
00089 return res;
00090 }
00091
00092 L4_INLINE l4_umword_t l4_utcb_exc_pc(l4_exc_regs_t const *u) L4_NOTHROW
00093 {
00094 return u->ip;
00095 }
00096
00097 L4_INLINE void l4_utcb_exc_pc_set(l4_exc_regs_t *u, l4_addr_t pc) L4_NOTHROW
00098 {
00099 u->ip = pc;
00100 }
00101
00102 L4_INLINE l4_umword_t l4_utcb_exc_typeval(l4_exc_regs_t const *u) L4_NOTHROW
00103 {
00104 return u->trapno;
00105 }
00106
00107 L4_INLINE int l4_utcb_exc_is_pf(l4_exc_regs_t const *u) L4_NOTHROW
00108 {
00109 return u->trapno == 14;
00110 }
00111
00112 L4_INLINE l4_addr_t l4_utcb_exc_pfa(l4_exc_regs_t const *u) L4_NOTHROW
00113 {
00114 return (u->pfa & ~7UL) | (u->err & 2);
00115 }
00116
00117 L4_INLINE int l4_utcb_exc_is_ex_regs_exception(l4_exc_regs_t const *u) L4_NOTHROW
00118 {
00119 return l4_utcb_exc_typeval(u) == 0xff;
00120 }
00121
00122 #endif /* ! __L4_SYS__INCLUDE__ARCH_AMD64__UTCB_H__ */

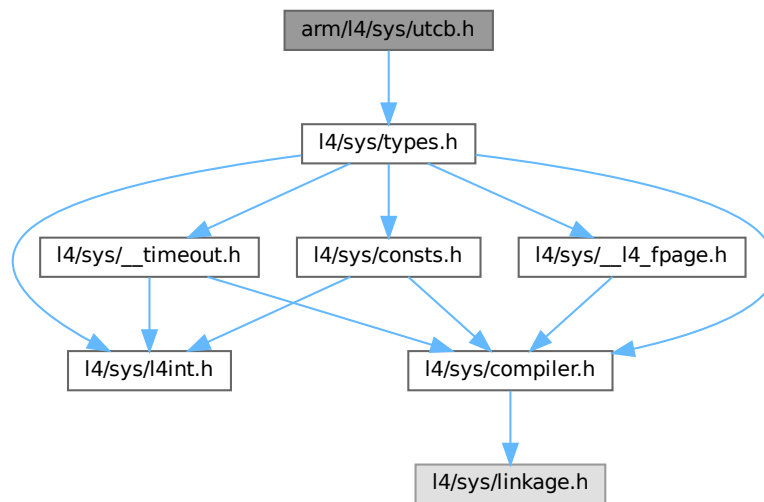
```

## 17.605 arm/l4/sys/utcb.h File Reference

UTCB definitions for ARM.

```
#include <l4/sys/types.h>
```

Include dependency graph for utcb.h:



## Data Structures

- struct `l4_exc_regs_t`  
*UTCB structure for exceptions.*

## Typedefs

- typedef struct `l4_exc_regs_t` `l4_exc_regs_t`  
*UTCB structure for exceptions.*

## Enumerations

- enum `L4_utcb_consts_arm`  
*UTCB constants for ARM.*

## Functions

- `l4_umword_t l4_utcb_exc_pc(l4_exc_regs_t const *u) L4_NOTHROW`  
*Access function to get the program counter of the exception state.*
- `void l4_utcb_exc_pc_set(l4_exc_regs_t *u, l4_addr_t pc) L4_NOTHROW`  
*Set the program counter register in the exception state.*
- `l4_umword_t l4_utcb_exc_typeval(l4_exc_regs_t const *u) L4_NOTHROW`  
*Get the value out of an exception UTCB that describes the type of exception.*
- `int l4_utcb_exc_is_pf(l4_exc_regs_t const *u) L4_NOTHROW`  
*Check whether an exception IPC is a page fault.*
- `l4_addr_t l4_utcb_exc_pfa(l4_exc_regs_t const *u) L4_NOTHROW`  
*Function to get the L4 style page fault address out of an exception.*
- `int l4_utcb_exc_is_ex_regs_exception(l4_exc_regs_t const *u) L4_NOTHROW`  
*Check whether an exception IPC was triggered via `l4_thread_ex_regs()`.*



## 17.605.1 Detailed Description

UTCB definitions for ARM.

Definition in file [utcb.h](#).

## 17.606 utcb.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #ifndef __L4_SYS__INCLUDE__ARCH_ARM__UTCB_H__
00014 #define __L4_SYS__INCLUDE__ARCH_ARM__UTCB_H__
00015
00016 #include <l4/sys/types.h>
00017
00022
00027 typedef struct l4_exc_regs_t
00028 {
00029 l4_umword_t pfa;
00030 l4_umword_t err;
00031
00032 l4_umword_t r[13];
00033 l4_umword_t sp;
00034 l4_umword_t ulr;
00035 l4_umword_t _dummy1;
00036 l4_umword_t pc;
00037 l4_umword_t cpsr;
00038 l4_umword_t tpidruro;
00039 l4_umword_t tpidrurw;
00040 } l4_exc_regs_t;
00041
00047 enum L4_utcb_consts_arm
00048 {
00049 L4_UTCB_EXCEPTION_REGS_SIZE = sizeof(l4_exc_regs_t) / sizeof(l4_umword_t),
00050 L4_UTCB_GENERIC_DATA_SIZE = 63,
00051 L4_UTCB_GENERIC_BUFFERS_SIZE = 58,
00052
00053 L4_UTCB_MSG_REGS_OFFSET = 0,
00054 L4_UTCB_BUF_REGS_OFFSET = 64 * sizeof(l4_umword_t),
00055 L4_UTCB_THREAD_REGS_OFFSET = 123 * sizeof(l4_umword_t),
00056
00057 L4_UTCB_INHERIT_FPU = 1UL << 24,
00058
00059 L4_UTCB_OFFSET = 512,
00060 };
00061
00062 #include_next <l4/sys/utcb.h>
00063
00064 /*
00065 * =====
00066 * Implementations.
00067 */
00068
00069 #ifdef __GNUC__
00070 L4_INLINE l4_utcb_t *l4_utcb_direct(void) L4_NOTHROW
00071 {
00072 #if defined(__ARM_ARCH) && __ARM_ARCH >= 7
00073 l4_utcb_t *utcb;
00074 __asm__ ("mrc p15, 0, %0, c13, c0, 2" : "=r" (utcb)); // TPIDRURW
00075 #else
00076 register l4_utcb_t *utcb __asm__ ("r0");
00077 __asm__ ("mov lr, pc\n"
00078 "mvn pc, #0xff\n"
00079 : "=r"(utcb) : : "lr"); // write 0xfffff00 to pc
00080 #endif
00081 return utcb;
00082 }
00083 #endif
00084
00085 L4_INLINE l4_umword_t l4_utcb_exc_pc(l4_exc_regs_t const *u) L4_NOTHROW
00086 {

```

```

00087 return u->pc;
00088 }
00089
00090 L4_INLINE void l4_utcb_exc_pc_set(l4_exc_regs_t *u, l4_addr_t pc) L4_NOTHROW
00091 {
00092 u->pc = pc;
00093 }
00094
00095 L4_INLINE l4_umword_t l4_utcb_exc_typeval(l4_exc_regs_t const *u) L4_NOTHROW
00096 {
00097 return u->err >> 26;
00098 }
00099
00100 L4_INLINE int l4_utcb_exc_is_pf(l4_exc_regs_t const *u) L4_NOTHROW
00101 {
00102 return ((u->err >> 26) & 0x30) == 0x20;
00103 }
00104
00105 L4_INLINE l4_addr_t l4_utcb_exc_pfa(l4_exc_regs_t const *u) L4_NOTHROW
00106 {
00107 return (u->pfa & ~7UL) | ((u->err >> 5) & 2);
00108 }
00109
00110 L4_INLINE int l4_utcb_exc_is_ex_regs_exception(l4_exc_regs_t const *u) L4_NOTHROW
00111 {
00112 return l4_utcb_exc_typeval(u) == 0x3e;
00113 }
00114
00115 #endif /* ! __L4_SYS__INCLUDE__ARCH_ARM__UTCB_H__ */

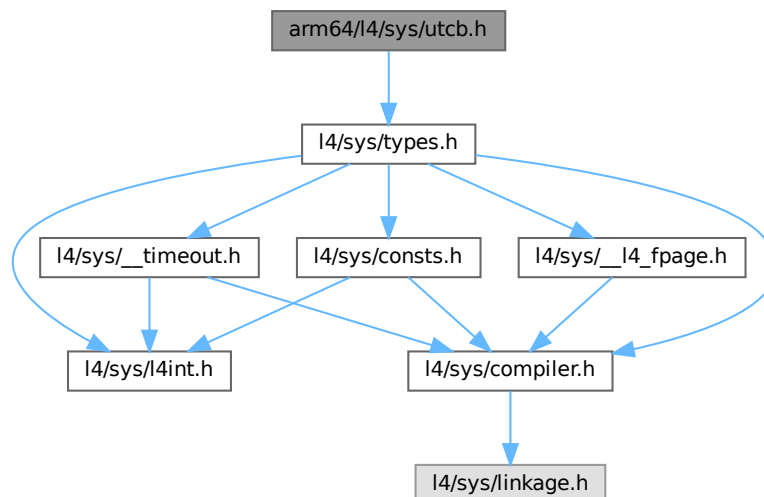
```

## 17.607 arm64/l4/sys/utcb.h File Reference

UTCB definitions for ARM64.

```
#include <l4/sys/types.h>
```

Include dependency graph for utcb.h:



### Data Structures

- struct `l4_exc_regs_t`  
*UTCB structure for exceptions.*

## Typedefs

- typedef struct l4\_exc\_regs\_t l4\_exc\_regs\_t  
*UTCB structure for exceptions.*

## Enumerations

- enum L4\_utcb\_consts\_arm64  
*UTCB constants for ARM64.*

## Functions

- l4\_umword\_t l4\_utcb\_exc\_pc (l4\_exc\_regs\_t const \*u) L4\_NOTHROW  
*Access function to get the program counter of the exception state.*
- void l4\_utcb\_exc\_pc\_set (l4\_exc\_regs\_t \*u, l4\_addr\_t pc) L4\_NOTHROW  
*Set the program counter register in the exception state.*
- l4\_umword\_t l4\_utcb\_exc\_typeval (l4\_exc\_regs\_t const \*u) L4\_NOTHROW  
*Get the value out of an exception UTCB that describes the type of exception.*
- int l4\_utcb\_exc\_is\_pf (l4\_exc\_regs\_t const \*u) L4\_NOTHROW  
*Check whether an exception IPC is a page fault.*
- l4\_addr\_t l4\_utcb\_exc\_pfa (l4\_exc\_regs\_t const \*u) L4\_NOTHROW  
*Function to get the L4 style page fault address out of an exception.*
- int l4\_utcb\_exc\_is\_ex\_regs\_exception (l4\_exc\_regs\_t const \*u) L4\_NOTHROW  
*Check whether an exception IPC was triggered via l4\_thread\_ex\_regs().*

## 17.607.1 Detailed Description

UTCB definitions for ARM64.

Definition in file [utcb.h](#).

## 17.608 utcb.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #ifndef __L4_SYS__INCLUDE__ARCH_ARM64__UTCB_H__
00014 #define __L4_SYS__INCLUDE__ARCH_ARM64__UTCB_H__
00015
00016 #include <l4/sys/types.h>
00017
00022
00027 typedef struct l4_exc_regs_t
00028 {
00029 l4_umword_t eret_work;
00030 l4_umword_t r[31];
00031 l4_umword_t reserved;
00032 l4_umword_t err;
00033
00034 l4_umword_t pfa;
00035 l4_umword_t sp;

```

```

00036 union { l4_umword_t ip; l4_umword_t pc; }; /* aliases for PC */
00037 union { l4_umword_t flags; l4_umword_t pstate; }; /* aliases for PSTATE (PSR) */
00038 l4_umword_t tpidruro;
00039 l4_umword_t tpidrurw;
00040 } l4_exc_regs_t;
00041
00047 enum L4_utcb_consts_arm64
00048 {
00049 L4_UTCB_EXCEPTION_REGS_SIZE = sizeof(l4_exc_regs_t) / sizeof(l4_umword_t),
00050 L4_UTCB_GENERIC_DATA_SIZE = 63,
00051 L4_UTCB_GENERIC_BUFFERS_SIZE = 58,
00052
00053 L4_UTCB_MSG_REGS_OFFSET = 0,
00054 L4_UTCB_BUF_REGS_OFFSET = 64 * sizeof(l4_umword_t),
00055 L4_UTCB_THREAD_REGS_OFFSET = 123 * sizeof(l4_umword_t),
00056
00057 L4_UTCB_INHERIT_FPU = 1UL < 24,
00058
00059 L4_UTCB_OFFSET = 1024,
00060 };
00061
00062 #include_next <l4/sys/utcb.h>
00063
00064 /*
00065 * =====
00066 * Implementations.
00067 */
00068
00069 #ifdef __GNUC__
00070 L4_INLINE l4_utcb_t *l4_utcb_direct(void) L4_NOTHROW
00071 {
00072 l4_utcb_t *utcb;
00073 __asm__ ("mrs %0, TPIDRRO_EL0" : "=r" (utcb));
00074 return utcb;
00075 }
00076 #endif
00077
00078 L4_INLINE l4_umword_t l4_utcb_exc_pc(l4_exc_regs_t const *u) L4_NOTHROW
00079 {
00080 return u->pc;
00081 }
00082
00083 L4_INLINE void l4_utcb_exc_pc_set(l4_exc_regs_t *u, l4_addr_t pc) L4_NOTHROW
00084 {
00085 u->pc = pc;
00086 }
00087
00088 L4_INLINE l4_umword_t l4_utcb_exc_typeval(l4_exc_regs_t const *u) L4_NOTHROW
00089 {
00090 return u->err >> 26;
00091 }
00092
00093 L4_INLINE int l4_utcb_exc_is_pf(l4_exc_regs_t const *u) L4_NOTHROW
00094 {
00095 return ((u->err >> 26) & 0x30) == 0x20;
00096 }
00097
00098 L4_INLINE l4_addr_t l4_utcb_exc_pfa(l4_exc_regs_t const *u) L4_NOTHROW
00099 {
00100 return (u->pfa & ~7UL) | ((u->err >> 5) & 2);
00101 }
00102
00103 L4_INLINE int l4_utcb_exc_is_ex_regs_exception(l4_exc_regs_t const *u) L4_NOTHROW
00104 {
00105 return (u->err >> 26) == 0x3e;
00106 }
00107
00108 #endif /* ! __L4_SYS__INCLUDE__ARCH_ARM64__UTCB_H__ */

```

## 17.609 l4/sys/utcb.h File Reference

UTCB definitions.

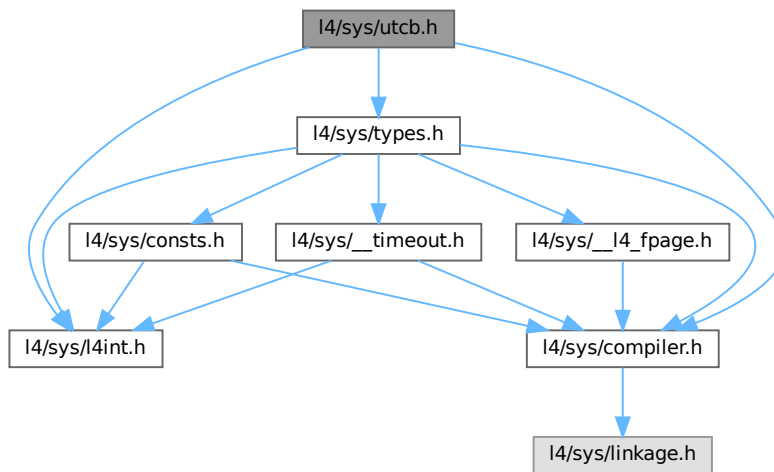
```

#include <l4/sys/types.h>
#include <l4/sys/compiler.h>

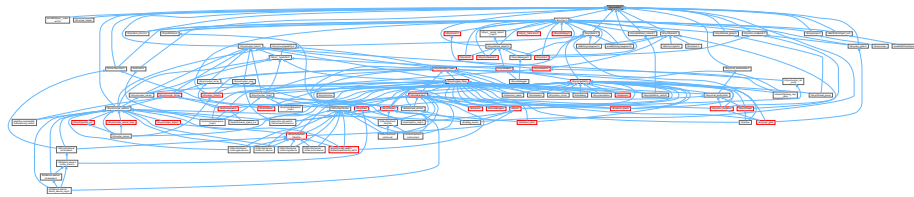
```

```
#include <l4/sys/l4int.h>
```

Include dependency graph for utcb.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- union [l4\\_msg\\_regs\\_t](#)  
*Encapsulation of the message-register block in the UTCB.*
- struct [l4\\_buf\\_regs\\_t](#)  
*Encapsulation of the buffer-registers block in the UTCB.*
- struct [l4\\_thread\\_regs\\_t](#)  
*Encapsulation of the thread-control-register block of the UTCB.*

## Typedefs

- typedef struct [l4\\_utcb\\_t](#) [l4\\_utcb\\_t](#)  
*Opaque type for the UTCB.*
- typedef union [l4\\_msg\\_regs\\_t](#) [l4\\_msg\\_regs\\_t](#)  
*Encapsulation of the message-register block in the UTCB.*
- typedef struct [l4\\_buf\\_regs\\_t](#) [l4\\_buf\\_regs\\_t](#)  
*Encapsulation of the buffer-registers block in the UTCB.*
- typedef struct [l4\\_thread\\_regs\\_t](#) [l4\\_thread\\_regs\\_t](#)  
*Encapsulation of the thread-control-register block of the UTCB.*

## Functions

- `l4_utcb_t * l4_utcb` (void) `L4_NOTHROW` `L4_PURE`  
*Get the UTCB address.*
- `l4_msg_regs_t * l4_utcb_mr` (void) `L4_NOTHROW` `L4_PURE`  
*Get the message-register block of a UTCB.*
- `l4_buf_regs_t * l4_utcb_br` (void) `L4_NOTHROW` `L4_PURE`  
*Get the buffer-register block of a UTCB.*
- `l4_thread_regs_t * l4_utcb_tcr` (void) `L4_NOTHROW` `L4_PURE`  
*Get the thread-control-register block of a UTCB.*
- `l4_exc_regs_t * l4_utcb_exc` (void) `L4_NOTHROW` `L4_PURE`  
*Get the message-register block of a UTCB (for an exception IPC).*
- `l4_umword_t l4_utcb_exc_pc` (`l4_exc_regs_t` const \*u) `L4_NOTHROW` `L4_PURE`  
*Access function to get the program counter of the exception state.*
- `void l4_utcb_exc_pc_set` (`l4_exc_regs_t` \*u, `l4_addr_t` pc) `L4_NOTHROW`  
*Set the program counter register in the exception state.*
- `unsigned long l4_utcb_exc_typeval` (`l4_exc_regs_t` const \*u) `L4_NOTHROW` `L4_PURE`  
*Get the value out of an exception UTCB that describes the type of exception.*
- `int l4_utcb_exc_is_pf` (`l4_exc_regs_t` const \*u) `L4_NOTHROW` `L4_PURE`  
*Check whether an exception IPC is a page fault.*
- `l4_addr_t l4_utcb_exc_pfa` (`l4_exc_regs_t` const \*u) `L4_NOTHROW` `L4_PURE`  
*Function to get the L4 style page fault address out of an exception.*
- `int l4_utcb_exc_is_ex_regs_exception` (`l4_exc_regs_t` const \*u) `L4_NOTHROW` `L4_PURE`  
*Check whether an exception IPC was triggered via `l4_thread_ex_regs()`.*
- `void l4_utcb_inherit_fpu` (int switch\_on) `L4_NOTHROW`  
*Enable or disable inheritance of FPU state to receiver.*
- `l4_timeout_s l4_timeout_abs` (`l4_kernel_clock_t` pint, int br) `L4_NOTHROW`  
*Set an absolute timeout.*
- `unsigned l4_utcb_mr64_idx` (unsigned idx) `L4_NOTHROW`  
*Get index into 64bit message registers alias from native-sized index.*

## 17.609.1 Detailed Description

UTCB definitions.

Definition in file `utcb.h`.

## 17.610 utcb.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002 */
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00005 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00006 * economic rights: Technische Universität Dresden (Germany)
00007 *
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010 /*****
00011 #ifndef _L4_SYS_UTCB_H
00012 #define _L4_SYS_UTCB_H
00013 #include <l4/sys/types.h>

```

```

00020 #include <l4/sys/compiler.h>
00021 #include <l4/sys/l4int.h>
00022
00047
00056 typedef struct l4_utcb_t l4_utcb_t;
00057
00062
00067 typedef union l4_msg_regs_t
00068 {
00069 l4_umword_t mr[L4_UTCB_GENERIC_DATA_SIZE];
00070 l4_uint64_t mr64[L4_UTCB_GENERIC_DATA_SIZE / (sizeof(l4_uint64_t)/sizeof(l4_umword_t))];
00071 } l4_msg_regs_t;
00072
00082 typedef struct l4_buf_regs_t
00083 {
00085 l4_umword_t bdr;
00086
00088 l4_umword_t br[L4_UTCB_GENERIC_BUFFERS_SIZE];
00089 } l4_buf_regs_t;
00090
00099 typedef struct l4_thread_regs_t
00100 {
00106 l4_umword_t error;
00107
00120 l4_umword_t free_marker;
00121
00123 l4_umword_t user[3];
00124 } l4_thread_regs_t;
00125
00126 L4_BEGIN_DECLS
00127
00138 L4_CV l4_utcb_t *l4_utcb_wrap(void) L4_NOTHROW L4_PURE;
00139
00145 L4_INLINE l4_utcb_t *l4_utcb_direct(void) L4_NOTHROW L4_PURE;
00146
00151 L4_INLINE l4_utcb_t *l4_utcb(void) L4_NOTHROW L4_PURE;
00152
00158 L4_INLINE l4_msg_regs_t *l4_utcb_mr(void) L4_NOTHROW L4_PURE;
00159
00164 L4_INLINE l4_msg_regs_t *l4_utcb_mr_u(l4_utcb_t *u) L4_NOTHROW L4_PURE;
00165
00172 L4_INLINE l4_buf_regs_t *l4_utcb_br(void) L4_NOTHROW L4_PURE;
00173
00178 L4_INLINE l4_buf_regs_t *l4_utcb_br_u(l4_utcb_t *u) L4_NOTHROW L4_PURE;
00179
00185 L4_INLINE l4_thread_regs_t *l4_utcb_tcr(void) L4_NOTHROW L4_PURE;
00186
00191 L4_INLINE l4_thread_regs_t *l4_utcb_tcr_u(l4_utcb_t *u) L4_NOTHROW L4_PURE;
00192
00198
00205 L4_INLINE l4_exc_regs_t *l4_utcb_exc(void) L4_NOTHROW L4_PURE;
00206
00211 L4_INLINE l4_exc_regs_t *l4_utcb_exc_u(l4_utcb_t *u) L4_NOTHROW L4_PURE;
00212
00220 L4_INLINE l4_umword_t l4_utcb_exc_pc(l4_exc_regs_t const *u) L4_NOTHROW L4_PURE;
00221
00230 L4_INLINE void l4_utcb_exc_pc_set(l4_exc_regs_t *u, l4_addr_t pc) L4_NOTHROW;
00231
00236 L4_INLINE unsigned long l4_utcb_exc_typeval(l4_exc_regs_t const *u) L4_NOTHROW L4_PURE;
00237
00247 L4_INLINE int l4_utcb_exc_is_pf(l4_exc_regs_t const *u) L4_NOTHROW L4_PURE;
00248
00253 L4_INLINE l4_addr_t l4_utcb_exc_pfa(l4_exc_regs_t const *u) L4_NOTHROW L4_PURE;
00254
00255
00266 L4_INLINE int l4_utcb_exc_is_ex_regs_exception(l4_exc_regs_t const *u) L4_NOTHROW L4_PURE;
00267
00272 L4_INLINE void l4_utcb_inherit_fpu(int switch_on) L4_NOTHROW;
00273
00277 L4_INLINE void l4_utcb_inherit_fpu_u(l4_utcb_t *u, int switch_on) L4_NOTHROW;
00278
00293 L4_INLINE
00294 l4_timeout_s l4_timeout_abs_u(l4_kernel_clock_t pint, int br,
00295 l4_utcb_t *utcb) L4_NOTHROW;
00309 L4_INLINE
00310 l4_timeout_s l4_timeout_abs(l4_kernel_clock_t pint, int br) L4_NOTHROW;
00311
00319 L4_INLINE
00320 unsigned l4_utcb_mr64_idx(unsigned idx) L4_NOTHROW;
00321
00322 /*****
00323 * Implementations
00324 *****/
00325
00326 L4_INLINE l4_msg_regs_t *l4_utcb_mr_u(l4_utcb_t *u) L4_NOTHROW
00327 { return (l4_msg_regs_t*)((char*)u + L4_UTCB_MSG_REGS_OFFSET); }
00328

```

```

00329 L4_INLINE l4_buf_regs_t *l4_utcb_br_u(l4_utcb_t *u) L4_NOTHROW
00330 { return (l4_buf_regs_t*)((char*)u + L4_UTCB_BUF_REGS_OFFSET); }
00331
00332 L4_INLINE l4_thread_regs_t *l4_utcb_tcr_u(l4_utcb_t *u) L4_NOTHROW
00333 { return (l4_thread_regs_t*)((char*)u + L4_UTCB_THREAD_REGS_OFFSET); }
00334
00335 L4_INLINE l4_exc_regs_t *l4_utcb_exc_u(l4_utcb_t *u) L4_NOTHROW
00336 { return (l4_exc_regs_t*)((char*)u + L4_UTCB_MSG_REGS_OFFSET); }
00337
00338 L4_INLINE void l4_utcb_inherit_fpu_u(l4_utcb_t *u, int switch_on) L4_NOTHROW
00339 {
00340 if (switch_on)
00341 l4_utcb_br_u(u)->bdr |= L4_UTCB_INHERIT_FPU;
00342 else
00343 l4_utcb_br_u(u)->bdr &= ~L4_UTCB_INHERIT_FPU;
00344 }
00345
00346 L4_INLINE l4_utcb_t *l4_utcb(void) L4_NOTHROW
00347 {
00348 #ifdef L4SYS_USE_UTCB_WRAP
00349 return l4_utcb_wrap();
00350 #else
00351 return l4_utcb_direct();
00352 #endif
00353 }
00354
00355
00356
00357
00358 L4_INLINE l4_msg_regs_t *l4_utcb_mr(void) L4_NOTHROW
00359 { return l4_utcb_mr_u(l4_utcb()); }
00360
00361 L4_INLINE l4_buf_regs_t *l4_utcb_br(void) L4_NOTHROW
00362 { return l4_utcb_br_u(l4_utcb()); }
00363
00364 L4_INLINE l4_thread_regs_t *l4_utcb_tcr(void) L4_NOTHROW
00365 { return l4_utcb_tcr_u(l4_utcb()); }
00366
00367 L4_INLINE l4_exc_regs_t *l4_utcb_exc(void) L4_NOTHROW
00368 { return l4_utcb_exc_u(l4_utcb()); }
00369
00370 L4_INLINE void l4_utcb_inherit_fpu(int switch_on) L4_NOTHROW
00371 { l4_utcb_inherit_fpu_u(l4_utcb(), switch_on); }
00372
00373 L4_INLINE
00374 l4_timeout_s l4_timeout_abs_u(l4_kernel_clock_t val, int pos,
00375 l4_utcb_t *utcb) L4_NOTHROW
00376 {
00377 union T
00378 {
00379 l4_kernel_clock_t t;
00380 l4_umword_t m[sizeof(l4_kernel_clock_t)/sizeof(l4_umword_t)];
00381 };
00382 l4_timeout_s to;
00383 to.t = 0x8000 | pos;
00384 ((union T*)(l4_utcb_br_u(utcb)->br + pos))->t = val;
00385 return to;
00386 }
00387
00388 L4_INLINE
00389 l4_timeout_s l4_timeout_abs(l4_kernel_clock_t val, int pos) L4_NOTHROW
00390 { return l4_timeout_abs_u(val, pos, l4_utcb()); }
00391
00392 L4_INLINE unsigned l4_utcb_mr64_idx(unsigned idx) L4_NOTHROW
00393 { return idx / (sizeof(l4_uint64_t) / sizeof(l4_umword_t)); }
00394
00395 L4_END_DECLS
00396
00397 #endif /* ! _L4_SYS_UTCB_H */

```

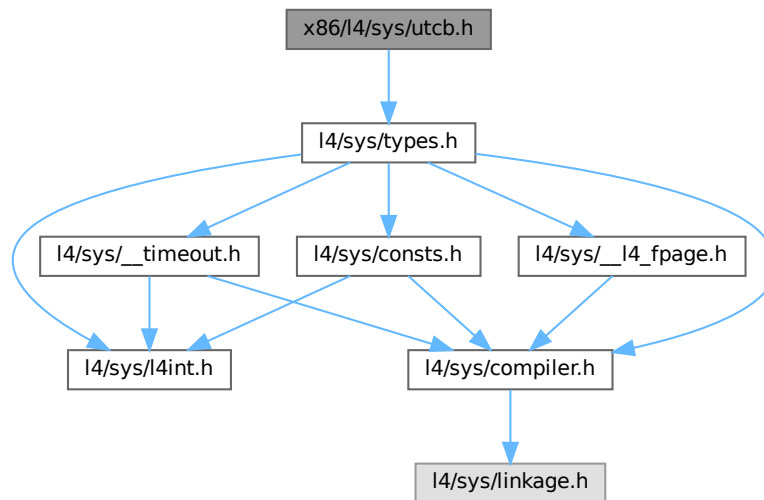
## 17.611 x86/i4/sys/utcb.h File Reference

UTCB definitions for x86.



```
#include <l4/sys/types.h>
```

Include dependency graph for utcb.h:



## Data Structures

- struct [l4\\_exc\\_regs\\_t](#)  
*UTCB structure for exceptions.*

## Typedefs

- typedef struct [l4\\_exc\\_regs\\_t](#) [l4\\_exc\\_regs\\_t](#)  
*UTCB structure for exceptions.*

## Enumerations

- enum [L4\\_utcb\\_consts\\_x86](#) {  
[L4\\_UTCB\\_EXCEPTION\\_REGS\\_SIZE](#) = 19 , [L4\\_UTCB\\_GENERIC\\_DATA\\_SIZE](#) = 63 , [L4\\_UTCB\\_GENERIC\\_BUFFERS\\_SIZE](#) = 58 , [L4\\_UTCB\\_MSG\\_REGS\\_OFFSET](#) = 0 ,  
[L4\\_UTCB\\_BUF\\_REGS\\_OFFSET](#) = 64 \* sizeof([l4\\_umword\\_t](#)) , [L4\\_UTCB\\_THREAD\\_REGS\\_OFFSET](#) = 123 \* sizeof([l4\\_umword\\_t](#)) , [L4\\_UTCB\\_INHERIT\\_FPU](#) = 1UL << 24 , [L4\\_UTCB\\_OFFSET](#) = 512 }  
*UTCB constants for x86.*

## Functions

- [l4\\_umword\\_t](#) [l4\\_utcb\\_exc\\_pc](#) ([l4\\_exc\\_regs\\_t](#) const \*u) [L4\\_NOTHROW](#)  
*Access function to get the program counter of the exception state.*
- void [l4\\_utcb\\_exc\\_pc\\_set](#) ([l4\\_exc\\_regs\\_t](#) \*u, [l4\\_addr\\_t](#) pc) [L4\\_NOTHROW](#)  
*Set the program counter register in the exception state.*
- [l4\\_umword\\_t](#) [l4\\_utcb\\_exc\\_typeval](#) ([l4\\_exc\\_regs\\_t](#) const \*u) [L4\\_NOTHROW](#)

*Get the value out of an exception UTCB that describes the type of exception.*

- `int l4_utcb_exc_is_pf (l4_exc_regs_t const *u) L4_NOTHROW`

*Check whether an exception IPC is a page fault.*

- `l4_addr_t l4_utcb_exc_pfa (l4_exc_regs_t const *u) L4_NOTHROW`

*Function to get the L4 style page fault address out of an exception.*

- `int l4_utcb_exc_is_ex_regs_exception (l4_exc_regs_t const *u) L4_NOTHROW`

*Check whether an exception IPC was triggered via `l4_thread_ex_regs()`.*

## 17.611.1 Detailed Description

UTCB definitions for x86.

Definition in file `utcb.h`.

## 17.612 utcb.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002 */
00003 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 /*****
00010 */
00011 #ifndef __L4_SYS__INCLUDE__ARCH_X86__UTCB_H__
00012 #define __L4_SYS__INCLUDE__ARCH_X86__UTCB_H__
00013
00014 #include <l4/sys/types.h>
00015
00016 enum l4_utcb_consts_x86
00017 {
00018 L4_UTCB_EXCEPTION_REGS_SIZE = 19,
00019 L4_UTCB_GENERIC_DATA_SIZE = 63,
00020 L4_UTCB_GENERIC_BUFFERS_SIZE = 58,
00021 L4_UTCB_MSG_REGS_OFFSET = 0,
00022 L4_UTCB_BUF_REGS_OFFSET = 64 * sizeof(l4_umword_t),
00023 L4_UTCB_THREAD_REGS_OFFSET = 123 * sizeof(l4_umword_t),
00024 L4_UTCB_INHERIT_FPU = 1UL < 24,
00025 L4_UTCB_OFFSET = 512,
00026 };
00027
00028 typedef struct l4_exc_regs_t
00029 {
00030 l4_umword_t es;
00031 l4_umword_t ds;
00032 l4_umword_t gs;
00033 l4_umword_t fs;
00034
00035 l4_umword_t edi;
00036 l4_umword_t esi;
00037 l4_umword_t ebp;
00038 l4_umword_t pfa;
00039 l4_umword_t ebx;
00040 l4_umword_t edx;
00041 l4_umword_t ecx;
00042 l4_umword_t eax;
00043
00044 l4_umword_t trapno;
00045 l4_umword_t err;
00046 }

```

```

00080 l4_umword_t ip;
00081 l4_umword_t dummy1;
00082 l4_umword_t flags;
00083 l4_umword_t sp;
00084 l4_umword_t ss;
00085 } l4_exc_regs_t;
00086
00087 #include_next <l4/sys/utcb.h>
00088
00089 /*
00090 * =====
00091 * Implementations.
00092 */
00093
00094 L4_INLINE l4_utcb_t *l4_utcb_direct(void) L4_NOTHROW
00095 {
00096 l4_utcb_t *utcb;
00097 __asm__ ("mov %%fs:0, %0" : "=r" (utcb));
00098 return utcb;
00099 }
00100
00101 L4_INLINE l4_umword_t l4_utcb_exc_pc(l4_exc_regs_t const *u) L4_NOTHROW
00102 {
00103 return u->ip;
00104 }
00105
00106 L4_INLINE void l4_utcb_exc_pc_set(l4_exc_regs_t *u, l4_addr_t pc) L4_NOTHROW
00107 {
00108 u->ip = pc;
00109 }
00110
00111 L4_INLINE void l4_utcb_exc_sp_set(l4_exc_regs_t *u, l4_addr_t sp) L4_NOTHROW
00112 {
00113 u->sp = sp;
00114 }
00115
00116 L4_INLINE l4_umword_t l4_utcb_exc_typeval(l4_exc_regs_t const *u) L4_NOTHROW
00117 {
00118 return u->trapno;
00119 }
00120
00121 L4_INLINE int l4_utcb_exc_is_pf(l4_exc_regs_t const *u) L4_NOTHROW
00122 {
00123 return u->trapno == 14;
00124 }
00125
00126 L4_INLINE l4_addr_t l4_utcb_exc_pfa(l4_exc_regs_t const *u) L4_NOTHROW
00127 {
00128 return (u->pfa & ~7UL) | (u->err & 2);
00129 }
00130
00131 L4_INLINE int l4_utcb_exc_is_ex_regs_exception(l4_exc_regs_t const *u) L4_NOTHROW
00132 {
00133 return l4_utcb_exc_typeval(u) == 0xff;
00134 }
00135
00136 #endif /* ! __L4_SYS__INCLUDE__ARCH_X86__UTCB_H__ */

```

## 17.613 l4/sys/vcon File Reference

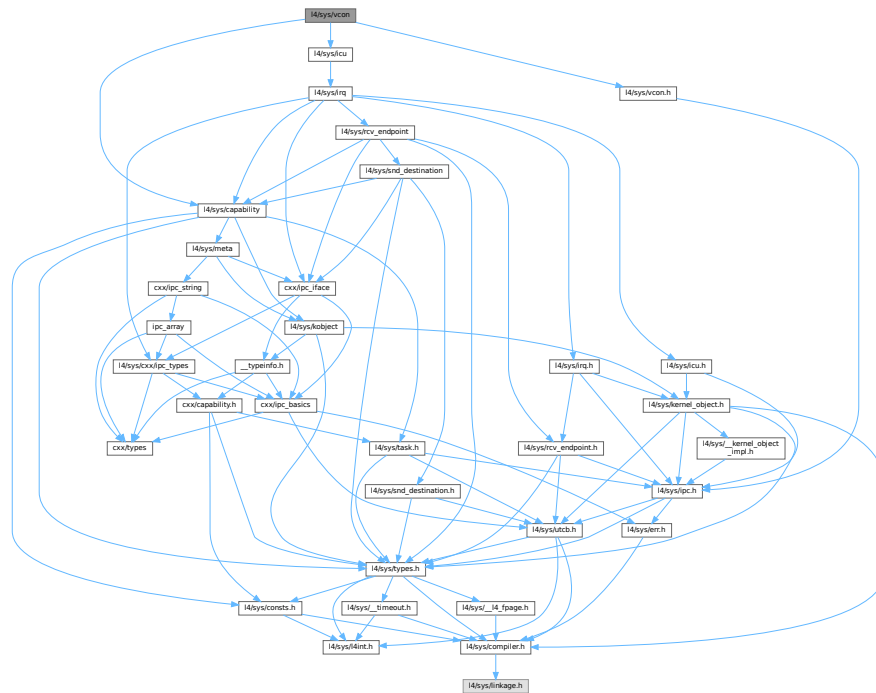
C++ Virtual console interface.

```

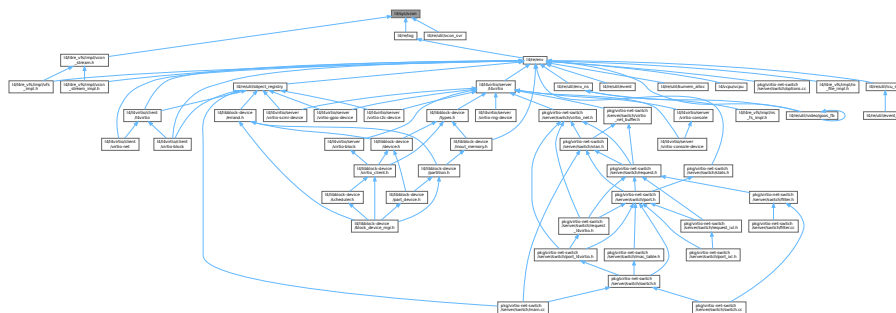
#include <l4/sys/icu>
#include <l4/sys/vcon.h>
#include <l4/sys/capability>

```

Include dependency graph for vcon:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class [L4::Vcon](#)  
*C++ L4 Vcon interface, see [Virtual Console](#) for the C interface.*

## Namespaces

- namespace **L4**  
*L4 low-level kernel interface.*

## 17.613.1 Detailed Description

C++ Virtual console interface.

Definition in file [vcon](#).

## 17.614 vcon

[Go to the documentation of this file.](#)

```

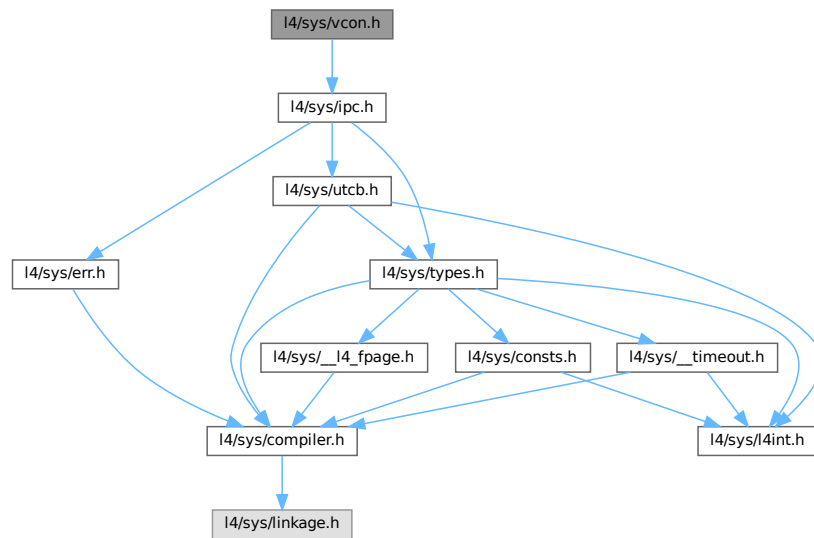
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00009 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #pragma once
00015
00016 #include <l4/sys/icu>
00017 #include <l4/sys/vcon.h>
00018 #include <l4/sys/capability>
00019
00020 namespace L4 {
00021
00045 class Vcon :
00046 public Kobject_t<Vcon, Icu, L4_PROTO_LOG>
00047 {
00048 public:
00064 l4_msgtag_t
00065 send(char const *buf, unsigned size, l4_utcb_t *utcb = l4_utcb()) const noexcept
00066 { return l4_vcon_send_u(cap(), buf, size, utcb); }
00067
00078 long
00079 write(char const *buf, unsigned size, l4_utcb_t *utcb = l4_utcb()) const noexcept
00080 { return l4_vcon_write_u(cap(), buf, size, utcb); }
00081
00097 int
00098 read(char *buf, unsigned size, l4_utcb_t *utcb = l4_utcb()) const noexcept
00099 { return l4_vcon_read_u(cap(), buf, size, utcb); }
00100
00124 int
00125 read_with_flags(char *buf, unsigned size, l4_utcb_t *utcb = l4_utcb()) const noexcept
00126 { return l4_vcon_read_with_flags_u(cap(), buf, size, utcb); }
00127
00137 l4_msgtag_t
00138 set_attr(l4_vcon_attr_t const *attr, l4_utcb_t *utcb = l4_utcb()) const noexcept
00139 { return l4_vcon_set_attr_u(cap(), attr, utcb); }
00140
00150 l4_msgtag_t
00151 get_attr(l4_vcon_attr_t *attr, l4_utcb_t *utcb = l4_utcb()) const noexcept
00152 { return l4_vcon_get_attr_u(cap(), attr, utcb); }
00153
00154 typedef L4::Typeid::Raw_ipc<Vcon> Rpcs;
00155 };
00156
00157 }
```

## 17.615 l4/sys/vcon.h File Reference

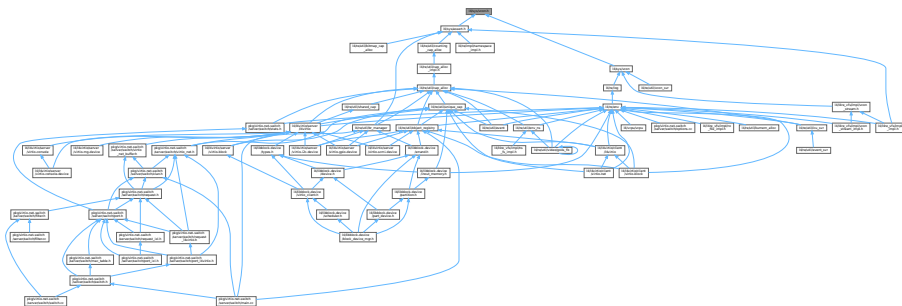
Virtual console interface.

```
#include <l4/sys/ipc.h>
```

Include dependency graph for vcon.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct `l4_vcon_attr_t`  
*Vcon attribute structure.*

## Typedefs

- typedef struct `l4_vcon_attr_t` `l4_vcon_attr_t`  
*Vcon attribute structure.*

## Enumerations

- enum `L4_vcon_size_consts` { `L4_VCON_WRITE_SIZE` = (L4\_UTCB\_GENERIC\_DATA\_SIZE - 2) \* sizeof(l4\_umword\_t) , `L4_VCON_READ_SIZE` = (L4\_UTCB\_GENERIC\_DATA\_SIZE - 1) \* sizeof(l4\_umword\_t) }  
*Size constants.*
- enum `L4_vcon_read_flags` { `L4_VCON_READ_SIZE_MASK` = 0x3ffffff , `L4_VCON_READ_STAT_BREAK` = 1 << 30 , `L4_VCON_READ_STAT_DONE` = 1 << 31 }  
*Vcon read flags.*
- enum `L4_vcon_i_flags` { `L4_VCON_INLCR` = 000100 , `L4_VCON_IGNCR` = 000200 , `L4_VCON_ICRNL` = 000400 }  
*Input flags.*
- enum `L4_vcon_o_flags` { `L4_VCON_ONLCR` = 000004 , `L4_VCON_OCRNL` = 000010 , `L4_VCON_ONLRET` = 000040 }  
*Output flags.*
- enum `L4_vcon_l_flags` { `L4_VCON_ICANON` = 000002 , `L4_VCON_ECHO` = 000010 }  
*Local flags.*
- enum `L4_vcon_ops` { `L4_VCON_WRITE_OP` = 0UL , `L4_VCON_READ_OP` = 1UL , `L4_VCON_SET_ATTR_OP` = 2UL , `L4_VCON_GET_ATTR_OP` = 3UL }  
*Operations on vcon objects.*

## Functions

- `l4_msgtag_t l4_vcon_send` (l4\_cap\_idx\_t vcon, char const \*buf, unsigned size) `L4_NOTHROW`  
*Send data to virtual console.*
- `l4_msgtag_t l4_vcon_send_u` (l4\_cap\_idx\_t vcon, char const \*buf, unsigned size, l4\_utcb\_t \*utcb) `L4_NOTHROW`  
*Send data to *this* virtual console.*
- `long l4_vcon_write` (l4\_cap\_idx\_t vcon, char const \*buf, unsigned size) `L4_NOTHROW`  
*Write data to virtual console.*
- `long l4_vcon_write_u` (l4\_cap\_idx\_t vcon, char const \*buf, unsigned size, l4\_utcb\_t \*utcb) `L4_NOTHROW`  
*Write data to *this* virtual console.*
- `int l4_vcon_read` (l4\_cap\_idx\_t vcon, char \*buf, unsigned size) `L4_NOTHROW`  
*Read data from virtual console.*
- `int l4_vcon_read_u` (l4\_cap\_idx\_t vcon, char \*buf, unsigned size, l4\_utcb\_t \*utcb) `L4_NOTHROW`  
*Read data from *this* virtual console.*
- `int l4_vcon_read_with_flags` (l4\_cap\_idx\_t vcon, char \*buf, unsigned size) `L4_NOTHROW`  
*Read data from virtual console, extended version including flags.*
- `l4_msgtag_t l4_vcon_set_attr` (l4\_cap\_idx\_t vcon, l4\_vcon\_attr\_t const \*attr) `L4_NOTHROW`  
*Set attributes of a Vcon.*
- `l4_msgtag_t l4_vcon_set_attr_u` (l4\_cap\_idx\_t vcon, l4\_vcon\_attr\_t const \*attr, l4\_utcb\_t \*utcb) `L4_NOTHROW`  
*Set the attributes of *this* virtual console.*
- `l4_msgtag_t l4_vcon_get_attr` (l4\_cap\_idx\_t vcon, l4\_vcon\_attr\_t \*attr) `L4_NOTHROW`  
*Get attributes of a Vcon.*
- `l4_msgtag_t l4_vcon_get_attr_u` (l4\_cap\_idx\_t vcon, l4\_vcon\_attr\_t \*attr, l4\_utcb\_t \*utcb) `L4_NOTHROW`  
*Get attributes of *this* virtual console.*
- `void l4_vcon_set_attr_raw` (l4\_vcon\_attr\_t \*attr) `L4_NOTHROW`  
*Set terminal attributes to disable all special processing.*

## 17.615.1 Detailed Description

Virtual console interface.

Definition in file [vcon.h](#).

## 17.615.2 Enumeration Type Documentation

### 17.615.2.1 L4\_vcon\_read\_flags

```
enum L4_vcon_read_flags
```

Vcon read flags.

#### Enumerator

|                         |                       |
|-------------------------|-----------------------|
| L4_VCON_READ_SIZE_MASK  | Size mask.            |
| L4_VCON_READ_STAT_BREAK | Break condition flag. |
| L4_VCON_READ_STAT_DONE  | Done condition flag.  |

Definition at line 170 of file [vcon.h](#).

## 17.616 vcon.h

[Go to the documentation of this file.](#)

```
00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00008 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #pragma once
00014
00015 #include <l4/sys/ipc.h>
00016
00039
00056 L4_INLINE l4_msgtag_t
00057 l4_vcon_send(l4_cap_idx_t vcon, char const *buf, unsigned size) L4_NOTHROW;
00058
00065 L4_INLINE l4_msgtag_t
00066 l4_vcon_send_u(l4_cap_idx_t vcon, char const *buf, unsigned size, l4_utcb_t *utcb) L4_NOTHROW;
00067
00079 L4_INLINE long
00080 l4_vcon_write(l4_cap_idx_t vcon, char const *buf, unsigned size) L4_NOTHROW;
00081
00088 L4_INLINE long
00089 l4_vcon_write_u(l4_cap_idx_t vcon, char const *buf, unsigned size, l4_utcb_t *utcb) L4_NOTHROW;
00090
00095 enum L4_vcon_size_consts
00096 {
00098 L4_VCON_WRITE_SIZE = (L4_UTCB_GENERIC_DATA_SIZE - 2) * sizeof(l4_umword_t),
00100 L4_VCON_READ_SIZE = (L4_UTCB_GENERIC_DATA_SIZE - 1) * sizeof(l4_umword_t),
00101 };
00102
00119 L4_INLINE int
00120 l4_vcon_read(l4_cap_idx_t vcon, char *buf, unsigned size) L4_NOTHROW;
00121
00128 L4_INLINE int
00129 l4_vcon_read_u(l4_cap_idx_t vcon, char *buf, unsigned size, l4_utcb_t *utcb) L4_NOTHROW;
```



```

00130
00157 L4_INLINE int
00158 l4_vcon_read_with_flags(l4_cap_idx_t vcon, char *buf, unsigned size) L4_NOTHROW;
00159
00163 L4_INLINE int
00164 l4_vcon_read_with_flags_u(l4_cap_idx_t vcon, char *buf, unsigned size,
00165 l4_utcb_t *utcb) L4_NOTHROW;
00166
00170 enum L4_vcon_read_flags
00171 {
00172 L4_VCON_READ_SIZE_MASK = 0x3fffffff,
00173 L4_VCON_READ_STAT_BREAK = 1 << 30,
00174 L4_VCON_READ_STAT_DONE = 1 << 31,
00175 };
00176
00187 typedef struct l4_vcon_attr_t
00188 {
00189 l4_umword_t i_flags;
00190 l4_umword_t o_flags;
00191 l4_umword_t l_flags;
00192
00193 #ifdef __cplusplus
00200 inline void set_raw();
00201 #endif
00202 } l4_vcon_attr_t;
00203
00208 enum L4_vcon_i_flags
00209 {
00210 L4_VCON_INLCR = 000100,
00211 L4_VCON_IGNCR = 000200,
00212 L4_VCON_ICRNL = 000400,
00213 };
00214
00219 enum L4_vcon_o_flags
00220 {
00221 L4_VCON_ONLCR = 000004,
00222 L4_VCON_OCRNL = 000010,
00223 L4_VCON_ONLRET = 000040,
00224 };
00225
00230 enum L4_vcon_l_flags
00231 {
00232 L4_VCON_ICANON = 000002,
00233 L4_VCON_ECHO = 000010,
00234 };
00235
00244 L4_INLINE l4_msgtag_t
00245 l4_vcon_set_attr(l4_cap_idx_t vcon, l4_vcon_attr_t const *attr) L4_NOTHROW;
00246
00253 L4_INLINE l4_msgtag_t
00254 l4_vcon_set_attr_u(l4_cap_idx_t vcon, l4_vcon_attr_t const *attr,
00255 l4_utcb_t *utcb) L4_NOTHROW;
00256
00265 L4_INLINE l4_msgtag_t
00266 l4_vcon_get_attr(l4_cap_idx_t vcon, l4_vcon_attr_t *attr) L4_NOTHROW;
00267
00274 L4_INLINE l4_msgtag_t
00275 l4_vcon_get_attr_u(l4_cap_idx_t vcon, l4_vcon_attr_t *attr,
00276 l4_utcb_t *utcb) L4_NOTHROW;
00277
00283 L4_INLINE void
00284 l4_vcon_set_attr_raw(l4_vcon_attr_t *attr) L4_NOTHROW;
00285
00286
00291 enum L4_vcon_ops
00292 {
00293 L4_VCON_WRITE_OP = 0UL,
00294 L4_VCON_READ_OP = 1UL,
00295 L4_VCON_SET_ATTR_OP = 2UL,
00296 L4_VCON_GET_ATTR_OP = 3UL,
00297 };
00298
00299 /***** Implementations *****/
00300
00301 L4_INLINE l4_msgtag_t
00302 l4_vcon_send_u(l4_cap_idx_t vcon, char const *buf, unsigned size, l4_utcb_t *utcb) L4_NOTHROW
00303 {
00304 l4_msg_regs_t *mr = l4_utcb_mr_u(utcb);
00305 mr->mr[0] = L4_VCON_WRITE_OP;
00306 mr->mr[1] = size;
00307 __builtin_memcpy(&mr->mr[2], buf, size);
00308 return l4_ipc_send(vcon, utcb,
00309 l4_msgtag(L4_PROTO_LOG, 2 + l4_bytes_to_mwords(size),
00310 0, L4_MSGTAG_SCHEDULE),
00311 L4_IPC_NEVER);
00312 }
00313

```

```

00314 L4_INLINE l4_msgtag_t
00315 l4_vcon_send(l4_cap_idx_t vcon, char const *buf, unsigned size) L4_NOTHROW
00316 {
00317 return l4_vcon_send_u(vcon, buf, size, l4_utcb());
00318 }
00319
00320 L4_INLINE long
00321 l4_vcon_write_u(l4_cap_idx_t vcon, char const *buf, unsigned size, l4_utcb_t *utcb) L4_NOTHROW
00322 {
00323 l4_msgtag_t t;
00324
00325 if (size > L4_VCON_WRITE_SIZE)
00326 size = L4_VCON_WRITE_SIZE;
00327
00328 t = l4_vcon_send_u(vcon, buf, size, utcb);
00329 if (l4_msgtag_has_error(t))
00330 return l4_error(t);
00331
00332 return (long) size;
00333 }
00334
00335 L4_INLINE long
00336 l4_vcon_write(l4_cap_idx_t vcon, char const *buf, unsigned size) L4_NOTHROW
00337 {
00338 return l4_vcon_write_u(vcon, buf, size, l4_utcb());
00339 }
00340
00341 L4_INLINE int
00342 l4_vcon_read_with_flags_u(l4_cap_idx_t vcon, char *buf, unsigned size,
00343 l4_utcb_t *utcb) L4_NOTHROW
00344 {
00345 int ret;
00346 unsigned r;
00347 l4_msg_regs_t *mr;
00348
00349 mr = l4_utcb_mr_u(utcb);
00350 mr->mr[0] = (size << 16) | L4_VCON_READ_OP;
00351
00352 ret = l4_error_u(l4_ipc_call(vcon, utcb,
00353 l4_msgtag(L4_PROTO_LOG, 1, 0, 0),
00354 L4_IPC_NEVER),
00355 utcb);
00356 if (ret < 0)
00357 return ret;
00358
00359 r = mr->mr[0] & L4_VCON_READ_SIZE_MASK;
00360
00361 if (!(mr->mr[0] & L4_VCON_READ_STAT_DONE)) // !eof
00362 ret = size + 1;
00363 else if (r < size)
00364 ret = r;
00365 else
00366 ret = size;
00367
00368 if (L4_LIKELY(buf != NULL))
00369 __builtin_memcpy(buf, &mr->mr[1], r < size ? r : size);
00370
00371 return ret | (mr->mr[0] & ~(L4_VCON_READ_STAT_DONE | L4_VCON_READ_SIZE_MASK));
00372 }
00373
00374 L4_INLINE int
00375 l4_vcon_read_with_flags(l4_cap_idx_t vcon, char *buf, unsigned size) L4_NOTHROW
00376 {
00377 return l4_vcon_read_with_flags_u(vcon, buf, size, l4_utcb());
00378 }
00379
00380 L4_INLINE int
00381 l4_vcon_read_u(l4_cap_idx_t vcon, char *buf, unsigned size, l4_utcb_t *utcb) L4_NOTHROW
00382 {
00383 int r = l4_vcon_read_with_flags_u(vcon, buf, size, utcb);
00384 if (r < 0)
00385 return r;
00386
00387 return r & L4_VCON_READ_SIZE_MASK;
00388 }
00389
00390 L4_INLINE int
00391 l4_vcon_read(l4_cap_idx_t vcon, char *buf, unsigned size) L4_NOTHROW
00392 {
00393 return l4_vcon_read_u(vcon, buf, size, l4_utcb());
00394 }
00395
00396 L4_INLINE l4_msgtag_t
00397 l4_vcon_set_attr_u(l4_cap_idx_t vcon, l4_vcon_attr_t const *attr,
00398 l4_utcb_t *utcb) L4_NOTHROW
00399 {
00400 l4_msg_regs_t *mr = l4_utcb_mr_u(utcb);

```

```

00401
00402 mr->mr[0] = L4_VCON_SET_ATTR_OP;
00403 __builtin_memcpy(&mr->mr[1], attr, sizeof(*attr));
00404
00405 return l4_ipc_call(vcon, utcb,
00406 l4_msgtag(L4_PROTO_LOG, 4, 0, 0),
00407 L4_IPC_NEVER);
00408 }
00409
00410 L4_INLINE l4_msgtag_t
00411 l4_vcon_set_attr(l4_cap_idx_t vcon, l4_vcon_attr_t const *attr) L4_NOTHROW
00412 {
00413 return l4_vcon_set_attr_u(vcon, attr, l4_utcb());
00414 }
00415
00416 L4_INLINE l4_msgtag_t
00417 l4_vcon_get_attr_u(l4_cap_idx_t vcon, l4_vcon_attr_t *attr,
00418 l4_utcb_t *utcb) L4_NOTHROW
00419 {
00420 l4_msgtag_t res;
00421 l4_msg_regs_t *mr = l4_utcb_mr_u(utcb);
00422
00423 mr->mr[0] = L4_VCON_GET_ATTR_OP;
00424
00425 res = l4_ipc_call(vcon, utcb,
00426 l4_msgtag(L4_PROTO_LOG, 1, 0, 0),
00427 L4_IPC_NEVER);
00428 if (l4_error_u(res, utcb) >= 0)
00429 __builtin_memcpy(attr, &mr->mr[1], sizeof(*attr));
00430
00431 return res;
00432 }
00433
00434 L4_INLINE l4_msgtag_t
00435 l4_vcon_get_attr(l4_cap_idx_t vcon, l4_vcon_attr_t *attr) L4_NOTHROW
00436 {
00437 return l4_vcon_get_attr_u(vcon, attr, l4_utcb());
00438 }
00439
00440 L4_INLINE void
00441 l4_vcon_set_attr_raw(l4_vcon_attr_t *attr) L4_NOTHROW
00442 {
00443 attr->i_flags = 0;
00444 attr->o_flags = 0;
00445 attr->l_flags = 0;
00446 }
00447
00448 #ifdef __cplusplus
00449 inline void
00450 l4_vcon_attr_t::set_raw()
00451 { l4_vcon_set_attr_raw(this); }
00452 #endif

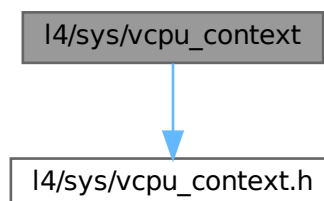
```

## 17.617 l4/sys/vcpu\_context File Reference

Hardware vCPU context interface.

```
#include <l4/sys/vcpu_context.h>
```

Include dependency graph for vcpu\_context:



## Namespaces

- namespace [L4](#)  
*[L4](#) low-level kernel interface.*

### 17.617.1 Detailed Description

Hardware vCPU context interface.

Definition in file [vcpu\\_context](#).

## 17.618 vcpu\_context

[Go to the documentation of this file.](#)

```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00006
00007 #pragma once
00008
00009 #include <l4/sys/vcpu_context.h>
00010
00011 namespace L4 {
00012
00013 class Vcpu_context :
00014 public Kobject_t<Vcpu_context, Kobject, L4_PROTO_VCPU_CONTEXT>
00015 {
00016 public:
00017 Vcpu_context(Vcpu_context const &) = delete;
00018 void operator = (Vcpu_context const &) = delete;
00019
00020 protected:
00021 Vcpu_context();
00022 };
00023
00024 };
```

## 17.619 vcpu\_context.h

```
00001
00006
00007 #pragma once
```

## 17.620 vm

```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2018 Adam Lackorzynski <adam@l4re.org>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/sys/__vm-arm.h>
```

## 17.621 vm

```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2018 Adam Lackorzynski <adam@l4re.org>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/sys/__vm-arm.h>
```

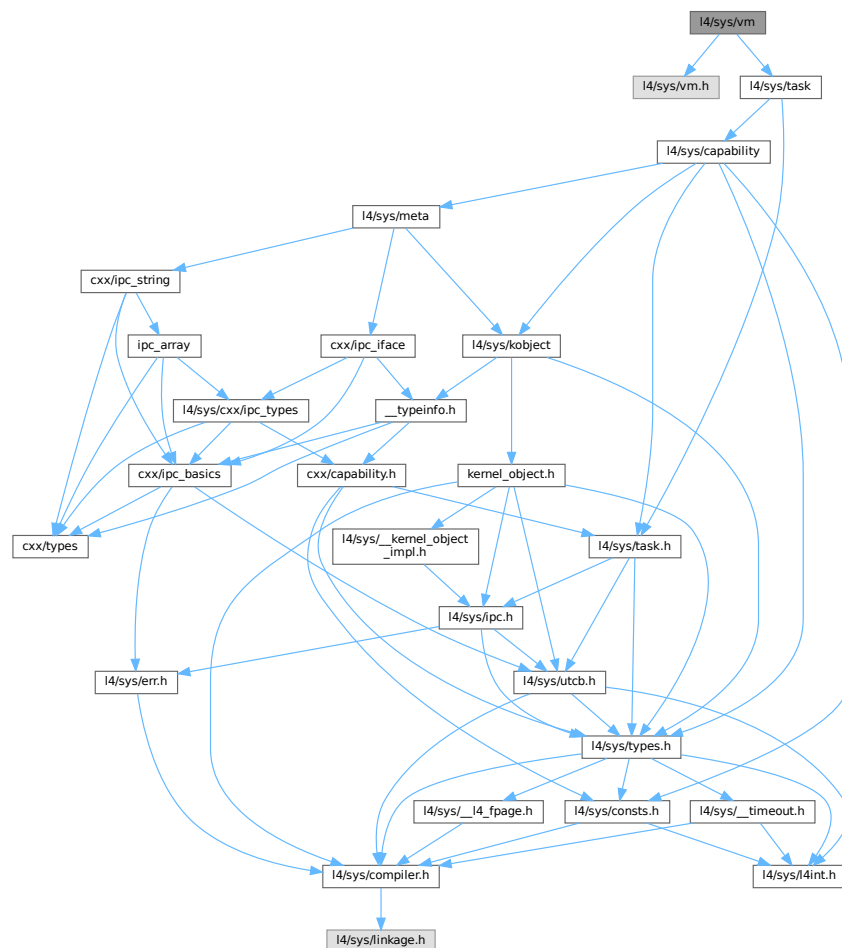
## 17.622 I4/sys/vm File Reference

Virtualization interface.

```
#include <l4/sys/vm.h>
```

```
#include <l4/sys/task>
```

Include dependency graph for vm:



### Data Structures

- class [L4::Vm](#)  
*Virtual machine host address space.*

### Namespaces

- namespace [L4](#)  
*L4 low-level kernel interface.*

### 17.622.1 Detailed Description

Virtualization interface.

Definition in file [vm](#).

## 17.623 vm

[Go to the documentation of this file.](#)

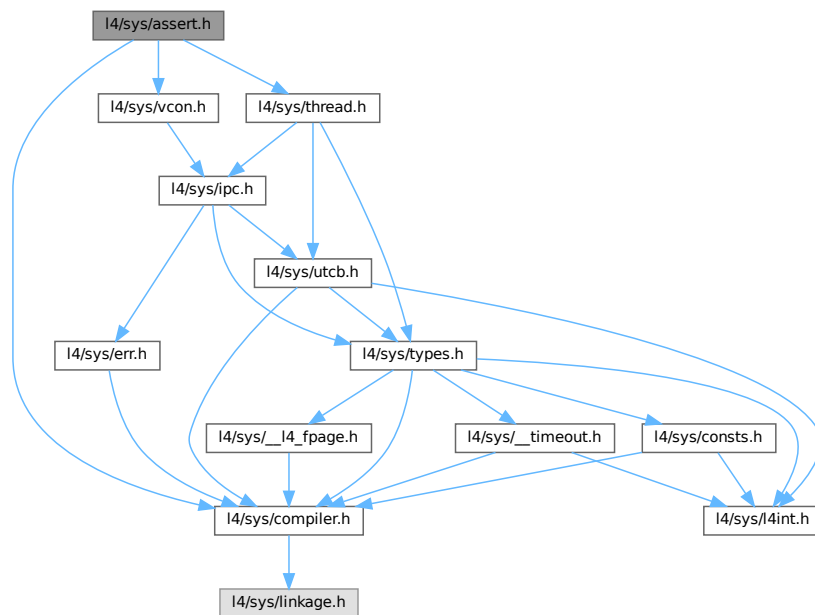
```
00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00006 /*
00007 * (c) 2008-2010 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013
00014 #pragma once
00015
00016 #include <l4/sys/vm.h>
00017 #include <l4/sys/task>
00018
00019 namespace L4 {
00020
00021 class Vm : public Kobject_t<Vm, Task, L4_PROTO_VM>
00022 {
00023 protected:
00024 Vm();
00025
00026 private:
00027 Vm(Vm const &);
00028 void operator = (Vm const &);
00029 };
00030
00031 };
```

## 17.624 l4/sys/assert.h File Reference

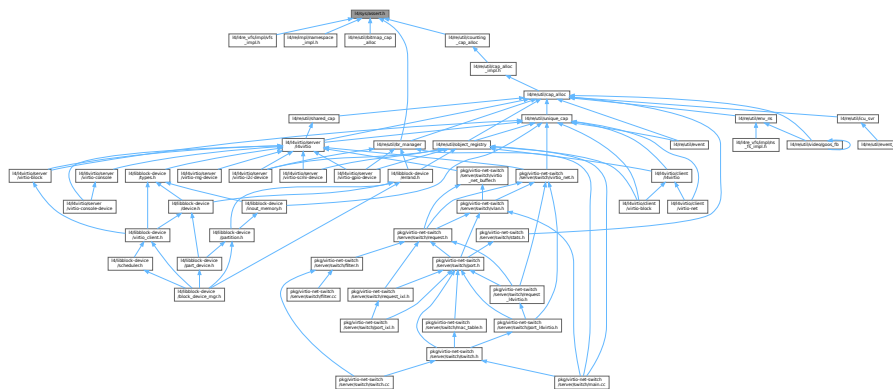
Low-level assert implementation.

```
#include <l4/sys/compiler.h>
#include <l4/sys/thread.h>
#include <l4/sys/vcon.h>
```

Include dependency graph for assert.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define l4_assert(expr)`  
*Low-level assert.*

## 17.624.1 Detailed Description

Low-level assert implementation.

Definition in file [assert.h](#).

## 17.624.2 Macro Definition Documentation

### 17.624.2.1 l4\_assert

```
#define l4_assert(
 expr)
```

#### Value:

```
l4_assert_fn(!(expr), __FILE__ ":" L4_stringify(__LINE__) ": Assertion \"" \
 L4_stringify(expr) "\" failed.\n")
```

Low-level assert.

#### Parameters

|             |                                              |
|-------------|----------------------------------------------|
| <i>expr</i> | Expression to be evaluate for the assertion. |
|-------------|----------------------------------------------|

This assertion is a low-level implementation that directly uses kernel primitives. Only use `l4_assert()` when the standard `assert()` functionality is not available.

Definition at line 32 of file [assert.h](#).

Referenced by [L4Re::Util::Cap\\_alloc\\_base::free\(\)](#), [L4Re::Util::Counting\\_cap\\_alloc< COUNTERTYPE, Dbg >::free\(\)](#), [L4Re::Util::Counting\\_cap\\_alloc< COUNTERTYPE, Dbg >::release\(\)](#), [L4Re::Util::Br\\_manager::set\\_rcv\\_cap\\_flags\(\)](#), and [Block\\_device::Device\\_mgr< DEV, FACTORY, SCHEDULER >::shutdown\\_event\(\)](#).

## 17.625 assert.h

[Go to the documentation of this file.](#)

```
00001
00005 /*
00006 * (c) 2015 Adam Lackorzynski <adam@l4re.org>
00007 *
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010 #pragma once
00011
00012 #ifndef NDEBUG
00013
00014 #define l4_assert(x) do { } while (0)
00015 #define l4_check(x) do { (void)(x); } while (0)
00016
00017 #else
00018
00019 #include <l4/sys/compiler.h>
00020 #include <l4/sys/thread.h>
00021 #include <l4/sys/vcon.h>
00022
00032 #define l4_assert(expr) \
00033 l4_assert_fn(!(expr), __FILE__ ":" L4_stringify(__LINE__) ": Assertion \"" \
00034 L4_stringify(expr) "\" failed.\n")
00035
00036 #define l4_check(expr) l4_assert(expr)
00037
00041 L4_ALWAYS_INLINE
00042 void l4_assert_fn(unsigned expr, const char *text) L4_NOTHROW;
00043
00047 L4_INLINE L4_NORETURN
00048 void l4_assert_abort(const char *text) L4_NOTHROW;
00049
00050
00051 /* IMPLEMENTATION -----*/
00052
00053 L4_INLINE L4_NORETURN
00054 void l4_assert_abort(const char *text) L4_NOTHROW
```



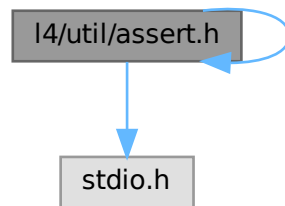
```
00055 {
00056 l4_vcon_write(L4_BASE_LOG_CAP, text, __builtin_strlen(text));
00057 for (;;)
00058 l4_thread_ex_regs(L4_INVALID_CAP, ~0UL, ~0UL,
00059 L4_THREAD_EX_REGS_TRIGGER_EXCEPTION);
00060 }
00061
00062 L4_ALWAYS_INLINE
00063 void l4_assert_fn(unsigned expr, const char *text) L4_NOTHROW
00064 {
00065 if (L4_LIKELY(expr))
00066 return;
00067 l4_assert_abort(text);
00068 }
00070
00071 #endif /* NDEBUG */
```

## 17.626 l4/util/assert.h File Reference

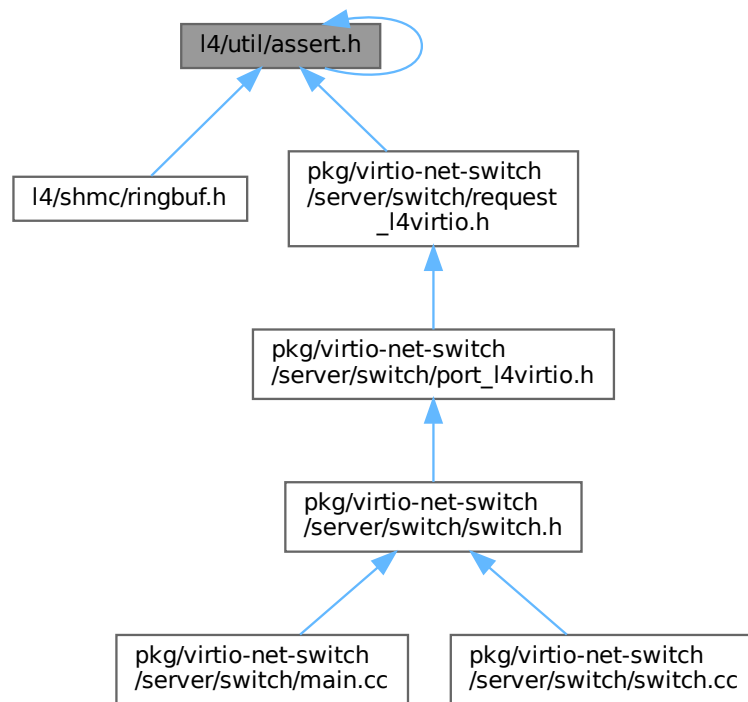
Some useful assert-style macros.

```
#include <stdio.h>
#include <assert.h>
```

Include dependency graph for assert.h:



This graph shows which files directly or indirectly include this file:



### 17.626.1 Detailed Description

Some useful assert-style macros.

#### Date

09/2009

#### Author

Bjoern Doebel [doebel@tudos.org](mailto:doebel@tudos.org)

Definition in file [assert.h](#).

## 17.627 assert.h

[Go to the documentation of this file.](#)

```

00001 /*****
00009 */
00010 * (c) 2009 Author(s)
00011 * economic rights: Technische Universität Dresden (Germany)
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */

```

```

00014
00015 /*****
00016 #pragma once
00017
00018 #ifndef NDEBUG
00019
00020 #define DO_NOTHING do {} while (0)
00021 #define ASSERT_ASSERT(x) DO_NOTHING
00022 #define ASSERT_VALID(c) DO_NOTHING
00023 #define ASSERT_EQUAL(a,b) DO_NOTHING
00024 #define ASSERT_NOT_EQUAL(a,b) DO_NOTHING
00025 #define ASSERT_LOWER_EQ(a,b) DO_NOTHING
00026 #define ASSERT_GREATER_EQ(a,b) DO_NOTHING
00027 #define ASSERT_BETWEEN(a,b,c) DO_NOTHING
00028 #define ASSERT_IPC_OK(i) DO_NOTHING
00029 #define ASSERT_OK(e) do { (void)e; } while (0)
00030 #define ASSERT_NOT_NULL(p) DO_NOTHING
00031 #ifndef assert
00032 #define assert(cond) DO_NOTHING
00033 #endif
00034
00035 #else // NDEBUG
00036
00037 #ifndef ASSERT_PRINTF
00038 #include <stdio.h>
00039 #define ASSERT_PRINTF printf
00040 #endif
00041 #ifndef ASSERT_ASSERT
00042 #include <assert.h>
00043 #define ASSERT_ASSERT(x) assert(x)
00044 #endif
00045
00046 #define ASSERT_VALID(cap) \
00047 do { \
00048 typeof(cap) _cap = cap; \
00049 if (!l4_is_invalid_cap(_cap)) { \
00050 ASSERT_PRINTF("%s: Cap invalid.\n", __func__); \
00051 ASSERT_ASSERT(!l4_is_invalid_cap(_cap)); \
00052 } \
00053 } while (0)
00054
00055 #define ASSERT_EQUAL(a, b) \
00056 do { \
00057 typeof(a) _a = a; \
00058 typeof(b) _b = b; \
00059 if (_a != _b) { \
00060 ASSERT_PRINTF("%s:\n", __func__); \
00061 ASSERT_PRINTF(" "#a" (%lx) != "#b" (%lx)\n", (unsigned long)_a, (unsigned long)_b); \
00062 ASSERT_ASSERT(_a == _b); \
00063 } \
00064 } while (0)
00065
00066 #define ASSERT_NOT_EQUAL(a, b) \
00067 do { \
00068 typeof(a) _a = a; \
00069 typeof(b) _b = b; \
00070 if (_a == _b) { \
00071 ASSERT_PRINTF("%s:\n", __func__); \
00072 ASSERT_PRINTF(" "#a" (%lx) == "#b" (%lx)\n", (unsigned long)_a, (unsigned long)_b); \
00073 ASSERT_ASSERT(_a != _b); \
00074 } \
00075 } while (0)
00076
00077 #define ASSERT_LOWER_EQ(val, max) \
00078 do { \
00079 typeof(val) _val = val; \
00080 typeof(max) _max = max; \
00081 if (_val > _max) { \
00082 ASSERT_PRINTF("%s:\n", __func__); \
00083 ASSERT_PRINTF(" "#val" (%lx) > "#max" (%lx)\n", (unsigned long)_val, (unsigned long)_max); \
00084 ASSERT_ASSERT(_val <= _max); \
00085 } \
00086 } while (0)
00087
00088 #define ASSERT_GREATER_EQ(val, min) \
00089 do { \
00090 typeof(val) _val = val; \
00091 typeof(min) _min = min; \
00092 if (_val < _min) { \
00093 ASSERT_PRINTF("%s:\n", __func__); \
00094 ASSERT_PRINTF(" "#val" (%lx) < "#min" (%lx)\n", (unsigned long)_val, (unsigned long)_min); \
00095 ASSERT_ASSERT(_val >= _min); \
00096 } \
00097 } while (0)
00098
00099
00100

```

```

00101 } while (0)
00102
00103
00104 #define ASSERT_BETWEEN(val, min, max) \
00105 ASSERT_LOWER_EQ((val), (max)); \
00106 ASSERT_GREATER_EQ((val), (min));
00107
00108
00109 #define ASSERT_IPC_OK(msgtag) \
00110 do { \
00111 int _r = l4_ipc_error(msgtag, l4_utcb()); \
00112 if (_r) { \
00113 ASSERT_PRINTF("%s: IPC Error: %lx\n", __func__, _r); \
00114 ASSERT_ASSERT(_r == 0); \
00115 } \
00116 } while (0)
00117
00118 #define ASSERT_OK(val) ASSERT_EQUAL((val), 0)
00119 #define ASSERT_NOT_NULL(ptr) ASSERT_NOT_EQUAL((ptr), (void *)0)
00120
00121 #endif // NDEBUG

```

## 17.628 atomic.h

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #pragma once
00014
00015 #include <l4/sys/compiler.h>
00016
00017 L4_BEGIN_DECLS
00018
00019 long int
00020 l4_atomic_add(volatile long int* mem, long int offset) L4_NOTHROW L4_LONG_CALL;
00021
00022 long int
00023 l4_atomic_xchg(volatile long int* mem, long int newval) L4_NOTHROW L4_LONG_CALL;
00024
00025 long int
00026 l4_atomic_cmpxchg(volatile long int* mem, long int oldval, long int newval) L4_NOTHROW L4_LONG_CALL;
00027
00028 L4_END_DECLS

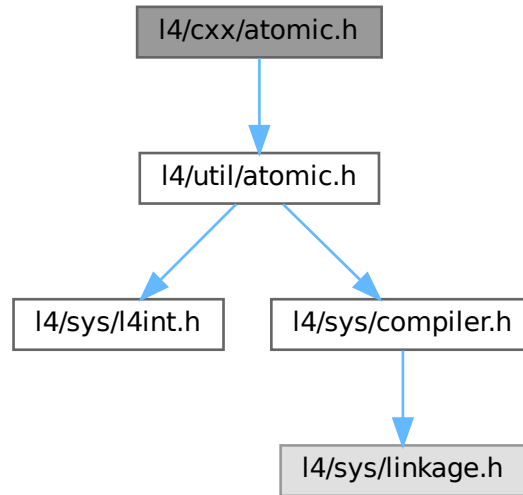
```

## 17.629 l4/cxx/atomic.h File Reference

Atomic template.

```
#include <l4/util/atomic.h>
```

Include dependency graph for atomic.h:



## Namespaces

- namespace `L4`  
*L4 low-level kernel interface.*

## 17.629.1 Detailed Description

Atomic template.

Definition in file [atomic.h](#).

## 17.630 atomic.h

[Go to the documentation of this file.](#)

```

00001
00002 /*
00003 * (c) 2004-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00004 * Alexander Warg <warg@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 #pragma once
00010 #include <l4/util/atomic.h>
00011
00012 extern "C" void ____error_compare_and_swap_does_not_support_3_bytes____();
00013 extern "C" void ____error_compare_and_swap_does_not_support_more_than_4_bytes____();
00014
00015 namespace L4
00016 {

```

```

00021 template< typename X >
00022 inline int compare_and_swap(X volatile *dst, X old_val, X new_val)
00023 {
00024 switch (sizeof(X))
00025 {
00026 case 1:
00027 return l4util_cmpxchg8((l4_uint8_t volatile*)dst, old_val, new_val);
00028 case 2:
00029 return l4util_cmpxchg16((l4_uint16_t volatile *)dst, old_val, new_val);
00030 case 3: ____error_compare_and_swap_does_not_support_3_bytes____();
00031 case 4:
00032 return l4util_cmpxchg32((l4_uint32_t volatile*)dst, old_val, new_val);
00033 default:
00034 ____error_compare_and_swap_does_not_support_more_than_4_bytes____();
00035 }
00036 return 0;
00037 }
00038 }

```

## 17.631 l4/util/atomic.h File Reference

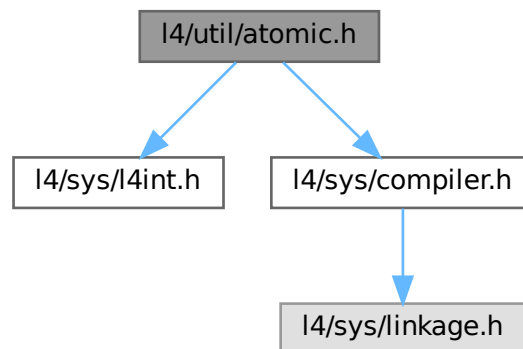
atomic operations header and generic implementations

```

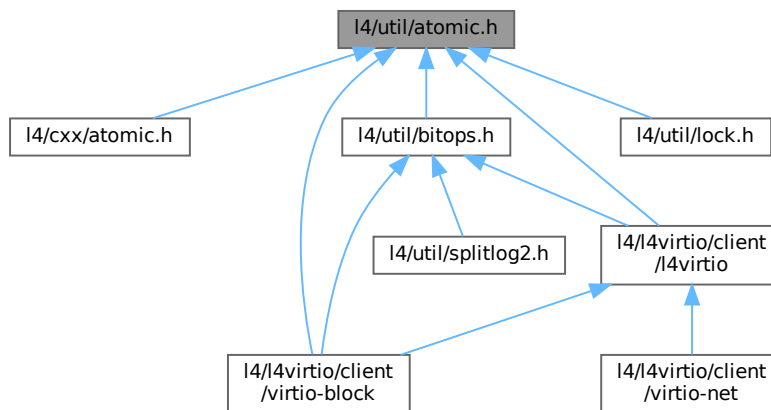
#include <l4/sys/l4int.h>
#include <l4/sys/compiler.h>

```

Include dependency graph for atomic.h:



This graph shows which files directly or indirectly include this file:



## Functions

- int [l4util\\_cmpxchg32](#) (volatile [l4\\_uint32\\_t](#) \*dest, [l4\\_uint32\\_t](#) cmp\_val, [l4\\_uint32\\_t](#) new\_val)  
*Atomic compare and exchange (32 bit version).*
- int [l4util\\_cmpxchg16](#) (volatile [l4\\_uint16\\_t](#) \*dest, [l4\\_uint16\\_t](#) cmp\_val, [l4\\_uint16\\_t](#) new\_val)  
*Atomic compare and exchange (16 bit version).*
- int [l4util\\_cmpxchg8](#) (volatile [l4\\_uint8\\_t](#) \*dest, [l4\\_uint8\\_t](#) cmp\_val, [l4\\_uint8\\_t](#) new\_val)  
*Atomic compare and exchange (8 bit version).*
- int [l4util\\_cmpxchg](#) (volatile [l4\\_umword\\_t](#) \*dest, [l4\\_umword\\_t](#) cmp\_val, [l4\\_umword\\_t](#) new\_val)  
*Atomic compare and exchange (machine wide fields).*
- [l4\\_uint32\\_t](#) [l4util\\_xchg32](#) (volatile [l4\\_uint32\\_t](#) \*dest, [l4\\_uint32\\_t](#) val)  
*Atomic exchange (32 bit version).*
- [l4\\_uint16\\_t](#) [l4util\\_xchg16](#) (volatile [l4\\_uint16\\_t](#) \*dest, [l4\\_uint16\\_t](#) val)  
*Atomic exchange (16 bit version).*
- [l4\\_uint8\\_t](#) [l4util\\_xchg8](#) (volatile [l4\\_uint8\\_t](#) \*dest, [l4\\_uint8\\_t](#) val)  
*Atomic exchange (8 bit version).*
- [l4\\_umword\\_t](#) [l4util\\_xchg](#) (volatile [l4\\_umword\\_t](#) \*dest, [l4\\_umword\\_t](#) val)  
*Atomic exchange (machine wide fields).*
- void [l4util\\_atomic\\_add](#) (volatile long \*dest, long val)  
*Atomic add.*
- void [l4util\\_atomic\\_inc](#) (volatile long \*dest)  
*Atomic increment.*

## Atomic add/sub/and/or (8,16,32 bit version) without result

- void [l4util\\_add8](#) (volatile [l4\\_uint8\\_t](#) \*dest, [l4\\_uint8\\_t](#) val)
- void [l4util\\_add16](#) (volatile [l4\\_uint16\\_t](#) \*dest, [l4\\_uint16\\_t](#) val)
- void [l4util\\_add32](#) (volatile [l4\\_uint32\\_t](#) \*dest, [l4\\_uint32\\_t](#) val)
- void [l4util\\_sub8](#) (volatile [l4\\_uint8\\_t](#) \*dest, [l4\\_uint8\\_t](#) val)
- void [l4util\\_sub16](#) (volatile [l4\\_uint16\\_t](#) \*dest, [l4\\_uint16\\_t](#) val)
- void [l4util\\_sub32](#) (volatile [l4\\_uint32\\_t](#) \*dest, [l4\\_uint32\\_t](#) val)
- void [l4util\\_and8](#) (volatile [l4\\_uint8\\_t](#) \*dest, [l4\\_uint8\\_t](#) val)
- void [l4util\\_and16](#) (volatile [l4\\_uint16\\_t](#) \*dest, [l4\\_uint16\\_t](#) val)

- void [l4util\\_and32](#) (volatile [l4\\_uint32\\_t](#) \*dest, [l4\\_uint32\\_t](#) val)
- void [l4util\\_or8](#) (volatile [l4\\_uint8\\_t](#) \*dest, [l4\\_uint8\\_t](#) val)
- void [l4util\\_or16](#) (volatile [l4\\_uint16\\_t](#) \*dest, [l4\\_uint16\\_t](#) val)
- void [l4util\\_or32](#) (volatile [l4\\_uint32\\_t](#) \*dest, [l4\\_uint32\\_t](#) val)

#### Atomic add/sub/and/or operations (8,16,32 bit) with result

- [l4\\_uint8\\_t l4util\\_add8\\_res](#) (volatile [l4\\_uint8\\_t](#) \*dest, [l4\\_uint8\\_t](#) val)
- [l4\\_uint16\\_t l4util\\_add16\\_res](#) (volatile [l4\\_uint16\\_t](#) \*dest, [l4\\_uint16\\_t](#) val)
- [l4\\_uint32\\_t l4util\\_add32\\_res](#) (volatile [l4\\_uint32\\_t](#) \*dest, [l4\\_uint32\\_t](#) val)
- [l4\\_uint8\\_t l4util\\_sub8\\_res](#) (volatile [l4\\_uint8\\_t](#) \*dest, [l4\\_uint8\\_t](#) val)
- [l4\\_uint16\\_t l4util\\_sub16\\_res](#) (volatile [l4\\_uint16\\_t](#) \*dest, [l4\\_uint16\\_t](#) val)
- [l4\\_uint32\\_t l4util\\_sub32\\_res](#) (volatile [l4\\_uint32\\_t](#) \*dest, [l4\\_uint32\\_t](#) val)
- [l4\\_uint8\\_t l4util\\_and8\\_res](#) (volatile [l4\\_uint8\\_t](#) \*dest, [l4\\_uint8\\_t](#) val)
- [l4\\_uint16\\_t l4util\\_and16\\_res](#) (volatile [l4\\_uint16\\_t](#) \*dest, [l4\\_uint16\\_t](#) val)
- [l4\\_uint32\\_t l4util\\_and32\\_res](#) (volatile [l4\\_uint32\\_t](#) \*dest, [l4\\_uint32\\_t](#) val)
- [l4\\_uint8\\_t l4util\\_or8\\_res](#) (volatile [l4\\_uint8\\_t](#) \*dest, [l4\\_uint8\\_t](#) val)
- [l4\\_uint16\\_t l4util\\_or16\\_res](#) (volatile [l4\\_uint16\\_t](#) \*dest, [l4\\_uint16\\_t](#) val)
- [l4\\_uint32\\_t l4util\\_or32\\_res](#) (volatile [l4\\_uint32\\_t](#) \*dest, [l4\\_uint32\\_t](#) val)

#### Atomic inc/dec (8,16,32 bit) without result

- void [l4util\\_inc8](#) (volatile [l4\\_uint8\\_t](#) \*dest)
- void [l4util\\_inc16](#) (volatile [l4\\_uint16\\_t](#) \*dest)
- void [l4util\\_inc32](#) (volatile [l4\\_uint32\\_t](#) \*dest)
- void [l4util\\_dec8](#) (volatile [l4\\_uint8\\_t](#) \*dest)
- void [l4util\\_dec16](#) (volatile [l4\\_uint16\\_t](#) \*dest)
- void [l4util\\_dec32](#) (volatile [l4\\_uint32\\_t](#) \*dest)

#### Atomic inc/dec (8,16,32 bit) with result

- [l4\\_uint8\\_t l4util\\_inc8\\_res](#) (volatile [l4\\_uint8\\_t](#) \*dest)
- [l4\\_uint16\\_t l4util\\_inc16\\_res](#) (volatile [l4\\_uint16\\_t](#) \*dest)
- [l4\\_uint32\\_t l4util\\_inc32\\_res](#) (volatile [l4\\_uint32\\_t](#) \*dest)
- [l4\\_uint8\\_t l4util\\_dec8\\_res](#) (volatile [l4\\_uint8\\_t](#) \*dest)
- [l4\\_uint16\\_t l4util\\_dec16\\_res](#) (volatile [l4\\_uint16\\_t](#) \*dest)
- [l4\\_uint32\\_t l4util\\_dec32\\_res](#) (volatile [l4\\_uint32\\_t](#) \*dest)

## 17.631.1 Detailed Description

atomic operations header and generic implementations

Date

10/20/2000

Author

Lars Reuther [reuther@os.inf.tu-dresden.de](mailto:reuther@os.inf.tu-dresden.de), Jork Loeser [jork@os.inf.tu-dresden.de](mailto:jork@os.inf.tu-dresden.de)

Definition in file [atomic.h](#).



## 17.632 atomic.h

[Go to the documentation of this file.](#)

```

00001 /*****
00010 */
00011 * (c) 2000-2009 Author(s)
00012 * economic rights: Technische Universität Dresden (Germany)
00013 * License: see LICENSE.spdx (in this directory or the directories above)
00014 */
00015
00016 /*****
00017 #ifndef __L4UTIL__INCLUDE__ATOMIC_H__
00018 #define __L4UTIL__INCLUDE__ATOMIC_H__
00019
00020 #include <l4/sys/l4int.h>
00021 #include <l4/sys/compiler.h>
00022
00023 /*****
00024 *** Prototypes
00025 *****/
00026
00027 L4_BEGIN_DECLS
00028
00029
00030
00031 #if __SIZEOF_LONG__ == 8
00032
00033
00034
00035 L4_INLINE int
00036 l4util_cmpxchg64(volatile l4_uint64_t * dest,
00037 l4_uint64_t cmp_val, l4_uint64_t new_val);
00038
00039 #endif
00040
00041 L4_INLINE int
00042 l4util_cmpxchg32(volatile l4_uint32_t * dest,
00043 l4_uint32_t cmp_val, l4_uint32_t new_val);
00044
00045 L4_INLINE int
00046 l4util_cmpxchg16(volatile l4_uint16_t * dest,
00047 l4_uint16_t cmp_val, l4_uint16_t new_val);
00048
00049 L4_INLINE int
00050 l4util_cmpxchg8(volatile l4_uint8_t * dest,
00051 l4_uint8_t cmp_val, l4_uint8_t new_val);
00052
00053 L4_INLINE int
00054 l4util_cmpxchg(volatile l4_umword_t * dest,
00055 l4_umword_t cmp_val, l4_umword_t new_val);
00056
00057 L4_INLINE l4_uint32_t
00058 l4util_xchg32(volatile l4_uint32_t * dest, l4_uint32_t val);
00059
00060 L4_INLINE l4_uint16_t
00061 l4util_xchg16(volatile l4_uint16_t * dest, l4_uint16_t val);
00062
00063 L4_INLINE l4_uint8_t
00064 l4util_xchg8(volatile l4_uint8_t * dest, l4_uint8_t val);
00065
00066 L4_INLINE l4_umword_t
00067 l4util_xchg(volatile l4_umword_t * dest, l4_umword_t val);
00068
00069
00070 L4_INLINE void
00071 l4util_add8(volatile l4_uint8_t *dest, l4_uint8_t val);
00072 L4_INLINE void
00073 l4util_add16(volatile l4_uint16_t *dest, l4_uint16_t val);
00074 L4_INLINE void
00075 l4util_add32(volatile l4_uint32_t *dest, l4_uint32_t val);
00076 L4_INLINE void
00077 l4util_sub8(volatile l4_uint8_t *dest, l4_uint8_t val);
00078 L4_INLINE void
00079 l4util_sub16(volatile l4_uint16_t *dest, l4_uint16_t val);
00080 L4_INLINE void
00081 l4util_sub32(volatile l4_uint32_t *dest, l4_uint32_t val);
00082 L4_INLINE void
00083 l4util_and8(volatile l4_uint8_t *dest, l4_uint8_t val);
00084 L4_INLINE void
00085 l4util_and16(volatile l4_uint16_t *dest, l4_uint16_t val);
00086 L4_INLINE void
00087 l4util_and32(volatile l4_uint32_t *dest, l4_uint32_t val);
00088 L4_INLINE void
00089 l4util_or8(volatile l4_uint8_t *dest, l4_uint8_t val);
00090 L4_INLINE void
00091 l4util_or16(volatile l4_uint16_t *dest, l4_uint16_t val);
00092 L4_INLINE void
00093 l4util_or32(volatile l4_uint32_t *dest, l4_uint32_t val);

```

```

00220
00222
00229 L4_INLINE l4_uint8_t
00230 l4util_add8_res(volatile l4_uint8_t *dest, l4_uint8_t val);
00232 L4_INLINE l4_uint16_t
00233 l4util_add16_res(volatile l4_uint16_t *dest, l4_uint16_t val);
00235 L4_INLINE l4_uint32_t
00236 l4util_add32_res(volatile l4_uint32_t *dest, l4_uint32_t val);
00238 L4_INLINE l4_uint8_t
00239 l4util_sub8_res(volatile l4_uint8_t *dest, l4_uint8_t val);
00241 L4_INLINE l4_uint16_t
00242 l4util_sub16_res(volatile l4_uint16_t *dest, l4_uint16_t val);
00244 L4_INLINE l4_uint32_t
00245 l4util_sub32_res(volatile l4_uint32_t *dest, l4_uint32_t val);
00247 L4_INLINE l4_uint8_t
00248 l4util_and8_res(volatile l4_uint8_t *dest, l4_uint8_t val);
00250 L4_INLINE l4_uint16_t
00251 l4util_and16_res(volatile l4_uint16_t *dest, l4_uint16_t val);
00253 L4_INLINE l4_uint32_t
00254 l4util_and32_res(volatile l4_uint32_t *dest, l4_uint32_t val);
00256 L4_INLINE l4_uint8_t
00257 l4util_or8_res(volatile l4_uint8_t *dest, l4_uint8_t val);
00259 L4_INLINE l4_uint16_t
00260 l4util_or16_res(volatile l4_uint16_t *dest, l4_uint16_t val);
00262 L4_INLINE l4_uint32_t
00263 l4util_or32_res(volatile l4_uint32_t *dest, l4_uint32_t val);
00265
00267
00272 L4_INLINE void
00273 l4util_inc8(volatile l4_uint8_t *dest);
00275 L4_INLINE void
00276 l4util_inc16(volatile l4_uint16_t *dest);
00278 L4_INLINE void
00279 l4util_inc32(volatile l4_uint32_t *dest);
00281 L4_INLINE void
00282 l4util_dec8(volatile l4_uint8_t *dest);
00284 L4_INLINE void
00285 l4util_dec16(volatile l4_uint16_t *dest);
00287 L4_INLINE void
00288 l4util_dec32(volatile l4_uint32_t *dest);
00290
00292
00298 L4_INLINE l4_uint8_t
00299 l4util_inc8_res(volatile l4_uint8_t *dest);
00301 L4_INLINE l4_uint16_t
00302 l4util_inc16_res(volatile l4_uint16_t *dest);
00304 L4_INLINE l4_uint32_t
00305 l4util_inc32_res(volatile l4_uint32_t *dest);
00307 L4_INLINE l4_uint8_t
00308 l4util_dec8_res(volatile l4_uint8_t *dest);
00310 L4_INLINE l4_uint16_t
00311 l4util_dec16_res(volatile l4_uint16_t *dest);
00313 L4_INLINE l4_uint32_t
00314 l4util_dec32_res(volatile l4_uint32_t *dest);
00316
00324 L4_INLINE void
00325 l4util_atomic_add(volatile long *dest, long val);
00326
00333 L4_INLINE void
00334 l4util_atomic_inc(volatile long *dest);
00335
00336 L4_END_DECLS
00337
00338 /*****
00339 * IMPLEMENTATION *
00340 *****/
00341
00342 #if __SIZEOF_LONG__ == 8
00343
00344 L4_INLINE int
00345 l4util_cmpxchg64(volatile l4_uint64_t * dest,
00346 l4_uint64_t cmp_val, l4_uint64_t new_val)
00347 {
00348 return __atomic_compare_exchange_n(dest, &cmp_val, new_val, 0,
00349 __ATOMIC_SEQ_CST, __ATOMIC_SEQ_CST);
00350 }
00351
00352 #endif
00353
00354 L4_INLINE int
00355 l4util_cmpxchg32(volatile l4_uint32_t * dest,
00356 l4_uint32_t cmp_val, l4_uint32_t new_val)
00357 {
00358 return __atomic_compare_exchange_n(dest, &cmp_val, new_val, 0,
00359 __ATOMIC_SEQ_CST, __ATOMIC_SEQ_CST);
00360 }
00361

```

```

00362 L4_INLINE int
00363 l4util_cmpxchg16(volatile l4_uint16_t * dest,
00364 l4_uint16_t cmp_val, l4_uint16_t new_val)
00365 {
00366 return __atomic_compare_exchange_n(dest, &cmp_val, new_val, 0,
00367 __ATOMIC_SEQ_CST, __ATOMIC_SEQ_CST);
00368 }
00369
00370 L4_INLINE int
00371 l4util_cmpxchg8(volatile l4_uint8_t * dest,
00372 l4_uint8_t cmp_val, l4_uint8_t new_val)
00373 {
00374 return __atomic_compare_exchange_n(dest, &cmp_val, new_val, 0,
00375 __ATOMIC_SEQ_CST, __ATOMIC_SEQ_CST);
00376 }
00377
00378 L4_INLINE int
00379 l4util_cmpxchg(volatile l4_umword_t * dest,
00380 l4_umword_t cmp_val, l4_umword_t new_val)
00381 {
00382 return __atomic_compare_exchange_n(dest, &cmp_val, new_val, 0,
00383 __ATOMIC_SEQ_CST, __ATOMIC_SEQ_CST);
00384 }
00385
00386 L4_INLINE l4_uint32_t
00387 l4util_xchg32(volatile l4_uint32_t * dest, l4_uint32_t val)
00388 {
00389 return __atomic_exchange_n(dest, val, __ATOMIC_SEQ_CST);
00390 }
00391
00392 L4_INLINE l4_uint16_t
00393 l4util_xchg16(volatile l4_uint16_t * dest, l4_uint16_t val)
00394 {
00395 return __atomic_exchange_n(dest, val, __ATOMIC_SEQ_CST);
00396 }
00397
00398 L4_INLINE l4_uint8_t
00399 l4util_xchg8(volatile l4_uint8_t * dest, l4_uint8_t val)
00400 {
00401 return __atomic_exchange_n(dest, val, __ATOMIC_SEQ_CST);
00402 }
00403
00404 L4_INLINE l4_umword_t
00405 l4util_xchg(volatile l4_umword_t * dest, l4_umword_t val)
00406 {
00407 return __atomic_exchange_n(dest, val, __ATOMIC_SEQ_CST);
00408 }
00409
00410 L4_INLINE void
00411 l4util_inc8(volatile l4_uint8_t *dest)
00412 { __atomic_fetch_add(dest, 1, __ATOMIC_SEQ_CST); }
00413
00414 L4_INLINE void
00415 l4util_inc16(volatile l4_uint16_t *dest)
00416 { __atomic_fetch_add(dest, 1, __ATOMIC_SEQ_CST); }
00417
00418 L4_INLINE void
00419 l4util_inc32(volatile l4_uint32_t *dest)
00420 { __atomic_fetch_add(dest, 1, __ATOMIC_SEQ_CST); }
00421
00422 L4_INLINE void
00423 l4util_atomic_inc(volatile long *dest)
00424 { __atomic_fetch_add(dest, 1, __ATOMIC_SEQ_CST); }
00425
00426 L4_INLINE void
00427 l4util_dec8(volatile l4_uint8_t *dest)
00428 { __atomic_fetch_sub(dest, 1, __ATOMIC_SEQ_CST); }
00429
00430 L4_INLINE void
00431 l4util_dec16(volatile l4_uint16_t *dest)
00432 { __atomic_fetch_sub(dest, 1, __ATOMIC_SEQ_CST); }
00433
00434 L4_INLINE void
00435 l4util_dec32(volatile l4_uint32_t *dest)
00436 { __atomic_fetch_sub(dest, 1, __ATOMIC_SEQ_CST); }
00437
00438
00439 L4_INLINE l4_uint8_t
00440 l4util_inc8_res(volatile l4_uint8_t *dest)
00441 { return __atomic_add_fetch(dest, 1, __ATOMIC_SEQ_CST); }
00442
00443 L4_INLINE l4_uint16_t
00444 l4util_inc16_res(volatile l4_uint16_t *dest)
00445 { return __atomic_add_fetch(dest, 1, __ATOMIC_SEQ_CST); }
00446
00447 L4_INLINE l4_uint32_t
00448 l4util_inc32_res(volatile l4_uint32_t *dest)

```

```

00449 { return __atomic_add_fetch(dest, 1, __ATOMIC_SEQ_CST); }
00450
00451 L4_INLINE l4_uint8_t
00452 l4util_dec8_res(volatile l4_uint8_t *dest)
00453 { return __atomic_sub_fetch(dest, 1, __ATOMIC_SEQ_CST); }
00454
00455 L4_INLINE l4_uint16_t
00456 l4util_dec16_res(volatile l4_uint16_t *dest)
00457 { return __atomic_sub_fetch(dest, 1, __ATOMIC_SEQ_CST); }
00458
00459 L4_INLINE l4_uint32_t
00460 l4util_dec32_res(volatile l4_uint32_t *dest)
00461 { return __atomic_sub_fetch(dest, 1, __ATOMIC_SEQ_CST); }
00462
00463 L4_INLINE l4_umword_t
00464 l4util_dec_res(volatile l4_umword_t *dest)
00465 { return __atomic_sub_fetch(dest, 1, __ATOMIC_SEQ_CST); }
00466
00467 L4_INLINE void
00468 l4util_add8(volatile l4_uint8_t *dest, l4_uint8_t val)
00469 { __atomic_fetch_add(dest, val, __ATOMIC_SEQ_CST); }
00470
00471 L4_INLINE void
00472 l4util_add16(volatile l4_uint16_t *dest, l4_uint16_t val)
00473 { __atomic_fetch_add(dest, val, __ATOMIC_SEQ_CST); }
00474
00475 L4_INLINE void
00476 l4util_add32(volatile l4_uint32_t *dest, l4_uint32_t val)
00477 { __atomic_fetch_add(dest, val, __ATOMIC_SEQ_CST); }
00478
00479 L4_INLINE void
00480 l4util_atomic_add(volatile long *dest, long val)
00481 { __atomic_fetch_add(dest, val, __ATOMIC_SEQ_CST); }
00482
00483 L4_INLINE void
00484 l4util_sub8(volatile l4_uint8_t *dest, l4_uint8_t val)
00485 { __atomic_fetch_sub(dest, val, __ATOMIC_SEQ_CST); }
00486
00487 L4_INLINE void
00488 l4util_sub16(volatile l4_uint16_t *dest, l4_uint16_t val)
00489 { __atomic_fetch_sub(dest, val, __ATOMIC_SEQ_CST); }
00490
00491 L4_INLINE void
00492 l4util_sub32(volatile l4_uint32_t *dest, l4_uint32_t val)
00493 { __atomic_fetch_sub(dest, val, __ATOMIC_SEQ_CST); }
00494
00495 L4_INLINE void
00496 l4util_and8(volatile l4_uint8_t *dest, l4_uint8_t val)
00497 { __atomic_fetch_and(dest, val, __ATOMIC_SEQ_CST); }
00498
00499 L4_INLINE void
00500 l4util_and16(volatile l4_uint16_t *dest, l4_uint16_t val)
00501 { __atomic_fetch_and(dest, val, __ATOMIC_SEQ_CST); }
00502
00503 L4_INLINE void
00504 l4util_and32(volatile l4_uint32_t *dest, l4_uint32_t val)
00505 { __atomic_fetch_and(dest, val, __ATOMIC_SEQ_CST); }
00506
00507 L4_INLINE void
00508 l4util_or8(volatile l4_uint8_t *dest, l4_uint8_t val)
00509 { __atomic_fetch_or(dest, val, __ATOMIC_SEQ_CST); }
00510
00511 L4_INLINE void
00512 l4util_or16(volatile l4_uint16_t *dest, l4_uint16_t val)
00513 { __atomic_fetch_or(dest, val, __ATOMIC_SEQ_CST); }
00514
00515 L4_INLINE void
00516 l4util_or32(volatile l4_uint32_t *dest, l4_uint32_t val)
00517 { __atomic_fetch_or(dest, val, __ATOMIC_SEQ_CST); }
00518
00519 L4_INLINE l4_uint8_t
00520 l4util_add8_res(volatile l4_uint8_t *dest, l4_uint8_t val)
00521 { return __atomic_add_fetch(dest, val, __ATOMIC_SEQ_CST); }
00522
00523 L4_INLINE l4_uint16_t
00524 l4util_add16_res(volatile l4_uint16_t *dest, l4_uint16_t val)
00525 { return __atomic_add_fetch(dest, val, __ATOMIC_SEQ_CST); }
00526
00527 L4_INLINE l4_uint32_t
00528 l4util_add32_res(volatile l4_uint32_t *dest, l4_uint32_t val)
00529 { return __atomic_add_fetch(dest, val, __ATOMIC_SEQ_CST); }
00530
00531 L4_INLINE l4_uint8_t
00532 l4util_sub8_res(volatile l4_uint8_t *dest, l4_uint8_t val)
00533 { return __atomic_sub_fetch(dest, val, __ATOMIC_SEQ_CST); }
00534
00535 L4_INLINE l4_uint16_t

```

```

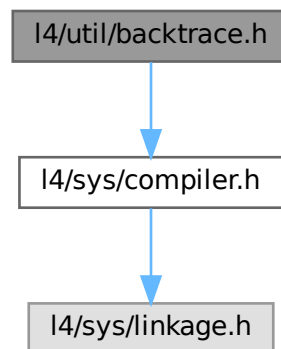
00536 l4util_sub16_res(volatile l4_uint16_t *dest, l4_uint16_t val)
00537 { return __atomic_sub_fetch(dest, val, __ATOMIC_SEQ_CST); }
00538
00539 L4_INLINE l4_uint32_t
00540 l4util_sub32_res(volatile l4_uint32_t *dest, l4_uint32_t val)
00541 { return __atomic_sub_fetch(dest, val, __ATOMIC_SEQ_CST); }
00542
00543 L4_INLINE l4_uint8_t
00544 l4util_and8_res(volatile l4_uint8_t *dest, l4_uint8_t val)
00545 { return __atomic_and_fetch(dest, val, __ATOMIC_SEQ_CST); }
00546
00547 L4_INLINE l4_uint16_t
00548 l4util_and16_res(volatile l4_uint16_t *dest, l4_uint16_t val)
00549 { return __atomic_and_fetch(dest, val, __ATOMIC_SEQ_CST); }
00550
00551 L4_INLINE l4_uint32_t
00552 l4util_and32_res(volatile l4_uint32_t *dest, l4_uint32_t val)
00553 { return __atomic_and_fetch(dest, val, __ATOMIC_SEQ_CST); }
00554
00555 L4_INLINE l4_uint8_t
00556 l4util_or8_res(volatile l4_uint8_t *dest, l4_uint8_t val)
00557 { return __atomic_or_fetch(dest, val, __ATOMIC_SEQ_CST); }
00558
00559 L4_INLINE l4_uint16_t
00560 l4util_or16_res(volatile l4_uint16_t *dest, l4_uint16_t val)
00561 { return __atomic_or_fetch(dest, val, __ATOMIC_SEQ_CST); }
00562
00563 L4_INLINE l4_uint32_t
00564 l4util_or32_res(volatile l4_uint32_t *dest, l4_uint32_t val)
00565 { return __atomic_or_fetch(dest, val, __ATOMIC_SEQ_CST); }
00566
00567 #endif /* ! __L4UTIL__INCLUDE__ATOMIC_H__ */

```

## 17.633 l4/util/backtrace.h File Reference

Backtrace.

#include <l4/sys/compiler.h>  
 Include dependency graph for backtrace.h:





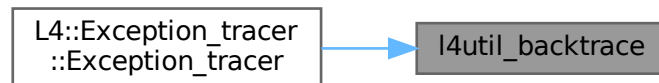
**Returns**

Number of entries

References [L4\\_END\\_DECLS](#).

Referenced by [L4::Exception\\_tracer::Exception\\_tracer\(\)](#).

Here is the caller graph for this function:

**17.634 backtrace.h**

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #pragma once
00012
00013 #include <l4/sys/compiler.h>
00014
00015 L4_BEGIN_DECLS
00016
00024 int l4util_backtrace(void **pc_array, int max_len);
00025
00026 L4_END_DECLS

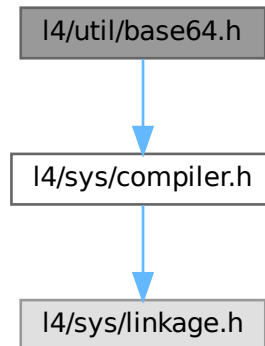
```

**17.635 l4/util/base64.h File Reference**

base 64 encoding and decoding functions adapted from Bob Trower 08/04/01

```
#include <l4/sys/compiler.h>
```

Include dependency graph for base64.h:



## Functions

- void **base64\_encode** (const char \*infile, unsigned int in\_size, char \*\*outfile)  
*base-64-encode string infile*
- void **base64\_decode** (const char \*infile, unsigned int in\_size, char \*\*outfile)  
*decode base-64-encoded string infile*

### 17.635.1 Detailed Description

base 64 encoding and decoding functions adapted from Bob Trower 08/04/01

#### Date

04/26/2002

#### Author

Joerg Nothnagel [jn6@os.inf.tu-dresden.de](mailto:jn6@os.inf.tu-dresden.de)

Definition in file [base64.h](#).



## 17.636 base64.h

[Go to the documentation of this file.](#)

```

00001
00009 /*
00010 * (c) 2008-2009 Author(s)
00011 * economic rights: Technische Universität Dresden (Germany)
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014
00015 #ifndef B64_EN_DECODE
00016 #define B64_EN_DECODE
00017
00018 #include <l4/sys/compiler.h>
00019
00020 L4_BEGIN_DECLS
00021
00027
00038 L4_CV void base64_encode(const char *infile, unsigned int in_size, char **outfile);
00039
00050 L4_CV void base64_decode(const char *infile, unsigned int in_size, char **outfile);
00051
00052 L4_END_DECLS
00053
00055 #endif //B64_EN_DECODE

```

## 17.637 l4/util/bitops.h File Reference

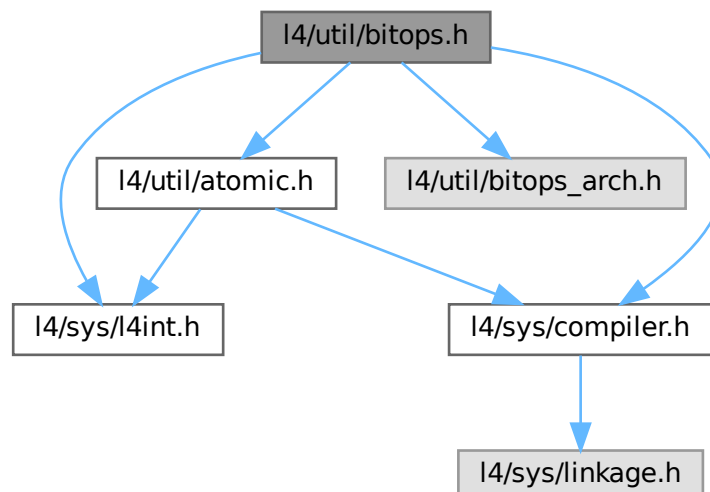
bit manipulation functions

```

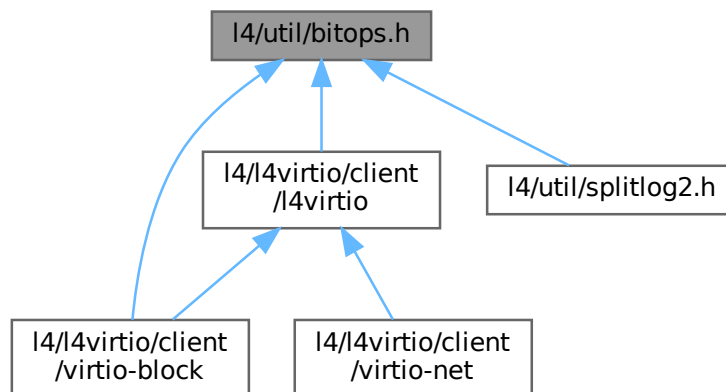
#include <l4/sys/l4int.h>
#include <l4/sys/compiler.h>
#include <l4/util/bitops_arch.h>
#include <l4/util/atomic.h>

```

Include dependency graph for bitops.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define I4util_test_and_clear_bit(b, dest)`  
*define some more usual names*

## Functions

- void `I4util_set_bit` (int b, volatile `I4_umword_t` \*dest)  
*Set bit in memory.*
- void `I4util_clear_bit` (int b, volatile `I4_umword_t` \*dest)  
*Clear bit in memory.*
- void `I4util_complement_bit` (int b, volatile `I4_umword_t` \*dest)  
*Complement bit in memory.*
- int `I4util_test_bit` (int b, const volatile `I4_umword_t` \*dest)  
*Test bit (return value of bit).*
- int `I4util_bts` (int b, volatile `I4_umword_t` \*dest)  
*Bit test and set.*
- int `I4util_btr` (int b, volatile `I4_umword_t` \*dest)  
*Bit test and reset.*
- int `I4util_btc` (int b, volatile `I4_umword_t` \*dest)  
*Bit test and complement.*
- int `I4util_bsr` (`I4_umword_t` word)  
*Bit scan reverse.*
- int `I4util_bsf` (`I4_umword_t` word)  
*Bit scan forward.*
- int `I4util_find_first_set_bit` (const void \*dest, `I4_size_t` size)  
*Find the first set bit in a memory region.*
- int `I4util_find_first_zero_bit` (const void \*dest, `I4_size_t` size)  
*Find the first zero bit in a memory region.*
- int `I4util_next_power2` (unsigned long val)  
*Find the next power of 2 for a given number.*

## 17.637.1 Detailed Description

bit manipulation functions

### Date

07/03/2001

### Author

Lars Reuther [reuther@os.inf.tu-dresden.de](mailto:reuther@os.inf.tu-dresden.de)

Definition in file [bitops.h](#).

## 17.638 bitops.h

[Go to the documentation of this file.](#)

```

00001 /*****
00009 */
00010 * (c) 2000-2009 Author(s)
00011 * economic rights: Technische Universität Dresden (Germany)
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014
00015 /*****
00016 #ifndef __L4UTIL__INCLUDE__BITOPS_H__
00017 #define __L4UTIL__INCLUDE__BITOPS_H__
00018
00019 /* L4 includes */
00020 #include <l4/sys/l4int.h>
00021 #include <l4/sys/compiler.h>
00022
00024 #define l4util_test_and_clear_bit(b, dest) l4util_btr(b, dest)
00025 #define l4util_test_and_set_bit(b, dest) l4util_bts(b, dest)
00026 #define l4util_test_and_change_bit(b, dest) l4util_btc(b, dest)
00027 #define l4util_log2(word) l4util_bsr(word)
00028
00029 /*****
00030 *** Prototypes
00031 *****/
00032
00033 L4_BEGIN_DECLS
00034
00039
00047 L4_INLINE void
00048 l4util_set_bit(int b, volatile l4_umword_t * dest);
00049
00057 L4_INLINE void
00058 l4util_clear_bit(int b, volatile l4_umword_t * dest);
00059
00067 L4_INLINE void
00068 l4util_complement_bit(int b, volatile l4_umword_t * dest);
00069
00079 L4_INLINE int
00080 l4util_test_bit(int b, const volatile l4_umword_t * dest);
00081
00093 L4_INLINE int
00094 l4util_bts(int b, volatile l4_umword_t * dest);
00095
00107 L4_INLINE int
00108 l4util_btr(int b, volatile l4_umword_t * dest);
00109
00121 L4_INLINE int
00122 l4util_btc(int b, volatile l4_umword_t * dest);
00123
00135 L4_INLINE int
00136 l4util_bsr(l4_umword_t word);
00137
00149 L4_INLINE int
00150 l4util_bsf(l4_umword_t word);
00151
00163 L4_INLINE int

```

```

00164 l4util_find_first_set_bit(const void * dest, l4_size_t size);
00165
00177 L4_INLINE int
00178 l4util_find_first_zero_bit(const void * dest, l4_size_t size);
00179
00180
00189 L4_INLINE int
00190 l4util_next_power2(unsigned long val);
00191
00192 L4_END_DECLS
00193
00194 /*****
00195 *** Implementation of specific version
00196 *****/
00197
00198 #include <l4/util/bitops_arch.h>
00199
00200 /*****
00201 *** Generic implementations
00202 *****/
00203
00204 #ifndef __L4UTIL_BITOPS_HAVE_ARCH_SET_BIT
00205 #include <l4/util/atomic.h>
00206 L4_INLINE void
00207 l4util_set_bit(int b, volatile l4_umword_t * dest)
00208 {
00209 l4_umword_t oldval, newval;
00210
00211 dest += b / (sizeof(*dest) * 8); /* advance dest to the proper element */
00212 b &= sizeof(*dest) * 8 - 1; /* modulo; cut off all upper bits */
00213
00214 do
00215 {
00216 oldval = *dest;
00217 newval = oldval | (1UL << b);
00218 }
00219 while (!l4util_cmpxchg(dest, oldval, newval));
00220 }
00221 #endif
00222
00223 #ifndef __L4UTIL_BITOPS_HAVE_ARCH_CLEAR_BIT
00224 #include <l4/util/atomic.h>
00225 L4_INLINE void
00226 l4util_clear_bit(int b, volatile l4_umword_t * dest)
00227 {
00228 l4_umword_t oldval, newval;
00229
00230 dest += b / (sizeof(*dest) * 8);
00231 b &= sizeof(*dest) * 8 - 1;
00232
00233 do
00234 {
00235 oldval = *dest;
00236 newval = oldval & ~(1UL << b);
00237 }
00238 while (!l4util_cmpxchg(dest, oldval, newval));
00239 }
00240 #endif
00241
00242 #ifndef __L4UTIL_BITOPS_HAVE_ARCH_TEST_BIT
00243 L4_INLINE int
00244 l4util_test_bit(int b, const volatile l4_umword_t * dest)
00245 {
00246 dest += b / (sizeof(*dest) * 8);
00247 b &= sizeof(*dest) * 8 - 1;
00248
00249 return (*dest >> b) & 1;
00250 }
00251 #endif
00252
00253 #ifndef __L4UTIL_BITOPS_HAVE_ARCH_BIT_TEST_AND_SET
00254 #include <l4/util/atomic.h>
00255 L4_INLINE int
00256 l4util_bts(int b, volatile l4_umword_t * dest)
00257 {
00258 l4_umword_t oldval, newval;
00259
00260 dest += b / (sizeof(*dest) * 8);
00261 b &= sizeof(*dest) * 8 - 1;
00262
00263 do
00264 {
00265 oldval = *dest;
00266 newval = oldval | (1UL << b);
00267 }
00268 while (!l4util_cmpxchg(dest, oldval, newval));
00269

```

```

00270 /* Return old bit */
00271 return (oldval >> b) & 1;
00272 }
00273 #endif
00274
00275 #ifndef __L4UTIL_BITOPS_HAVE_ARCH_BIT_TEST_AND_RESET
00276 #include <l4/util/atomic.h>
00277 L4_INLINE int
00278 l4util_btr(int b, volatile l4_umword_t * dest)
00279 {
00280 l4_umword_t oldval, newval;
00281
00282 dest += b / (sizeof(*dest) * 8);
00283 b &= sizeof(*dest) * 8 - 1;
00284
00285 do
00286 {
00287 oldval = *dest;
00288 newval = oldval & ~(1UL << b);
00289 }
00290 while (!l4util_cmpxchg(dest, oldval, newval));
00291
00292 /* Return old bit */
00293 return (oldval >> b) & 1;
00294 }
00295 #endif
00296
00297 #ifndef __L4UTIL_BITOPS_HAVE_ARCH_BIT_SCAN_REVERSE
00298 L4_INLINE int
00299 l4util_bsr(l4_umword_t word)
00300 {
00301 int i;
00302
00303 if (!word)
00304 return -1;
00305
00306 for (i = 8 * sizeof(word) - 1; i >= 0; i--)
00307 if ((1UL << i) & word)
00308 return i;
00309
00310 __builtin_unreachable();
00311 }
00312 #endif
00313
00314 #ifndef __L4UTIL_BITOPS_HAVE_ARCH_BIT_SCAN_FORWARD
00315 L4_INLINE int
00316 l4util_bsf(l4_umword_t word)
00317 {
00318 unsigned int i;
00319
00320 if (!word)
00321 return -1;
00322
00323 for (i = 0; i < sizeof(word) * 8; i++)
00324 if ((1UL << i) & word)
00325 return i;
00326
00327 __builtin_unreachable();
00328 }
00329 #endif
00330
00331 #ifndef __L4UTIL_BITOPS_HAVE_ARCH_FIND_FIRST_ZERO_BIT
00332 L4_INLINE int
00333 l4util_find_first_zero_bit(const void * dest, l4_size_t size)
00334 {
00335 l4_size_t i, j;
00336 unsigned long *v = (unsigned long*)dest;
00337
00338 if (!size)
00339 return 0;
00340
00341 size = (size + 31) & ~0x1f; /* Grmbl: adapt to x86 implementation... */
00342
00343 for (i = j = 0; i < size; i++, j++)
00344 {
00345 if (j >= sizeof(*v) * 8)
00346 {
00347 j = 0;
00348 v++;
00349 }
00350 if (!(1UL << j) & *v)
00351 return i;
00352 }
00353 return size + 1;
00354 }
00355 #endif
00356

```

```

00357 #ifndef __L4UTIL_BITOPS_HAVE_ARCH_COMPLEMENT_BIT
00358 L4_INLINE void
00359 l4util_complement_bit(int b, volatile l4_umword_t * dest)
00360 {
00361 dest += b / (sizeof(*dest) * 8);
00362 b &= sizeof(*dest) * 8 - 1;
00363 *dest ^= 1UL << b;
00364 }
00365 #endif
00366
00367 /*
00368 * Adapted from:
00369 * http://en.wikipedia.org/wiki/Power_of_two#Algorithm_to_find_the_next-highest_power_of_two
00370 */
00371 L4_INLINE int
00372 l4util_next_power2(unsigned long val)
00373 {
00374 unsigned i;
00375 if (val == 0)
00376 return 1;
00377 val--;
00378 for (i=1; i < sizeof(unsigned long)*8; i++)
00379 val = val | val >> i;
00380 return val+1;
00381 }
00382
00383 /* Non-implemented version, catch with a linker warning */
00384 extern int __this_l4util_bitops_function_is_not_implemented_for_this_arch__sorry(void);
00385
00386 #ifndef __L4UTIL_BITOPS_HAVE_ARCH_BIT_TEST_AND_COMPLEMENT
00387 L4_INLINE int
00388 l4util_btc(int b, volatile l4_umword_t * dest)
00389 { (void)b; (void)dest; __this_l4util_bitops_function_is_not_implemented_for_this_arch__sorry(); return 0; }
00390 #endif
00391
00392 #ifndef __L4UTIL_BITOPS_HAVE_ARCH_FIND_FIRST_SET_BIT
00393 L4_INLINE int
00394 l4util_find_first_set_bit(const void * dest, l4_size_t size)
00395 { (void)dest; (void)size; __this_l4util_bitops_function_is_not_implemented_for_this_arch__sorry(); return 0; }
00396 #endif
00397
00398 #endif /* ! __L4UTIL__INCLUDE__BITOPS_H__ */

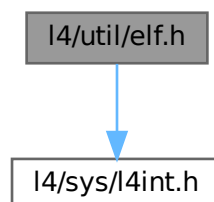
```

## 17.639 l4/util/elf.h File Reference

ELF definition.

```
#include <l4/sys/l4int.h>
```

Include dependency graph for elf.h:



## Data Structures

- struct [Elf32\\_Ehdr](#)  
*ELF32 header.*
- struct [Elf64\\_Ehdr](#)  
*ELF64 header.*
- struct [Elf32\\_Shdr](#)  
*ELF32 section header.*
- struct [Elf64\\_Shdr](#)  
*ELF64 section header.*
- struct [Elf32\\_Phdr](#)  
*ELF32 program header.*
- struct [Elf64\\_Phdr](#)  
*ELF64 program header.*
- struct [Elf32\\_Dyn](#)  
*ELF32 dynamic entry.*
- struct [Elf64\\_Dyn](#)  
*ELF64 dynamic entry.*
- struct [Elf32\\_Rel](#)  
*ELF32 relocation entry w/o addend.*
- struct [Elf32\\_Rela](#)  
*ELF32 relocation entry w/ addend.*
- struct [Elf64\\_Rel](#)  
*ELF64 relocation entry w/o addend.*
- struct [Elf64\\_Rela](#)  
*ELF64 relocation entry w/ addend.*
- struct [Elf32\\_Sym](#)  
*ELF32 symbol table entry.*
- struct [Elf64\\_Sym](#)  
*ELF64 symbol table entry.*
- struct [Elf32\\_Auxv](#)  
*Auxiliary vector (32-bit).*
- struct [Elf64\\_Auxv](#)  
*Auxiliary vector (64-bit).*

## Macros

- #define [ElfW](#)(type)  
*Use 64 or 32 bits types depending on the target architecture.*
- #define [ELF32\\_R\\_SYM](#)(i)  
*Symbol table index.*
- #define [ELF32\\_R\\_TYPE](#)(i)
- #define [ELF32\\_R\\_INFO](#)(s, t)  
*Create info from symbol table index + type.*
- #define [ELF64\\_R\\_SYM](#)(i)  
*Symbol table index.*
- #define [ELF64\\_R\\_TYPE](#)(i)
- #define [ELF64\\_R\\_INFO](#)(s, t)  
*Create info from symbol table index + type.*
- #define [ELF32\\_ST\\_BIND](#)(i)
- #define [ELF32\\_ST\\_TYPE](#)(i)

- `#define ELF32_ST_INFO(b, t)`  
*Make info from bind + type.*
- `#define ELF64_ST_BIND(i)`
- `#define ELF64_ST_TYPE(i)`
- `#define ELF64_ST_INFO(b, t)`  
*Make info from bind + type.*

## Typedefs

- `typedef struct Elf32_Auxv Elf32_Auxv`  
*Auxiliary vector (32-bit).*
- `typedef struct Elf64_Auxv Elf64_Auxv`  
*Auxiliary vector (64-bit).*

## ELF types

- `typedef l4\_uint32\_t Elf32_Addr`  
*size 4 align 4*
- `typedef l4\_uint32\_t Elf32_Off`  
*size 4 align 4*
- `typedef l4\_uint16\_t Elf32_Half`  
*size 2 align 2*
- `typedef l4\_uint32\_t Elf32_Word`  
*size 4 align 4*
- `typedef l4\_int32\_t Elf32_Sword`  
*size 4 align 4*
- `typedef l4\_uint64\_t Elf64_Addr`  
*size 8 align 8*
- `typedef l4\_uint64\_t Elf64_Off`  
*size 8 align 8*
- `typedef l4\_uint16\_t Elf64_Half`  
*size 2 align 2*
- `typedef l4\_uint32\_t Elf64_Word`  
*size 4 align 4*
- `typedef l4\_int32\_t Elf64_Sword`  
*size 4 align 4*
- `typedef l4\_uint64\_t Elf64_Xword`  
*size 8 align 8*
- `typedef l4\_int64\_t Elf64_Sxword`  
*size 8 align 8*

## Enumerations

- `enum { EI_NIDENT = 16 }`
- `enum Elf_ETs {`  
`ET\_NONE = 0 , ET\_REL = 1 , ET\_EXEC = 2 , ET\_DYN = 3 ,`  
`ET\_CORE = 4 , ET\_LOPROC = 0xff00 , ET\_HIPROC = 0xffff }`  
*Object file type.*



- enum `Elf_EMs` {  
`EM_NONE` = 0 , `EM_M32` = 1 , `EM_SPARC` = 2 , `EM_386` = 3 ,  
`EM_68K` = 4 , `EM_88K` = 5 , `EM_860` = 7 , `EM_MIPS` = 8 ,  
`EM_MIPS_RS4_BE` = 10 , `EM_SPARC64` = 11 , `EM_PARISC` = 15 , `EM_VPP500` = 17 ,  
`EM_SPARC32PLUS` = 18 , `EM_960` = 19 , `EM_PPC` = 20 , `EM_V800` = 36 ,  
`EM_FR20` = 37 , `EM_RH32` = 38 , `EM_RCE` = 39 , `EM_ARM` = 40 ,  
`EM_ALPHA` = 41 , `EM_SH` = 42 , `EM_SPARCV9` = 43 , `EM_TRICORE` = 44 ,  
`EM_ARC` = 45 , `EM_H8_300` = 46 , `EM_H8_300H` = 47 , `EM_H8S` = 48 ,  
`EM_H8_500` = 49 , `EM_IA_64` = 50 , `EM_MIPS_X` = 51 , `EM_COLDFIRE` = 52 ,  
`EM_68HC12` = 53 , `EM_X86_64` = 62 , `EM_PDSP` = 63 , `EM_FX66` = 66 ,  
`EM_ST9PLUS` = 67 , `EM_ST7` = 68 , `EM_68HC16` = 69 , `EM_68HC11` = 70 ,  
`EM_68HC08` = 71 , `EM_68HC05` = 72 , `EM_SVX` = 73 , `EM_ST19` = 74 ,  
`EM_VAX` = 75 , `EM_CRIS` = 76 , `EM_JAVELIN` = 77 , `EM_FIREPATH` = 78 ,  
`EM_ZSP` = 79 , `EM_MMIX` = 80 , `EM_HUANY` = 81 , `EM_PRISM` = 82 ,  
`EM_AVR` = 83 , `EM_FR30` = 84 , `EM_D10V` = 85 , `EM_D30V` = 86 ,  
`EM_V850` = 87 , `EM_M32R` = 88 , `EM_MN10300` = 89 , `EM_MN10200` = 90 ,  
`EM_PJ` = 91 , `EM_OPENRISC` = 92 , `EM_ARC_A5` = 93 , `EM_XTENSA` = 94 ,  
`EM_ALTERA_NIOS2` = 113 , `EM_AARCH64` = 183 , `EM_TILEPRO` = 188 , `EM_MICROBLAZE` = 189 ,  
`EM_TILEGX` = 191 , `EM_RISCV` = 243 , **`EM_NUM`** = 244 }  
*Required architecture.*
- enum `Elf_EVs` { `EV_NONE` = 0 , `EV_CURRENT` = 1 }  
*Object file version.*
- enum `Elf_EIs` {  
`EI_MAG0` = 0 , `EI_MAG1` = 1 , `EI_MAG2` = 2 , `EI_MAG3` = 3 ,  
`EI_CLASS` = 4 , `EI_DATA` = 5 , `EI_VERSION` = 6 , `EI_OSABI` = 7 ,  
`EI_ABIVERSION` = 8 , `EI_PAD` = 9 }  
*Identification Indices.*
- enum `Elf_MAGs` { `ELFMAG0` = 0x7f , `ELFMAG1` = 'E' , `ELFMAG2` = 'L' , `ELFMAG3` = 'F' }  
*Magic number.*
- enum `Elf_CIASSs` { `ELFCLASSNONE` = 0 , `ELFCLASS32` = 1 , `ELFCLASS64` = 2 , `ELFCLASSNUM` = 3 }  
*File class or capacity.*
- enum `Elf_DATAs` { `ELFDATANONE` = 0 , `ELFDATA2LSB` = 1 , `ELFDATA2MSB` = 2 , `ELFDATANUM` = 3 }  
*Data encoding.*
- enum `Elf_OSABIs` {  
`ELFOSABI_NONE` = 0 , `ELFOSABI_SYSV` = 0 , `ELFOSABI_HPUX` = 1 , `ELFOSABI_NETBSD` = 2 ,  
`ELFOSABI_LINUX` = 3 , `ELFOSABI_SOLARIS` = 6 , `ELFOSABI_AIX` = 7 , `ELFOSABI_IRIX` = 8 ,  
`ELFOSABI_FREEBSD` = 9 , `ELFOSABI_TRU64` = 10 , `ELFOSABI_MODESTO` = 11 , `ELFOSABI_OPENBSD`  
= 12 ,  
`ELFOSABI_ARM` = 97 , `ELFOSABI_STANDALONE` = 255 }  
*Identify operating system and ABI to which the object is targeted.*
- enum `Elf_SHNs` {  
`SHN_UNDEF` = 0 , `SHN_LORESERVE` = 0xff00 , `SHN_LOPROC` = 0xff00 , `SHN_HIPROC` = 0xff1f ,  
`SHN_ABS` = 0xffff1 , `SHN_COMMON` = 0xffff2 , `SHN_HIRESERVE` = 0xffff }  
*Special section indexes.*
- enum `Elf_SHTs` {  
`SHT_NULL` = 0 , `SHT_PROGBITS` = 1 , `SHT_SYMTAB` = 2 , `SHT_STRTAB` = 3 ,  
`SHT_RELA` = 4 , `SHT_HASH` = 5 , `SHT_DYNAMIC` = 6 , `SHT_NOTE` = 7 ,  
`SHT_NOBITS` = 8 , `SHT_REL` = 9 , `SHT_SHLIB` = 10 , `SHT_DYNSYM` = 11 ,  
`SHT_INIT_ARRAY` = 14 , `SHT_FINI_ARRAY` = 15 , `SHT_PREINIT_ARRAY` = 16 , `SHT_GROUP` = 17 ,  
`SHT_SYMTAB_SHNDX` = 18 , `SHT_NUM` = 19 , `SHT_LOOS` = 0x60000000 , `SHT_HIOS` = 0x6fffffff ,  
`SHT_LOPROC` = 0x70000000 , `SHT_HIPROC` = 0x7fffffff , `SHT_LOUSER` = 0x80000000 , `SHT_HIUSER` =  
0xffffffff }  
*Section type.*
- enum `Elf_SHFs` {  
`SHF_WRITE` = 0x1 , `SHF_ALLOC` = 0x2 , `SHF_EXECINSTR` = 0x4 , `SHF_MERGE` = 0x10 ,  
`SHF_STRINGS` = 0x20 , `SHF_INFO_LINK` = 0x40 , `SHF_OS_NONCONFORMING` = 0x100 , `SHF_GROUP`

= 0x200 ,  
 SHF\_TLS = 0x400 , SHF\_MASKOS = 0x0ff00000 , SHF\_MASKPROC = 0xf0000000 }

*Section attribute flags.*

- enum Elf\_PTs {  
 PT\_NULL = 0 , PT\_LOAD = 1 , PT\_DYNAMIC = 2 , PT\_INTERP = 3 ,  
 PT\_NOTE = 4 , PT\_SHLIB = 5 , PT\_PHDR = 6 , PT\_TLS = 7 ,  
 PT\_NUM = 8 , PT\_LOOS = 0x60000000 , PT\_HIOS = 0x6ffffff , PT\_LOPROC = 0x70000000 ,  
 PT\_HIPROC = 0x7ffffff , PT\_GNU\_EH\_FRAME = PT\_LOOS + 0x474e550 , PT\_GNU\_STACK = PT\_LOOS  
 + 0x474e551 , PT\_GNU\_RELRO = PT\_LOOS + 0x474e552 ,  
 PT\_L4\_STACK = PT\_LOOS + 0x12 , PT\_L4\_AUX = PT\_LOOS + 0x14 }

*Segment types.*

- enum ELF\_PFs {  
 PF\_X = 0x1 , PF\_W = 0x2 , PF\_R = 0x4 , PF\_MASKOS = 0x0ff00000 ,  
 PF\_MASKPROC = 0x7ffffff }

*Segment permissions.*

- enum Elf\_NTscore {  
 NT\_PRSTATUS = 1 , NT\_FPREGSET = 2 , NT\_PRPSINFO = 3 , NT\_PRXREG = 4 ,  
 NT\_TASKSTRUCT = 4 , NT\_PLATFORM = 5 , NT\_AUXV = 6 , NT\_GWINDOWS = 7 ,  
 NT\_ASRS = 8 , NT\_PSTATUS = 10 , NT\_PSINFO = 13 , NT\_PRCRED = 14 ,  
 NT\_UTSNAME = 15 , NT\_LWPSTATUS = 16 , NT\_LWPSINFO = 17 , NT\_PRFPXREG = 20 }

*Legal values for note segment descriptor types for core files.*

- enum Elf\_NTscobj { NT\_VERSION = 1 }

*Legal values for the note segment descriptor types for object files.*

- enum Elf\_DTsc {  
 DT\_NULL = 0 , DT\_NEEDED = 1 , DT\_PLTRELSZ = 2 , DT\_PLTGOT = 3 ,  
 DT\_HASH = 4 , DT\_STRTAB = 5 , DT\_SYMTAB = 6 , DT\_RELA = 7 ,  
 DT\_RELASZ = 8 , DT\_RELAENT = 9 , DT\_STRSZ = 10 , DT\_SYMENT = 11 ,  
 DT\_INIT = 12 , DT\_FINI = 13 , DT\_SONAME = 14 , DT\_RPATH = 15 ,  
 DT\_SYMBOLIC = 16 , DT\_REL = 17 , DT\_RELSZ = 18 , DT\_RELENT = 19 ,  
 DT\_PTRREL = 20 , DT\_DEBUG = 21 , DT\_TEXTREL = 22 , DT\_JMPREL = 23 ,  
 DT\_BIND\_NOW = 24 , DT\_INIT\_ARRAY = 25 , DT\_FINI\_ARRAY = 26 , DT\_INIT\_ARRAYSZ = 27 ,  
 DT\_FINI\_ARRAYSZ = 28 , DT\_RUNPATH = 29 , DT\_FLAGS = 30 , DT\_ENCODING = 32 ,  
 DT\_PREINIT\_ARRAY = 32 , DT\_PREINIT\_ARRAYSZ = 33 , DT\_NUM = 34 , DT\_LOOS = 0x6000000d ,  
 DT\_HIOS = 0x6ffffff0 , DT\_LOPROC = 0x70000000 , DT\_HIPROC = 0x7ffffff }

*Dynamic Array Tags.*

- enum Elf\_DFsc {  
 DF\_ORIGIN = 0x00000001 , DF\_SYMBOLIC = 0x00000002 , DF\_TEXTREL = 0x00000004 ,  
 DF\_BIND\_NOW = 0x00000008 ,  
 DF\_STATIC\_TLS = 0x00000010 }

*Values of Elf32\_Dyn.d\_un.d\_val, Elf64\_Dyn.d\_un.d\_val in the DT\_FLAGS entry.*

- enum Elf\_DF\_1sc {  
 DF\_1\_NOW = 0x00000001 , DF\_1\_GLOBAL = 0x00000002 , DF\_1\_GROUP = 0x00000004 ,  
 DF\_1\_NODELETE = 0x00000008 ,  
 DF\_1\_LOADFLTR = 0x00000010 , DF\_1\_INITFIRST = 0x00000020 , DF\_1\_NOOPEN = 0x00000040 ,  
 DF\_1\_ORIGIN = 0x00000080 ,  
 DF\_1\_DIRECT = 0x00000100 , DF\_1\_TRANS = 0x00000200 , DF\_1\_INTERPOSE = 0x00000400 ,  
 DF\_1\_NODEFLIB = 0x00000800 ,  
 DF\_1\_NODUMP = 0x00001000 , DF\_1\_CONFALT = 0x00002000 , DF\_1\_ENDFILTEE = 0x00004000 ,  
 DF\_1\_DISPRELDNE = 0x00008000 ,  
 DF\_1\_DISPRELPND = 0x00010000 }

*State flags selectable in the Elf32\_Dyn.d\_un.d\_val / Elf64\_Dyn.d\_un.d\_val element of the DT\_FLAGS\_1 entry in the dynamic section.*

- enum Elf\_DTF\_1sc

*Flags for the feature selection in DT\_FEATURE\_1.*

- enum Elf\_DF\_P1sc { DF\_P1\_LAZYLOAD = 0x00000001 , DF\_P1\_GROUPEPERM = 0x00000002 }

*Flags in the DT\_POSFLAG\_1 entry effecting only the next DT\_\* entry.*

- enum `Elf_R_386_s` {  
`R_386_NONE` = 0 , `R_386_32` = 1 , `R_386_PC32` = 2 , `R_386_GOT32` = 3 ,  
`R_386_PLT32` = 4 , `R_386_COPY` = 5 , `R_386_GLOB_DAT` = 6 , `R_386_JMP_SLOT` = 7 ,  
`R_386_RELATIVE` = 8 , `R_386_GOTOFF` = 9 , `R_386_GOTPC` = 10 , `R_386_32PLT` = 11 ,  
`R_386_TLS_TPOFF` = 14 , `R_386_TLS_IE` = 15 , `R_386_TLS_GOTIE` = 16 , `R_386_TLS_LE` = 17 ,  
`R_386_TLS_GD` = 18 , `R_386_TLS_LDM` = 19 , `R_386_16` = 20 , `R_386_PC16` = 21 ,  
`R_386_8` = 22 , `R_386_PC8` = 23 , `R_386_TLS_GD_32` = 24 , `R_386_TLS_GD_PUSH` = 25 ,  
`R_386_TLS_GD_CALL` = 26 , `R_386_TLS_GD_POP` = 27 , `R_386_TLS_LDM_32` = 28 , `R_386_TLS_LDM_PUSH`  
= 29 ,  
`R_386_TLS_LDM_CALL` = 30 , `R_386_TLS_LDM_POP` = 31 , `R_386_TLS_LDO_32` = 32 , `R_386_TLS_IE_32`  
= 33 ,  
`R_386_TLS_LE_32` = 34 , `R_386_TLS_DTPMOD32` = 35 , `R_386_TLS_DTPOFF32` = 36 , `R_386_TLS_TPOFF32`  
= 37 ,  
`R_386_NUM` = 38 }  
*Relocation types (processor specific).*
- enum `Elf_EF_ARM_s` { }  
*ARM specific declarations.*
- enum `Elf_STT_ARM_s`  
*Additional symbol types for Thumb.*
- enum `Elf_SHF_s_ARM` { `SHF_ARM_ENTRYSECT` = 0x10000000 , `SHF_ARM_COMDEF` = 0x80000000 }  
*ARM-specific values for `Elf32_Shdr.sh_flags` / `Elf64_Shdr.sh_flags`.*
- enum `Elf_ARM_SBs` { `PF_ARM_SB` = 0x10000000 }  
*ARM-specific program header flags.*
- enum `Elf_R_ARM_s` {  
`R_ARM_NONE` = 0 , `R_ARM_PC24` = 1 , `R_ARM_ABS32` = 2 , `R_ARM_REL32` = 3 ,  
`R_ARM_PC13` = 4 , `R_ARM_ABS16` = 5 , `R_ARM_ABS12` = 6 , `R_ARM_THM_ABS5` = 7 ,  
`R_ARM_ABS8` = 8 , `R_ARM_SBREL32` = 9 , `R_ARM_THM_PC22` = 10 , `R_ARM_THM_PC8` = 11 ,  
`R_ARM_AMP_VCALL9` = 12 , `R_ARM_SWI24` = 13 , `R_ARM_THM_SWI8` = 14 , `R_ARM_XPC25` = 15 ,  
`R_ARM_THM_XPC22` = 16 , `R_ARM_COPY` = 20 , `R_ARM_GLOB_DAT` = 21 , `R_ARM_JUMP_SLOT` = 22 ,  
`R_ARM_RELATIVE` = 23 , `R_ARM_GOTOFF` = 24 , `R_ARM_GOTPC` = 25 , `R_ARM_GOT32` = 26 ,  
`R_ARM_PLT32` = 27 , `R_ARM_ALU_PCREL_7_0` = 32 , `R_ARM_ALU_PCREL_15_8` = 33 , `R_ARM_↵`  
`ALU_PCREL_23_15` = 34 ,  
`R_ARM_LDR_SBREL_11_0` = 35 , `R_ARM_ALU_SBREL_19_12` = 36 , `R_ARM_ALU_SBREL_27_20` =  
37 , `R_ARM_GNU_VTENTRY` = 100 ,  
`R_ARM_GNU_VTINHERIT` = 101 , `R_ARM_THM_PC11` = 102 , `R_ARM_THM_PC9` = 103 , `R_ARM_↵`  
`RXPC25` = 249 ,  
`R_ARM_RSBREL32` = 250 , `R_ARM_THM_RPC22` = 251 , `R_ARM_RREL32` = 252 , `R_ARM_RABS22` =  
253 ,  
`R_ARM_RPC24` = 254 , `R_ARM_RBASE` = 255 , `R_ARM_NUM` = 256 }  
*ARM relocations.*
- enum `Elf_R_AARCH64_s` { `R_AARCH64_NONE` = 0 , `R_AARCH64_RELATIVE` = 1027 }  
*AARCH64 relocations.*
- enum `Elf_R_X86_64_s` {  
`R_X86_64_NONE` = 0 , `R_X86_64_64` = 1 , `R_X86_64_PC32` = 2 , `R_X86_64_GOT32` = 3 ,  
`R_X86_64_PLT32` = 4 , `R_X86_64_COPY` = 5 , `R_X86_64_GLOB_DAT` = 6 , `R_X86_64_JUMP_SLOT` = 7 ,  
`R_X86_64_RELATIVE` = 8 , `R_X86_64_GOTPCREL` = 9 , `R_X86_64_32` = 10 , `R_X86_64_32S` = 11 ,  
`R_X86_64_16` = 12 , `R_X86_64_PC16` = 13 , `R_X86_64_8` = 14 , `R_X86_64_PC8` = 15 ,  
`R_X86_64_DTPMOD64` = 16 , `R_X86_64_DTPOFF64` = 17 , `R_X86_64_TPOFF64` = 18 , `R_X86_64_TLSD`  
= 19 ,  
`R_X86_64_TLSD` = 20 , `R_X86_64_DTPOFF32` = 21 , `R_X86_64_GOTTPOFF` = 22 , `R_X86_64_TPOFF32`  
= 23 ,  
`R_X86_64_NUM` = 24 }  
*AMD x86-64 relocations.*
- enum `Elf_STNs`  
*Symbol Table Entry.*

- enum `Elf_STBs` {  
`STB_LOCAL` = 0 , `STB_GLOBAL` = 1 , `STB_WEAK` = 2 , `STB_LOOS` = 10 ,  
`STB_HIOS` = 12 , `STB_LOPROC` = 13 , `STB_HIPROC` = 15 }  
*Symbol Binding.*
- enum `Elf_STTs` {  
`STT_NOTYPE` = 0 , `STT_OBJECT` = 1 , `STT_FUNC` = 2 , `STT_SECTION` = 3 ,  
`STT_FILE` = 4 , `STT_LOOS` = 10 , `STT_HIOS` = 12 , `STT_LOPROC` = 13 ,  
`STT_HIPROC` = 15 }  
*Symbol Types.*
- enum `Elf_ATs` {  
`AT_NULL` = 0 , `AT_IGNORE` = 1 , `AT_EXECD` = 2 , `AT_PHDR` = 3 ,  
`AT_PHENT` = 4 , `AT_PHNUM` = 5 , `AT_PAGESZ` = 6 , `AT_BASE` = 7 ,  
`AT_FLAGS` = 8 , `AT_ENTRY` = 9 , `AT_NOTELF` = 10 , `AT_UID` = 11 ,  
`AT_EUID` = 12 , `AT_GID` = 13 , `AT_EGID` = 14 , `AT_L4_AUX` = 0xf0 ,  
`AT_L4_ENV` = 0xf1 }  
*Legal values for `Elf32_Auxv.atype` / `Elf64_Auxv.atype`.*

## 17.639.1 Detailed Description

ELF definition.

Date

08/18/2000

Author

Frank Mehnert [fm3@os.inf.tu-dresden.de](mailto:fm3@os.inf.tu-dresden.de) Alexander Warg [aw11@os.inf.tu-dresden.de](mailto:aw11@os.inf.tu-dresden.de)

Many structs from "Executable and Linkable Format (ELF)", Portable Formats Specification, Version 1.1 and "↔ System V Application Binary Interface - DRAFT - April 29, 1998" The Santa Cruz Operation, Inc. (see [http↔://www.sco.com/developer/gabi/contents.html](http://www.sco.com/developer/gabi/contents.html))

Definition in file [elf.h](#).

## 17.640 elf.h

[Go to the documentation of this file.](#)

```

00001
00019 /*
00020 * (c) 2008-2009 Author(s)
00021 * economic rights: Technische Universität Dresden (Germany)
00022 * License: see LICENSE.spdx (in this directory or the directories above)
00023 */
00024
00025 /* (c) 2003-2006 Technische Universitaet Dresden
00026 * License: see LICENSE.spdx (in this directory or the directories above) */
00027
00028 #pragma once
00029
00030 #include <14/sys/l4int.h>
00031
00037
00042 typedef l4_uint32_t Elf32_Addr;
00043 typedef l4_uint32_t Elf32_Off;
00044 typedef l4_uint16_t Elf32_Half;
00045 typedef l4_uint32_t Elf32_Word;
00046 typedef l4_int32_t Elf32_Sword;
```

```

00047 typedef l4_uint64_t Elf64_Addr;
00048 typedef l4_uint64_t Elf64_Off;
00049 typedef l4_uint16_t Elf64_Half;
00050 typedef l4_uint32_t Elf64_Word;
00051 typedef l4_int32_t Elf64_Sword;
00052 typedef l4_uint64_t Elf64_Xword;
00053 typedef l4_int64_t Elf64_Sxword;
00055
00060 #if L4_MWORD_BITS == 64
00061 # define ElfW(type) _ElfW(Elf, 64, type)
00062 #else
00063 # define ElfW(type) _ElfW(Elf, 32, type)
00064 #endif
00065 #define _ElfW(e,w,t) __ElfW(e, w, _##t)
00066 #define __ElfW(e,w,t) e##w##t
00067
00068 #if defined(ARCH_x86)
00069 # define L4_ARCH_EI_DATA ELFDATA2LSB
00070 # define L4_ARCH_E_MACHINE EM_386
00071 # define L4_ARCH_EI_CLASS ELFCLASS32
00072 #elif defined(ARCH_amd64)
00073 # define L4_ARCH_EI_DATA ELFDATA2LSB
00074 # define L4_ARCH_E_MACHINE EM_X86_64
00075 # define L4_ARCH_EI_CLASS ELFCLASS64
00076 #elif defined(ARCH_arm)
00077 # define L4_ARCH_EI_DATA ELFDATA2LSB
00078 # define L4_ARCH_E_MACHINE EM_ARM
00079 # define L4_ARCH_EI_CLASS ELFCLASS32
00080 #elif defined(ARCH_arm64)
00081 # define L4_ARCH_EI_DATA ELFDATA2LSB
00082 # define L4_ARCH_E_MACHINE EM_AARCH64
00083 # define L4_ARCH_EI_CLASS ELFCLASS64
00084 #elif defined(ARCH_ppc32)
00085 # define L4_ARCH_EI_DATA ELFDATA2MSB
00086 # define L4_ARCH_E_MACHINE EM_PPC
00087 # define L4_ARCH_EI_CLASS ELFCLASS32
00088 #elif defined(ARCH_sparc)
00089 # define L4_ARCH_EI_DATA ELFDATA2MSB
00090 # define L4_ARCH_E_MACHINE EM_SPARC
00091 # define L4_ARCH_EI_CLASS ELFCLASS32
00092 #elif defined(ARCH_mips)
00093 # define L4_ARCH_EI_DATA ELFDATA2LSB
00094 # define L4_ARCH_E_MACHINE EM_MIPS
00095 # ifdef __mips64
00096 # define L4_ARCH_EI_CLASS ELFCLASS64
00097 # else
00098 # define L4_ARCH_EI_CLASS ELFCLASS32
00099 # endif
00100 #elif defined(ARCH_riscv)
00101 # define L4_ARCH_EI_DATA ELFDATA2LSB
00102 # define L4_ARCH_E_MACHINE EM_RISCV
00103 # if __riscv_xlen == 64
00104 # define L4_ARCH_EI_CLASS ELFCLASS64
00105 # else
00106 # define L4_ARCH_EI_CLASS ELFCLASS32
00107 # endif
00108 #else
00109 # warning elf.h: Unsupported build architecture!
00110 #endif
00111
00112
00114
00116
00117 enum
00118 {
00119 EI_NIDENT = 16,
00120 };
00121
00125 typedef struct
00126 {
00127 unsigned char e_ident[EI_NIDENT];
00128 Elf32_Half e_type;
00129 Elf32_Half e_machine;
00130 Elf32_Word e_version;
00131 Elf32_Addr e_entry;
00132 Elf32_Off e_phoff;
00133 Elf32_Off e_shoff;
00134 Elf32_Word e_flags;
00135 Elf32_Half e_ehsize;
00136 Elf32_Half e_phentsize;
00137 Elf32_Half e_phnum;
00138 Elf32_Half e_shentsize;
00139 Elf32_Half e_shnum;
00140 Elf32_Half e_shstrndx;
00141 } Elf32_Ehdr;
00142
00146 typedef struct

```

```

00147 {
00148 unsigned char e_ident[EI_NIDENT];
00149 Elf64_Half e_type;
00150 Elf64_Half e_machine;
00151 Elf64_Word e_version;
00152 Elf64_Addr e_entry;
00153 Elf64_Off e_phoff;
00154 Elf64_Off e_shoff;
00155 Elf64_Word e_flags;
00156 Elf64_Half e_ehsize;
00157 Elf64_Half e_phentsize;
00158 Elf64_Half e_phnum;
00159 Elf64_Half e_shentsize;
00160 Elf64_Half e_shnum;
00161 Elf64_Half e_shstrndx;
00162 } Elf64_Ehdr;
00163
00164 enum Elf_ETs
00165 {
00170 ET_NONE = 0,
00171 ET_REL = 1,
00172 ET_EXEC = 2,
00173 ET_DYN = 3,
00174 ET_CORE = 4,
00175 ET_LOPROC = 0xff00,
00176 ET_HIPROC = 0xffff,
00177 };
00178
00183 enum Elf_EMs
00184 {
00185 EM_NONE = 0,
00186 EM_M32 = 1,
00187 EM_SPARC = 2,
00188 EM_386 = 3,
00189 EM_68K = 4,
00190 EM_88K = 5,
00191 EM_860 = 7,
00192 EM_MIPS = 8,
00193 EM_MIPS_RS4_BE = 10,
00194 EM_SPARC64 = 11,
00195 EM_PARISC = 15,
00196 EM_VPP500 = 17,
00197 EM_SPARC32PLUS = 18,
00198 EM_960 = 19,
00199 EM_PPC = 20,
00200 EM_V800 = 36,
00201 EM_FR20 = 37,
00202 EM_RH32 = 38,
00203 EM_RCE = 39,
00204 EM_ARM = 40,
00205 EM_ALPHA = 41,
00206 EM_SH = 42,
00207 EM_SPARCV9 = 43,
00208 EM_TRICORE = 44,
00209 EM_ARC = 45,
00210 EM_H8_300 = 46,
00211 EM_H8_300H = 47,
00212 EM_H8S = 48,
00213 EM_H8_500 = 49,
00214 EM_IA_64 = 50,
00215 EM_MIPS_X = 51,
00216 EM_COLDFIRE = 52,
00217 EM_68HC12 = 53,
00218 EM_X86_64 = 62,
00219 EM_PDSP = 63,
00220 EM_FX66 = 66,
00221 EM_ST9PLUS = 67,
00222 EM_ST7 = 68,
00223 EM_68HC16 = 69,
00224 EM_68HC11 = 70,
00225 EM_68HC08 = 71,
00226 EM_68HC05 = 72,
00227 EM_SVX = 73,
00228 EM_ST19 = 74,
00229 EM_VAX = 75,
00230 EM_CRIS = 76,
00231 EM_JAVELIN = 77,
00232 EM_FIREPATH = 78,
00233 EM_ZSP = 79,
00234 EM_MMIX = 80,
00235 EM_HUANY = 81,
00236 EM_PRISM = 82,
00237 EM_AVR = 83,
00238 EM_FR30 = 84,
00239 EM_D10V = 85,
00240 EM_D30V = 86,
00241 EM_V850 = 87,

```

```

00242 EM_M32R = 88,
00243 EM_MN10300 = 89,
00244 EM_MN10200 = 90,
00245 EM_PJ = 91,
00246 EM_OPENRISC = 92,
00247 EM_ARC_A5 = 93,
00248 EM_XTENSA = 94,
00249 EM_ALTERA_NIOS2 = 113,
00250 EM_AARCH64 = 183,
00251 EM_TILEPRO = 188,
00252 EM_MICROBLAZE = 189,
00253 EM_TILEGX = 191,
00254 EM_RISCV = 243,
00255 EM_NUM = 244,
00256 };
00257
00258 #if 0
00259 #define EM_ALPHA 0x9026 /* interim value used by Linux until the
00260 committee comes up with a final number */
00261 #define EM_S390 0xA390 /* interim value used for IBM S390 */
00262 #endif
00263
00266 enum Elf_EVs
00267 {
00268 EV_NONE = 0,
00269 EV_CURRENT = 1,
00270 };
00271
00274 enum Elf_EIs
00275 {
00276 EI_MAG0 = 0,
00277 EI_MAG1 = 1,
00278 EI_MAG2 = 2,
00279 EI_MAG3 = 3,
00280 EI_CLASS = 4,
00281 EI_DATA = 5,
00282 EI_VERSION = 6,
00283 EI_OSABI = 7,
00284 EI_ABIVERSION = 8,
00285 EI_PAD = 9,
00286 };
00287
00289 enum Elf_MAGs
00290 {
00291 ELFMAG0 = 0x7f,
00292 ELFMAG1 = 'E',
00293 ELFMAG2 = 'L',
00294 ELFMAG3 = 'F',
00295 };
00296
00298 enum Elf_CLASSES
00299 {
00300 ELFCLASSNONE = 0,
00301 ELFCLASS32 = 1,
00302 ELFCLASS64 = 2,
00303 ELFCLASSNUM = 3,
00304 };
00305
00307 enum Elf_DATAs
00308 {
00309 ELFDATANONE = 0,
00310 ELFDATA2LSB = 1,
00311 ELFDATA2MSB = 2,
00312 ELFDATANUM = 3,
00313 };
00314
00316 enum Elf_OSABIs
00317 {
00318 ELFOSABI_NONE = 0,
00319 ELFOSABI_SYSV = 0,
00320 ELFOSABI_HPUX = 1,
00321 ELFOSABI_NETBSD = 2,
00322 ELFOSABI_LINUX = 3,
00323 ELFOSABI_SOLARIS = 6,
00324 ELFOSABI_AIX = 7,
00325 ELFOSABI_IRIX = 8,
00326 ELFOSABI_FREEBSD = 9,
00327 ELFOSABI_TRU64 = 10,
00328 ELFOSABI_MODESTO = 11,
00329 ELFOSABI_OPENBSD = 12,
00330 ELFOSABI_ARM = 97,
00331 ELFOSABI_STANDALONE = 255,
00332 };
00333
00335 enum Elf_SHNs
00336 {
00337 SHN_UNDEF = 0,

```

```

00338 SHN_LORESERVE = 0xff00,
00339 SHN_LOPROC = 0xff00,
00340 SHN_HIPROC = 0xff1f,
00341 SHN_ABS = 0xffff1,
00342 SHN_COMMON = 0xffff2,
00343 SHN_HIRESERVE = 0xffff,
00344 };
00345
00347 typedef struct
00348 {
00349 Elf32_Word sh_name;
00350 Elf32_Word sh_type;
00351 Elf32_Word sh_flags;
00352 Elf32_Addr sh_addr;
00353 Elf32_Off sh_offset;
00354 Elf32_Word sh_size;
00355 Elf32_Word sh_link;
00356 Elf32_Word sh_info;
00357 Elf32_Word sh_addralign;
00358 Elf32_Word sh_entsize;
00359 } Elf32_Shdr;
00360
00362 typedef struct
00363 {
00364 Elf64_Word sh_name;
00365 Elf64_Word sh_type;
00366 Elf64_Xword sh_flags;
00367 Elf64_Addr sh_addr;
00368 Elf64_Off sh_offset;
00369 Elf64_Xword sh_size;
00370 Elf64_Word sh_link;
00371 Elf64_Word sh_info;
00372 Elf64_Xword sh_addralign;
00373 Elf64_Xword sh_entsize;
00374 } Elf64_Shdr;
00375
00377 enum Elf_SHTs
00378 {
00379 SHT_NULL = 0,
00380 SHT_PROGBITS = 1,
00381 SHT_SYMTAB = 2,
00382 SHT_STRTAB = 3,
00383 SHT_RELA = 4,
00384 SHT_HASH = 5,
00385 SHT_DYNAMIC = 6,
00386 SHT_NOTE = 7,
00387 SHT_NOBITS = 8,
00388 SHT_REL = 9,
00389 SHT_SHLIB = 10,
00390 SHT_DYNSYM = 11,
00391 SHT_INIT_ARRAY = 14,
00392 SHT_FINI_ARRAY = 15,
00393 SHT_PREINIT_ARRAY = 16,
00394 SHT_GROUP = 17,
00395 SHT_SYMTAB_SHNDX = 18,
00396 SHT_NUM = 19,
00397 SHT_LOOS = 0x60000000,
00398 SHT_HIOS = 0x6fffffff,
00399 SHT_LOPROC = 0x70000000,
00400 SHT_HIPROC = 0x7fffffff,
00401 SHT_LOUSER = 0x80000000,
00402 SHT_HIUSER = 0xffffffff,
00403 };
00404
00406 enum Elf_SHFs
00407 {
00408 SHF_WRITE = 0x1,
00409 SHF_ALLOC = 0x2,
00410 SHF_EXECINSTR = 0x4,
00411 SHF_MERGE = 0x10,
00412 SHF_STRINGS = 0x20,
00413 SHF_INFO_LINK = 0x40,
00414 SHF_LINK_ORDER = 0x80,
00415 SHF_OS_NONCONFORMING = 0x100,
00417 SHF_GROUP = 0x200,
00418 SHF_TLS = 0x400,
00419 SHF_MASKOS = 0x0ff00000,
00420 SHF_MASKPROC = 0xf0000000,
00421 };
00422
00423
00425 typedef struct
00426 {
00427 Elf32_Word p_type;
00428 Elf32_Off p_offset;
00429 Elf32_Addr p_vaddr;
00430 Elf32_Addr p_paddr;

```



```

00431 Elf32_Word p_filesz;
00432 Elf32_Word p_memsz;
00433 Elf32_Word p_flags;
00434 Elf32_Word p_align;
00435 } Elf32_Phdr;
00436
00437 typedef struct
00438 {
00439 Elf64_Word p_type;
00440 Elf64_Word p_flags;
00441 Elf64_Off p_offset;
00442 Elf64_Addr p_vaddr;
00443 Elf64_Addr p_paddr;
00444 Elf64_Xword p_filesz;
00445 Elf64_Xword p_memsz;
00446 Elf64_Xword p_align;
00447 } Elf64_Phdr;
00448
00449 enum Elf_PTs
00450 {
00451 PT_NULL = 0,
00452 PT_LOAD = 1,
00453 PT_DYNAMIC = 2,
00454 PT_INTERP = 3,
00455 PT_NOTE = 4,
00456 PT_SHLIB = 5,
00457 PT_PHDR = 6,
00458 PT_TLS = 7,
00459 PT_NUM = 8,
00460 PT_LOOS = 0x60000000,
00461 PT_HIOS = 0x6fffffff,
00462 PT_LOPROC = 0x70000000,
00463 PT_HIPROC = 0x7fffffff,
00464 PT_GNU_EH_FRAME = PT_LOOS + 0x474e550,
00465 PT_GNU_STACK = PT_LOOS + 0x474e551,
00466 PT_GNU_RELRO = PT_LOOS + 0x474e552,
00467 PT_L4_STACK = PT_LOOS + 0x12,
00468 PT_L4_AUX = PT_LOOS + 0x14,
00469 };
00470
00471 enum ELF_PFs
00472 {
00473 PF_X = 0x1,
00474 PF_W = 0x2,
00475 PF_R = 0x4,
00476 PF_MASKOS = 0x0ff00000,
00477 PF_MASKPROC = 0x7fffffff,
00478 };
00479
00480 enum Elf_NTs_core
00481 {
00482 NT_PRSTATUS = 1,
00483 NT_FPREGSET = 2,
00484 NT_PRPSINFO = 3,
00485 NT_PRXREG = 4,
00486 NT_TASKSTRUCT = 4,
00487 NT_PLATFORM = 5,
00488 NT_AUXV = 6,
00489 NT_GWINDOWS = 7,
00490 NT_ASRS = 8,
00491 NT_PSTATUS = 10,
00492 NT_PSINFO = 13,
00493 NT_PRCRED = 14,
00494 NT_UTSNAME = 15,
00495 NT_LWPSTATUS = 16,
00496 NT_LWPSINFO = 17,
00497 NT_PRFPXREG = 20,
00498 };
00499
00500 enum Elf_NTs_obj
00501 {
00502 NT_VERSION = 1,
00503 };
00504
00505 typedef struct
00506 {
00507 Elf32_Sword d_tag;
00508 union
00509 {
00510 Elf32_Word d_val;
00511 Elf32_Addr d_ptr;
00512 } d_un;
00513 } Elf32_Dyn;
00514
00515 typedef struct

```

```

00525 {
00526 Elf64_Sxword d_tag;
00527 union
00528 {
00529 Elf64_Xword d_val;
00530 Elf64_Addr d_ptr;
00531 } d_un;
00532 } Elf64_Dyn;
00533
00535 enum Elf_DTs
00536 {
00537 DT_NULL = 0,
00538 DT_NEEDED = 1,
00539 DT_PLTRELSZ = 2,
00540 DT_PLTGOT = 3,
00541 DT_HASH = 4,
00542 DT_STRTAB = 5,
00543 DT_SYMTAB = 6,
00544 DT_RELA = 7,
00545 DT_RELASZ = 8,
00546 DT_RELAENT = 9,
00547 DT_STRSZ = 10,
00548 DT_SYMENT = 11,
00549 DT_INIT = 12,
00550 DT_FINI = 13,
00551 DT_SONAME = 14,
00552 DT_RPATH = 15,
00553 DT_SYMBOLIC = 16,
00554 DT_REL = 17,
00555 DT_RELSZ = 18,
00556 DT_RELENT = 19,
00557 DT_PTRREL = 20,
00558 DT_DEBUG = 21,
00559 DT_TEXTREL = 22,
00560 DT_JMPREL = 23,
00561 DT_BIND_NOW = 24,
00562 DT_INIT_ARRAY = 25,
00563 DT_FINI_ARRAY = 26,
00564 DT_INIT_ARRAYSZ = 27,
00565 DT_FINI_ARRAYSZ = 28,
00566 DT_RUNPATH = 29,
00567 DT_FLAGS = 30,
00568 DT_ENCODING = 32,
00569 DT_PREINIT_ARRAY = 32,
00570 DT_PREINIT_ARRAYSZ = 33,
00571 DT_NUM = 34,
00572 DT_LOOS = 0x6000000d,
00573 DT_HIOS = 0x6ffff000,
00574 DT_LOPROC = 0x70000000,
00575 DT_HIPROC = 0x7fffffff,
00576 };
00577
00581 enum Elf_DFs
00582 {
00583 DF_ORIGIN = 0x00000001,
00584 DF_SYMBOLIC = 0x00000002,
00585 DF_TEXTREL = 0x00000004,
00586 DF_BIND_NOW = 0x00000008,
00587 DF_STATIC_TLS = 0x00000010,
00588 };
00589
00594 enum Elf_DF_1s
00595 {
00596 DF_1_NOW = 0x00000001,
00597 DF_1_GLOBAL = 0x00000002,
00598 DF_1_GROUP = 0x00000004,
00599 DF_1_NODELETE = 0x00000008,
00600 DF_1_LOADFLTR = 0x00000010,
00601 DF_1_INITFIRST = 0x00000020,
00602 DF_1_NOOPEN = 0x00000040,
00603 DF_1_ORIGIN = 0x00000080,
00604 DF_1_DIRECT = 0x00000100,
00605 DF_1_TRANS = 0x00000200,
00606 DF_1_INTERPOSE = 0x00000400,
00607 DF_1_NODEFLIB = 0x00000800,
00608 DF_1_NODUMP = 0x00001000,
00609 DF_1_CONFALT = 0x00002000,
00610 DF_1_ENDFILTEE = 0x00004000,
00611 DF_1_DISPRELDNE = 0x00008000,
00612 DF_1_DISPRELPND = 0x00010000,
00613 };
00614
00616 enum Elf_DTF_1s
00617 {
00618 DTF_1_PARINIT = 0x00000001,
00619 DTF_1_CONFEXP = 0x00000002,
00620 };

```

```

00621
00623 enum Elf_DF_Pls
00624 {
00625 DF_P1_LAZYLOAD = 0x00000001,
00626 DF_P1_GROUPPERM = 0x00000002,
00628 };
00629
00631 typedef struct
00632 {
00633 Elf32_Addr r_offset;
00634 Elf32_Word r_info;
00635 } Elf32_Rel;
00636
00638 typedef struct
00639 {
00640 Elf32_Addr r_offset;
00641 Elf32_Word r_info;
00642 Elf32_Sword r_addend;
00643 } Elf32_Rela;
00644
00646 typedef struct
00647 {
00648 Elf64_Addr r_offset;
00649 Elf64_Xword r_info;
00650 } Elf64_Rel;
00651
00653 typedef struct
00654 {
00655 Elf64_Addr r_offset;
00656 Elf64_Xword r_info;
00657 Elf64_Sxword r_addend;
00658 } Elf64_Rela;
00659
00661 #define ELF32_R_SYM(i) ((i)>>8)
00663 #define ELF32_R_TYPE(i) ((unsigned char)(i))
00665 #define ELF32_R_INFO(s,t) (((s)<<8)+(unsigned char)(t))
00666
00668 #define ELF64_R_SYM(i) ((i)>>32)
00669
00671 #define ELF64_R_TYPE(i) ((i)&0xffffffffL)
00672
00674 #define ELF64_R_INFO(s,t) (((s)<<32)+(t)&0xffffffffL)
00675
00677 enum Elf_R_386_s
00678 {
00679 R_386_NONE = 0,
00680 R_386_32 = 1,
00681 R_386_PC32 = 2,
00682 R_386_GOT32 = 3,
00683 R_386_PLT32 = 4,
00684 R_386_COPY = 5,
00685 R_386_GLOB_DAT = 6,
00686 R_386_JMP_SLOT = 7,
00687 R_386_RELATIVE = 8,
00688 R_386_GOTOFF = 9,
00689 R_386_GOTPC = 10,
00690 R_386_32PLT = 11,
00691 R_386_TLS_TPOFF = 14,
00692 R_386_TLS_IE = 15,
00694 R_386_TLS_GOTIE = 16,
00695 R_386_TLS_LE = 17,
00696 R_386_TLS_GD = 18,
00698 R_386_TLS_LDM = 19,
00700 R_386_16 = 20,
00701 R_386_PC16 = 21,
00702 R_386_8 = 22,
00703 R_386_PC8 = 23,
00704 R_386_TLS_GD_32 = 24,
00706 R_386_TLS_GD_PUSH = 25,
00707 R_386_TLS_GD_CALL = 26,
00709 R_386_TLS_GD_POP = 27,
00710 R_386_TLS_LDM_32 = 28,
00712 R_386_TLS_LDM_PUSH = 29,
00713 R_386_TLS_LDM_CALL = 30,
00715 R_386_TLS_LDM_POP = 31,
00716 R_386_TLS_LDO_32 = 32,
00717 R_386_TLS_IE_32 = 33,
00719 R_386_TLS_LE_32 = 34,
00721 R_386_TLS_DTPMOD32 = 35,
00722 R_386_TLS_DTPOFF32 = 36,
00723 R_386_TLS_TPOFF32 = 37,
00724 R_386_NUM = 38,
00725 };
00726
00728
00730 enum Elf_EF_ARM_s
00731 {

```

```

00732 EF_ARM_RELEXEC = 0x01,
00733 EF_ARM_HASENTRY = 0x02,
00734 EF_ARM_INTERWORK = 0x04,
00735 EF_ARM_APCS_26 = 0x08,
00736 EF_ARM_APCS_FLOAT = 0x10,
00737 EF_ARM_PIC = 0x20,
00738 EF_ARM_ALIGN8 = 0x40,
00739 EF_ARM_NEW_ABI = 0x80,
00740 EF_ARM_OLD_ABI = 0x100,
00741
00742 /* Other constants defined in the ARM ELF spec. version B-01. */
00743 /* NB. These conflict with values defined above. */
00744 EF_ARM_SYMSARESORTED = 0x04,
00745 EF_ARM_DYNSYMSUSESEGIDX = 0x08,
00746 EF_ARM_MAPSYMSFIRST = 0x10,
00747 EF_ARM_EABIMASK = 0xFF000000,
00748
00749 #define EF_ARM_EABI_VERSION(flags) ((flags) & EF_ARM_EABIMASK)
00750 EF_ARM_EABI_UNKNOWN = 0x00000000,
00751 EF_ARM_EABI_VER1 = 0x01000000,
00752 EF_ARM_EABI_VER2 = 0x02000000,
00753 };
00754
00756 enum Elf_STT_ARM_s
00757 {
00758 STT_ARM_TFUNC = 0xd,
00759 };
00760
00762 enum Elf_SHF_s_ARM
00763 {
00764 SHF_ARM_ENTRYSECT = 0x10000000,
00765 SHF_ARM_COMDEF = 0x80000000,
00766 };
00767
00770 enum Elf_ARM_SBs
00771 {
00772 PF_ARM_SB = 0x10000000,
00773 };
00774
00775
00777 enum Elf_R_ARM_s
00778 {
00779 R_ARM_NONE = 0,
00780 R_ARM_PC24 = 1,
00781 R_ARM_ABS32 = 2,
00782 R_ARM_REL32 = 3,
00783 R_ARM_PC13 = 4,
00784 R_ARM_ABS16 = 5,
00785 R_ARM_ABS12 = 6,
00786 R_ARM_THM_ABS5 = 7,
00787 R_ARM_ABS8 = 8,
00788 R_ARM_SBREL32 = 9,
00789 R_ARM_THM_PC22 = 10,
00790 R_ARM_THM_PC8 = 11,
00791 R_ARM AMP_VCALL9 = 12,
00792 R_ARM_SWI24 = 13,
00793 R_ARM_THM_SWI8 = 14,
00794 R_ARM_XPC25 = 15,
00795 R_ARM_THM_XPC22 = 16,
00796 R_ARM_COPY = 20,
00797 R_ARM_GLOB_DAT = 21,
00798 R_ARM_JUMP_SLOT = 22,
00799 R_ARM_RELATIVE = 23,
00800 R_ARM_GOTOFF = 24,
00801 R_ARM_GOTPC = 25,
00802 R_ARM_GOT32 = 26,
00803 R_ARM_PLT32 = 27,
00804 R_ARM_ALU_PCREL_7_0 = 32,
00805 R_ARM_ALU_PCREL_15_8 = 33,
00806 R_ARM_ALU_PCREL_23_15 = 34,
00807 R_ARM_LDR_SBREL_11_0 = 35,
00808 R_ARM_ALU_SBREL_19_12 = 36,
00809 R_ARM_ALU_SBREL_27_20 = 37,
00810 R_ARM_GNU_VTENTRY = 100,
00811 R_ARM_GNU_VTINHERIT = 101,
00812 R_ARM_THM_PC11 = 102,
00813 R_ARM_THM_PC9 = 103,
00814 R_ARM_RXPC25 = 249,
00815 R_ARM_RSBREL32 = 250,
00816 R_ARM_THM_RPC22 = 251,
00817 R_ARM_RREL32 = 252,
00818 R_ARM_RABS22 = 253,
00819 R_ARM_RPC24 = 254,
00820 R_ARM_RBASE = 255,
00821 R_ARM_NUM = 256,
00822 };
00823
00825 enum Elf_R_AARCH64_s

```

```

00826 {
00827 R_AARCH64_NONE = 0,
00828 R_AARCH64_RELATIVE = 1027,
00829 };
00830
00832 enum Elf_R_X86_64_s
00833 {
00834 R_X86_64_NONE = 0,
00835 R_X86_64_64 = 1,
00836 R_X86_64_PC32 = 2,
00837 R_X86_64_GOT32 = 3,
00838 R_X86_64_PLT32 = 4,
00839 R_X86_64_COPY = 5,
00840 R_X86_64_GLOB_DAT = 6,
00841 R_X86_64_JUMP_SLOT = 7,
00842 R_X86_64_RELATIVE = 8,
00843 R_X86_64_GOTPCREL = 9,
00844 R_X86_64_32 = 10,
00845 R_X86_64_32S = 11,
00846 R_X86_64_16 = 12,
00847 R_X86_64_PC16 = 13,
00848 R_X86_64_8 = 14,
00849 R_X86_64_PC8 = 15,
00850 R_X86_64_DTPMOD64 = 16,
00851 R_X86_64_DTPOFF64 = 17,
00852 R_X86_64_TPOFF64 = 18,
00853 R_X86_64_TLSGD = 19,
00855 R_X86_64_TLSLD = 20,
00857 R_X86_64_DTPOFF32 = 21,
00858 R_X86_64_GOTTPOFF = 22,
00860 R_X86_64_TPOFF32 = 23,
00861 R_X86_64_NUM = 24,
00862 };
00863
00865 enum Elf_STNs
00866 {
00867 STN_UNDEF = 0,
00868 };
00869
00871 typedef struct
00872 {
00873 Elf32_Word st_name;
00874 Elf32_Addr st_value;
00875 Elf32_Word st_size;
00876 unsigned char st_info;
00877 unsigned char st_other;
00878 Elf32_Half st_shndx;
00879 } Elf32_Sym;
00880
00882 typedef struct
00883 {
00884 Elf64_Word st_name;
00885 unsigned char st_info;
00886 unsigned char st_other;
00887 Elf64_Half st_shndx;
00888 Elf64_Addr st_value;
00889 Elf64_Xword st_size;
00890 } Elf64_Sym;
00891
00893 #define ELF32_ST_BIND(i) ((i)>>4)
00894
00896 #define ELF32_ST_TYPE(i) ((i)&0xf)
00897
00899 #define ELF32_ST_INFO(b,t) (((b)<<4)+((t)&0xf))
00900
00902 #define ELF64_ST_BIND(i) ((i)>>4)
00903
00905 #define ELF64_ST_TYPE(i) ((i)&0xf)
00906
00908 #define ELF64_ST_INFO(b,t) (((b)<<4)+((t)&0xf))
00909
00912 enum Elf_STBs
00913 {
00914 STB_LOCAL = 0,
00915 STB_GLOBAL = 1,
00916 STB_WEAK = 2,
00917 STB_LOOS = 10,
00918 STB_HIOS = 12,
00919 STB_LOPROC = 13,
00920 STB_HIPROC = 15,
00921 };
00922
00925 enum Elf_STTs
00926 {
00927 STT_NOTYPE = 0,
00928 STT_OBJECT = 1,
00929 STT_FUNC = 2,

```

```

00930 STT_SECTION = 3,
00931 STT_FILE = 4,
00932 STT_LOOS = 10,
00933 STT_HIOS = 12,
00934 STT_LOPROC = 13,
00935 STT_HIPROC = 15,
00936 };
00937
00939 enum Elf_ATs
00940 {
00941 AT_NULL = 0,
00942 AT_IGNORE = 1,
00943 AT_EXECD = 2,
00944 AT_PHDR = 3,
00945 AT_PHEMT = 4,
00946 AT_PHNUM = 5,
00947 AT_PAGESZ = 6,
00948 AT_BASE = 7,
00949 AT_FLAGS = 8,
00950 AT_ENTRY = 9,
00951 AT_NOTELF = 10,
00952 AT_UID = 11,
00953 AT_EUID = 12,
00954 AT_GID = 13,
00955 AT_EGID = 14,
00956
00957 AT_L4_AUX = 0xf0,
00958 AT_L4_ENV = 0xf1,
00959 };
00960
00962 typedef struct Elf32_Auxv
00963 {
00964 Elf32_Word atype;
00965 Elf32_Word avalue;
00966 } Elf32_Auxv;
00967
00969 typedef struct Elf64_Auxv
00970 {
00971 Elf64_Word atype;
00972 Elf64_Word avalue;
00973 } Elf64_Auxv;
00974
00975 typedef struct
00976 {
00977 Elf32_Word n_namesz;
00978 Elf32_Word n_descsz;
00979 Elf32_Word n_type;
00980 } Elf32_Nhdr;
00981
00982 typedef struct
00983 {
00984 Elf64_Word n_namesz;
00985 Elf64_Word n_descsz;
00986 Elf64_Word n_type;
00987 } Elf64_Nhdr;
00988
00996 static inline int l4util_elf_check_magic(ElfW(Ehdr) const *hdr);
00997
01005 static inline int l4util_elf_check_arch(ElfW(Ehdr) const *hdr);
01006
01013 static inline ElfW(Phdr) *l4util_elf_phdr(ElfW(Ehdr) const *hdr);
01014
01015
01016 /* Implementations */
01017
01018 static inline
01019 int l4util_elf_check_magic(ElfW(Ehdr) const *hdr)
01020 {
01021 return hdr->e_ident[EI_MAG0] == ELFMAG0
01022 && hdr->e_ident[EI_MAG1] == ELFMAG1
01023 && hdr->e_ident[EI_MAG2] == ELFMAG2
01024 && hdr->e_ident[EI_MAG3] == ELFMAG3;
01025 }
01026
01027 static inline
01028 int l4util_elf_check_arch(ElfW(Ehdr) const *hdr)
01029 {
01030 return hdr->e_ident[EI_CLASS] == L4_ARCH_EI_CLASS
01031 && hdr->e_ident[EI_DATA] == L4_ARCH_EI_DATA
01032 && hdr->e_machine == L4_ARCH_E_MACHINE;
01033 }
01034
01035 static inline
01036 ElfW(Phdr) *l4util_elf_phdr(ElfW(Ehdr) const *hdr)
01037 {
01038 return (ElfW(Phdr) *) ((char *)hdr + hdr->e_phoff);
01039 }

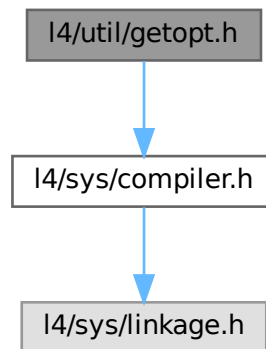
```

## 17.641 l4/util/getopt.h File Reference

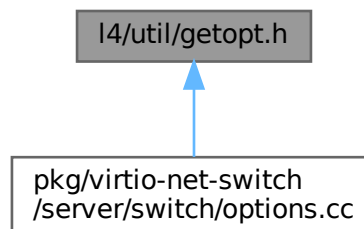
getopt

```
#include <l4/sys/compiler.h>
```

Include dependency graph for getopt.h:



This graph shows which files directly or indirectly include this file:



### 17.641.1 Detailed Description

getopt

Definition in file [getopt.h](#).

## 17.642 getopt.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #ifndef _GETOPT_H
00012 #define _GETOPT_H
00013
00014 #ifndef NULL
00015 #define NULL 0
00016 #endif
00017
00018 #include <l4/sys/compiler.h>
00019
00020 L4_BEGIN_DECLS
00021
00022 /* For communication from `getopt' to the caller.
00023 * When `getopt' finds an option that takes an argument,
00024 * the argument value is returned here.
00025 * Also, when `ordering' is RETURN_IN_ORDER,
00026 * each non-option ARGV-element is returned here. */
00027
00028 extern char *optarg;
00029
00030 /* Index in ARGV of the next element to be scanned.
00031 * This is used for communication to and from the caller
00032 * and for communication between successive calls to `getopt'.
00033 *
00034 * On entry to `getopt', zero means this is the first call; initialize.
00035 *
00036 * When `getopt' returns -1, this is the index of the first of the
00037 * non-option elements that the caller should itself scan.
00038 *
00039 * Otherwise, `optind' communicates from one call to the next
00040 * how much of ARGV has been scanned so far. */
00041
00042 extern int optind;
00043
00044 /* Callers store zero here to inhibit the error message `getopt' prints
00045 * for unrecognized options. */
00046
00047 extern int opterr;
00048
00049 /* Set to an option character which was unrecognized. */
00050
00051 extern int optopt;
00052
00053 /* Describe the long-named options requested by the application.
00054 * The LONG_OPTIONS argument to getopt_long or getopt_long_only is a vector
00055 * of `struct option' terminated by an element containing a name which is
00056 * zero.
00057 *
00058 * The field `has_arg' is:
00059 * no_argument (or 0) if the option does not take an argument,
00060 * required_argument (or 1) if the option requires an argument,
00061 * optional_argument (or 2) if the option takes an optional argument.
00062 *
00063 * If the field `flag' is not NULL, it points to a variable that is set
00064 * to the value given in the field `val' when the option is found, but
00065 * left unchanged if the option is not found.
00066 *
00067 * To have a long-named option do something other than set an `int' to
00068 * a compiled-in constant, such as set a value from `optarg', set the
00069 * option's `flag' field to zero and its `val' field to a nonzero
00070 * value (the equivalent single-letter option character, if there is
00071 * one). For long options that have a zero `flag' field, `getopt'
00072 * returns the contents of the `val' field. */
00073
00074 struct option
00075 {
00076 const char *name;
00077 /* has_arg can't be an enum because some compilers complain about
00078 * type mismatches in all the code that assumes it is an int. */
00079 int has_arg;
00080 int *flag;
00081 int val;
00082 };
00083
00084 /* Names for the values of the `has_arg' field of `struct option'. */
00085

```



```

00086 #define no_argument 0
00087 #define required_argument 1
00088 #define optional_argument 2
00089
00090 L4_CV int getopt (int argc, char *const *argv, const char *shortopts);
00091
00092 L4_CV int getopt_long (int argc, char *const *argv, const char *shortopts,
00093 const struct option *longopts, int *longind);
00094 L4_CV int getopt_long_only (int argc, char *const *argv,
00095 const char *shortopts,
00096 const struct option *longopts, int *longind);
00097
00098 L4_CV int _getopt_internal (int argc, char *const *argv,
00099 const char *shortopts,
00100 const struct option *longopts, int *longind,
00101 int long_only);
00102
00103 L4_END_DECLS
00104
00105 #endif /* _GETOPT_H */

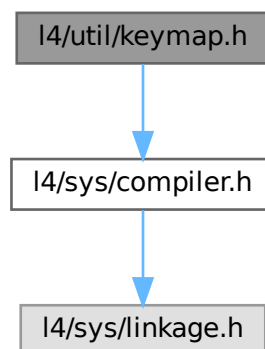
```

## 17.643 l4/util/keymap.h File Reference

Event to ASCII key mapping.

```
#include <l4/sys/compiler.h>
```

Include dependency graph for keymap.h:



### 17.643.1 Detailed Description

Event to ASCII key mapping.

Definition in file [keymap.h](#).

## 17.644 keymap.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010 #ifndef __L4UTIL__KEYMAP_H__
00011 #define __L4UTIL__KEYMAP_H__
00012
00013 #include <l4/sys/compiler.h>
00014
00015 L4_BEGIN_DECLS
00016
00017 int l4util_map_event_to_keymap(unsigned value, unsigned shift);
00018
00019 L4_END_DECLS
00020
00021
00022 #endif /* __L4UTIL__KEYMAP_H__ */

```

## 17.645 l4/sys/kip.h File Reference

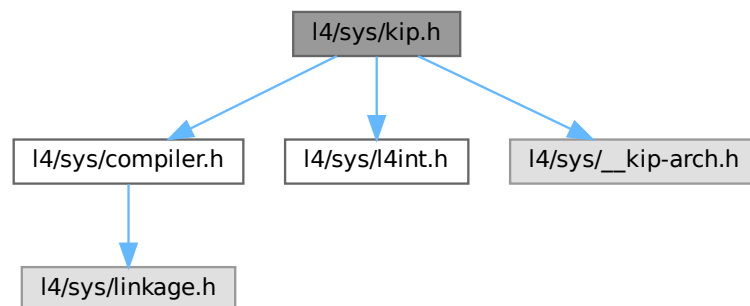
Kernel Info Page access functions.

```

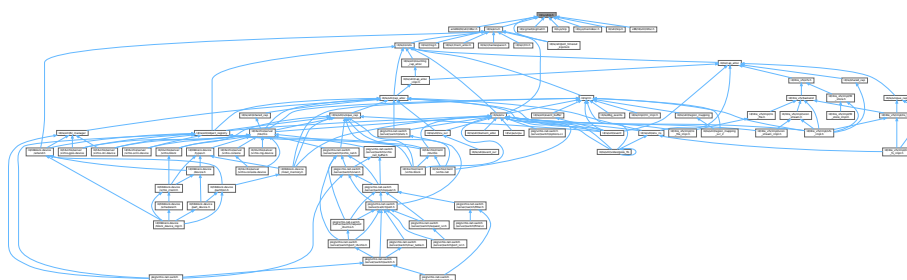
#include <l4/sys/compiler.h>
#include <l4/sys/l4int.h>
#include <l4/sys/__kip-arch.h>

```

Include dependency graph for kip.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [l4\\_kernel\\_info\\_t](#)  
*L4 Kernel Interface Page.*

## Macros

- #define **L4\_KERNEL\_INFO\_MAGIC** (0x4BE6344CL) /\* "L4μK" \*/  
*Kernel Info Page identifier ("L4μK").*
- #define **l4\_kip\_for\_each\_feature**(s)  
*Cycle through kernel features given in the KIP.*

## Typedefs

- typedef struct [l4\\_kernel\\_info\\_t](#) [l4\\_kernel\\_info\\_t](#)  
*L4 Kernel Interface Page.*

## Enumerations

- enum { [L4\\_KIP\\_OFFS\\_READ\\_US](#) = 0x900 , [L4\\_KIP\\_OFFS\\_READ\\_NS](#) = 0x980 }

## Functions

- [l4\\_kernel\\_info\\_t](#) const \* [l4\\_kip](#) (void) [L4\\_NOTHROW](#)  
*Get Kernel Info Page.*
- [l4\\_umword\\_t](#) [l4\\_kip\\_version](#) ([l4\\_kernel\\_info\\_t](#) const \*kip) [L4\\_NOTHROW](#)  
*Get the kernel version.*
- const char \* [l4\\_kip\\_version\\_string](#) ([l4\\_kernel\\_info\\_t](#) const \*kip) [L4\\_NOTHROW](#)  
*Get the kernel version string.*
- int [l4\\_kernel\\_info\\_version\\_offset](#) ([l4\\_kernel\\_info\\_t](#) const \*kip) [L4\\_NOTHROW](#)  
*Return offset in bytes of version\_strings relative to the KIP base.*
- [l4\\_cpu\\_time\\_t](#) [l4\\_kip\\_clock](#) ([l4\\_kernel\\_info\\_t](#) const \*kip) [L4\\_NOTHROW](#)  
*Return clock value from the KIP.*
- [l4\\_umword\\_t](#) [l4\\_kip\\_clock\\_lw](#) ([l4\\_kernel\\_info\\_t](#) const \*kip) [L4\\_NOTHROW](#)  
*Return least significant machine word of clock value from the KIP.*
- [l4\\_uint64\\_t](#) [l4\\_kip\\_clock\\_ns](#) ([l4\\_kernel\\_info\\_t](#) const \*kip) [L4\\_NOTHROW](#)  
*Return current clock using the KIP in nanoseconds.*
- int [l4\\_kip\\_kernel\\_has\\_feature](#) ([l4\\_kernel\\_info\\_t](#) const \*kip, char const \*str)  
*Check if kernel supports a feature.*

## 17.645.1 Detailed Description

Kernel Info Page access functions.

Definition in file [kip.h](#).

## 17.645.2 Macro Definition Documentation

### 17.645.2.1 l4\_kip\_for\_each\_feature

```
#define l4_kip_for_each_feature(
 s)
```

**Value:**

```
for (s += __builtin_strlen(s) + 1; *s; s += __builtin_strlen(s) + 1)
```

Cycle through kernel features given in the KIP.

Cycles through all KIP kernel feature strings. *s* must be a character pointer (`char const *`) initialized with [l4\\_kip\\_version\\_string\(\)](#).

Definition at line 272 of file [kip.h](#).

Referenced by [l4\\_kip\\_kernel\\_has\\_feature\(\)](#).

## 17.645.3 Function Documentation

### 17.645.3.1 l4\_kip\_kernel\_has\_feature()

```
int l4_kip_kernel_has_feature (
 l4_kernel_info_t const * kip,
 char const * str) [inline]
```

Check if kernel supports a feature.

**Parameters**

|            |                                        |
|------------|----------------------------------------|
| <i>kip</i> | Pointer to the kernel info page (KIP). |
| <i>str</i> | Feature name to check.                 |

**Returns**

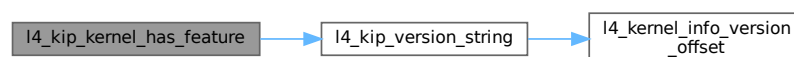
1 if the kernel supports the feature, 0 if not.

Checks the feature field in the KIP for the given string.

Definition at line 286 of file [kip.h](#).

References [l4\\_kip\\_for\\_each\\_feature](#), and [l4\\_kip\\_version\\_string\(\)](#).

Here is the call graph for this function:



## 17.646 kip.h

[Go to the documentation of this file.](#)

```

00001
00006 /*
00007 * (c) 2008-2013 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>
00009 * economic rights: Technische Universität Dresden (Germany)
00010 *
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013 #pragma once
00014
00015 #include <l4/sys/compiler.h>
00016 #include <l4/sys/l4int.h>
00017
00018 #include <l4/sys/__kip-arch.h>
00019
00023 struct l4_kip_platform_info
00024 {
00025 char name[16];
00026 l4_uint32_t is_mp;
00027 struct l4_kip_platform_info_arch arch;
00028 };
00029
00036 typedef struct l4_kernel_info_t
00037 {
00038 /* offset 0x00 */
00039 l4_uint32_t magic;
00042 l4_uint32_t version;
00043 l4_uint8_t offset_version_strings;
00044 l4_uint8_t _fill0[3];
00045 l4_uint8_t kip_sys_calls;
00046 l4_uint8_t node;
00047 l4_uint8_t _fill1[2];
00048
00049 /* offset 0x10 */
00050 l4_uint64_t sigma0_ip;
00051 l4_uint64_t root_ip;
00052
00053 /* offset 0x20 */
00054 volatile l4_cpu_time_t _clock_val;
00055 l4_uint64_t frequency_cpu;
00056
00057 /* offset 0x30 */
00058 l4_uint64_t acpi_rsdp_addr;
00059 l4_uint64_t dt_addr;
00060
00061 /* offset 0x40 */
00062 l4_uint64_t user_ptr;
00063 l4_uint64_t _res0[1];
00064
00065 /* offset 0x50 */
00066 l4_uint32_t scheduler_granularity;
00067 l4_uint32_t mem_descs;
00068 l4_uint32_t mem_descs_num;
00069 l4_uint32_t _res1[1];
00070
00071 /* offset 0x60 */
00072 l4_uint64_t _res2[2];
00073
00074 /* offset 0x70 */
00075 struct l4_kip_platform_info platform_info;
00076 } l4_kernel_info_t;
00077
00085
00089 enum l4_kernel_info_consts_t
00090 {
00091 L4_KIP_VERSION_FIASCO = 0x87004444,
00092 L4_KIP_VERSION_FIASCO_MASK = 0xff00ffff,
00093 };
00094
00095 enum
00096 {
00105 L4_KIP_OFFS_READ_US = 0x900,
00106
00116 L4_KIP_OFFS_READ_NS = 0x980,
00117 };
00118
00122 extern l4_kernel_info_t const *l4_global_kip;
00123
00127 #define L4_KERNEL_INFO_MAGIC (0x4BE6344CL) /* "L4µK" */
00128
00129
00135 L4_INLINE l4_kernel_info_t const *l4_kip(void) L4_NOTHROW;

```

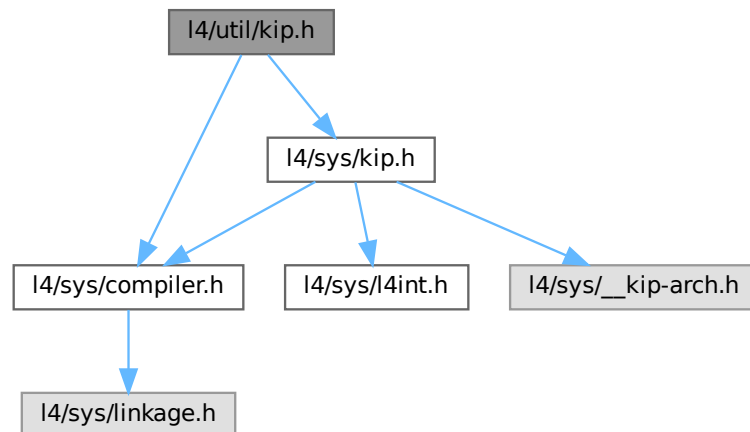
```

00136
00137
00145 L4_INLINE l4_umword_t l4_kip_version(l4_kernel_info_t const *kip) L4_NOTHROW;
00146
00154 L4_INLINE const char *l4_kip_version_string(l4_kernel_info_t const *kip) L4_NOTHROW;
00155
00164 L4_INLINE int
00165 l4_kernel_info_version_offset(l4_kernel_info_t const *kip) L4_NOTHROW;
00166
00184 L4_INLINE l4_cpu_time_t
00185 l4_kip_clock(l4_kernel_info_t const *kip) L4_NOTHROW;
00186
00199 L4_INLINE l4_umword_t
00200 l4_kip_clock_lw(l4_kernel_info_t const *kip) L4_NOTHROW
00201 L4_DEPRECATED("Use l4_kip_clock() instead");
00202
00216 L4_INLINE l4_uint64_t
00217 l4_kip_clock_ns(l4_kernel_info_t const *kip) L4_NOTHROW;
00218
00220
00221 /*****
00222 * Implementations
00223 *****/
00224
00225 L4_INLINE l4_kernel_info_t const*
00226 l4_kip(void) L4_NOTHROW
00227 { return l4_global_kip; }
00228
00229 L4_INLINE l4_umword_t
00230 l4_kip_version(l4_kernel_info_t const *kip) L4_NOTHROW
00231 { return kip->version & L4_KIP_VERSION_FIASCO_MASK; }
00232
00233 L4_INLINE const char*
00234 l4_kip_version_string(l4_kernel_info_t const *k) L4_NOTHROW
00235 { return (const char *)k + l4_kernel_info_version_offset(k); }
00236
00237 L4_INLINE int
00238 l4_kernel_info_version_offset(l4_kernel_info_t const *kip) L4_NOTHROW
00239 { return kip->offset_version_strings « 4; }
00240
00241 L4_INLINE l4_cpu_time_t
00242 l4_kip_clock(l4_kernel_info_t const *kip) L4_NOTHROW
00243 {
00244 // Use kernel-provided code to determine the current clock.
00245 typedef l4_uint64_t (*kip_time_fn_read_us)(void);
00246 kip_time_fn_read_us read_us =
00247 (kip_time_fn_read_us)((l4_uint8_t const*)kip + L4_KIP_OFFS_READ_US);
00248 return read_us();
00249 }
00250
00251 L4_INLINE l4_cpu_time_t
00252 l4_kip_clock_ns(l4_kernel_info_t const *kip) L4_NOTHROW
00253 {
00254 typedef l4_uint64_t (*kip_time_fn_read_ns)(void);
00255 kip_time_fn_read_ns read_ns =
00256 (kip_time_fn_read_ns)((l4_uint8_t const*)kip + L4_KIP_OFFS_READ_NS);
00257 return read_ns();
00258 }
00259
00260 L4_INLINE l4_umword_t
00261 l4_kip_clock_lw(l4_kernel_info_t const *kip) L4_NOTHROW
00262 {
00263 return l4_kip_clock(kip);
00264 }
00265
00272 #define l4_kip_for_each_feature(s) \
00273 for (s += __builtin_strlen(s) + 1; *s; s += __builtin_strlen(s) + 1)
00274
00285 L4_INLINE int
00286 l4_kip_kernel_has_feature(l4_kernel_info_t const *kip, char const *str)
00287 {
00288 const char *s = l4_kip_version_string(kip);
00289 if (!s)
00290 return 0;
00291 l4_kip_for_each_feature(s)
00292 {
00293 if (__builtin_strcmp(s, str) == 0)
00294 return 1;
00295 }
00296 return 0;
00297 }
00298
00299 }

```

## 17.647 l4/util/kip.h File Reference

```
#include <l4/sys/kip.h>
#include <l4/sys/compiler.h>
Include dependency graph for kip.h:
```



### Macros

- `#define l4util_kip_for_each_feature(s)`  
Cycle through kernel features given in the KIP.

### Functions

- `L4_BEGIN_DECLS int l4util_kip_kernel_has_feature (l4_kernel_info_t const *k, char const *str)`  
Check if kernel supports a feature.
- `unsigned long l4util_kip_kernel_abi_version (l4_kernel_info_t const *k)`  
Return kernel ABI version.

## 17.648 kip.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011
00012 #pragma once
00013
00014 #include <l4/sys/kip.h>
00015 #include <l4/sys/compiler.h>
00016
00022
```

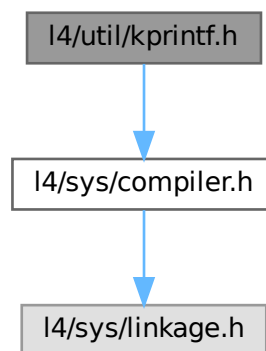
```
00023
00024 L4_BEGIN_DECLS
00025
00038 L4_CV int l4util_kip_kernel_has_feature(l4_kernel_info_t const *k, char const *str);
00039
00046 L4_CV unsigned long l4util_kip_kernel_abi_version(l4_kernel_info_t const *k);
00047
00048 L4_END_DECLS
00049
00058 #define l4util_kip_for_each_feature(s) l4_kip_for_each_feature(s)
00059
00061
```

## 17.649 l4/util/kprintf.h File Reference

printf using the kernel debugger

```
#include <l4/sys/compiler.h>
```

Include dependency graph for kprintf.h:



### 17.649.1 Detailed Description

printf using the kernel debugger

Date

04/05/2007

Author

Adam Lackorzynski [adam@os.inf.tu-dresden.de](mailto:adam@os.inf.tu-dresden.de),

Definition in file [kprintf.h](#).



## 17.650 kprintf.h

[Go to the documentation of this file.](#)

```

00001 /*****
00009 */
00010 * (c) 2007-2009 Author(s)
00011 * economic rights: Technische Universität Dresden (Germany)
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014
00015 #ifndef __L4UTIL__INCLUDE__KPRINTF_H__
00016 #define __L4UTIL__INCLUDE__KPRINTF_H__
00017
00018 #include <l4/sys/compiler.h>
00019
00020 L4_BEGIN_DECLS
00021
00022 L4_CV int l4_kprintf(const char *fmt, ...)
00023 __attribute__((format (printf, 1, 2)));
00024
00025 L4_END_DECLS
00026
00027 #endif /* ! __L4UTIL__INCLUDE__KPRINTF_H__ */

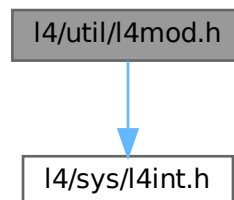
```

## 17.651 l4/util/l4mod.h File Reference

L4mod structures and constants.

```
#include <l4/sys/l4int.h>
```

Include dependency graph for l4mod.h:



### Data Structures

- struct [l4util\\_l4mod\\_mod](#)  
*A single module.*
- struct [l4util\\_l4mod\\_info](#)  
*Base module structure.*

### Enumerations

- enum [l4util\\_l4mod\\_mod\\_info\\_flag](#) {  
[L4util\\_l4mod\\_mod\\_flag\\_unspec](#) = 0 , [L4util\\_l4mod\\_mod\\_flag\\_kernel](#) = 1 , [L4util\\_l4mod\\_mod\\_flag\\_sigma0](#) =  
2 , [L4util\\_l4mod\\_mod\\_flag\\_roottask](#) = 3 ,  
[L4util\\_l4mod\\_mod\\_flag\\_mask](#) = 7 << 0 }  
*Flags for l4util\_l4mod\_mod.flags.*

### 17.651.1 Detailed Description

L4mod structures and constants.

Definition in file [l4mod.h](#).

### 17.651.2 Enumeration Type Documentation

#### 17.651.2.1 l4util\_l4mod\_mod\_info\_flag

enum [l4util\\_l4mod\\_mod\\_info\\_flag](#)

Flags for [l4util\\_l4mod\\_mod.flags](#).

#### Enumerator

|                                |                                |
|--------------------------------|--------------------------------|
| L4util_l4mod_mod_flag_unspec   | Flag for a generic module.     |
| L4util_l4mod_mod_flag_kernel   | Flag for the kernel module.    |
| L4util_l4mod_mod_flag_sigma0   | Flag for the sigma0 module.    |
| L4util_l4mod_mod_flag_roottask | Flag for the root task module. |
| L4util_l4mod_mod_flag_mask     | Mask for specified flags.      |

Definition at line 17 of file [l4mod.h](#).

## 17.652 l4mod.h

[Go to the documentation of this file.](#)

```

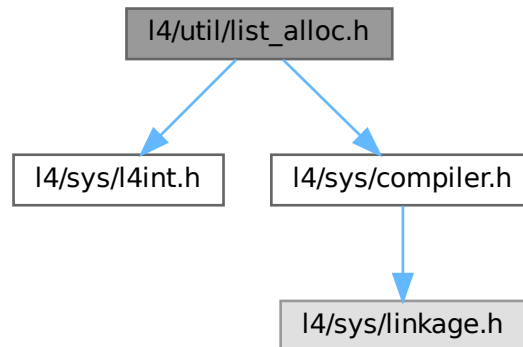
00001 /*
00002 * Copyright (C) 2021-2022, 2024 Kernkonzept GmbH.
00003 * Author(s): Adam Lackorzynski <adam@l4re.org>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00012 #pragma once
00013
00014 #include <l4/sys/l4int.h>
00015
00017 enum l4util_l4mod_mod_info_flag
00018 {
00019 L4util_l4mod_mod_flag_unspec = 0,
00020 L4util_l4mod_mod_flag_kernel = 1,
00021 L4util_l4mod_mod_flag_sigma0 = 2,
00022 L4util_l4mod_mod_flag_roottask = 3,
00023 L4util_l4mod_mod_flag_mask = 7 « 0,
00024 };
00025
00027 typedef struct
00028 {
00029 l4_uint64_t flags;
00030 l4_uint64_t mod_start;
00031 l4_uint64_t mod_end;
00032 l4_uint64_t cmdline;
00033 } l4util_l4mod_mod;
00034
00036 typedef struct
00037 {
00038 l4_uint64_t flags;
00039 l4_uint64_t cmdline;
00040 l4_uint64_t mods_addr;
00041 l4_uint32_t mods_count;
00042 l4_uint32_t _pad;
00043
00048 l4_uint64_t vbe_ctrl_info;
00049 l4_uint64_t vbe_mode_info;
00050 } l4util_l4mod_info;

```

## 17.653 l4/util/list\_alloc.h File Reference

Simple list-based allocator.

```
#include <l4/sys/l4int.h>
#include <l4/sys/compiler.h>
Include dependency graph for list_alloc.h:
```



### Functions

- [L4\\_BEGIN\\_DECLS](#) void [l4la\\_free](#) (l4la\_free\_t \*\*first, void \*block, [l4\\_size\\_t](#) size)  
*Add free memory to memory pool.*
- void \* [l4la\\_alloc](#) (l4la\_free\_t \*\*first, [l4\\_size\\_t](#) size, unsigned align)  
*Allocate memory from pool.*
- void [l4la\\_dump](#) (l4la\_free\_t \*\*first)  
*Show all list members.*
- void [l4la\\_init](#) (l4la\_free\_t \*\*first)  
*Init memory pool.*
- [l4\\_size\\_t](#) [l4la\\_avail](#) (l4la\_free\_t \*\*first)  
*Show available memory in pool.*

### 17.653.1 Detailed Description

Simple list-based allocator.

Taken from the Fiasco kernel.

#### Date

Alexander Warg <aw11os.inf.tu-dresden.de> Frank Mehnert [fm3@os.inf.tu-dresden.de](mailto:fm3@os.inf.tu-dresden.de)

Definition in file [list\\_alloc.h](#).

## 17.653.2 Function Documentation

### 17.653.2.1 l4la\_alloc()

```
void * l4la_alloc (
 l4la_free_t ** first,
 l4_size_t size,
 unsigned align)
```

Allocate memory from pool.

#### Parameters

|              |                                    |
|--------------|------------------------------------|
| <i>first</i> | list identifier                    |
| <i>size</i>  | length of memory block to allocate |
| <i>align</i> | alignment                          |

References [L4\\_CV](#).

### 17.653.2.2 l4la\_avail()

```
l4_size_t l4la_avail (
 l4la_free_t ** first)
```

Show available memory in pool.

#### Parameters

|              |                 |
|--------------|-----------------|
| <i>first</i> | list identifier |
|--------------|-----------------|

References [L4\\_CV](#), and [L4\\_END\\_DECLS](#).

### 17.653.2.3 l4la\_dump()

```
void l4la_dump (
 l4la_free_t ** first)
```

Show all list members.

#### Parameters

|              |                 |
|--------------|-----------------|
| <i>first</i> | list identifier |
|--------------|-----------------|

References [L4\\_CV](#).

### 17.653.2.4 l4la\_free()

```
L4_BEGIN_DECLS void l4la_free (
 l4la_free_t ** first,
 void * block,
 l4_size_t size)
```

Add free memory to memory pool.

#### Parameters

---

|              |                                |
|--------------|--------------------------------|
| <i>first</i> | list identifier                |
| <i>block</i> | address of unused memory block |
| <i>size</i>  | size of memory block           |

References [L4\\_CV](#).

### 17.653.2.5 l4la\_init()

```
void l4la_init (
 l4la_free_t ** first)
```

Init memory pool.

#### Parameters

|              |                 |
|--------------|-----------------|
| <i>first</i> | list identifier |
|--------------|-----------------|

References [L4\\_CV](#).

## 17.654 list\_alloc.h

[Go to the documentation of this file.](#)

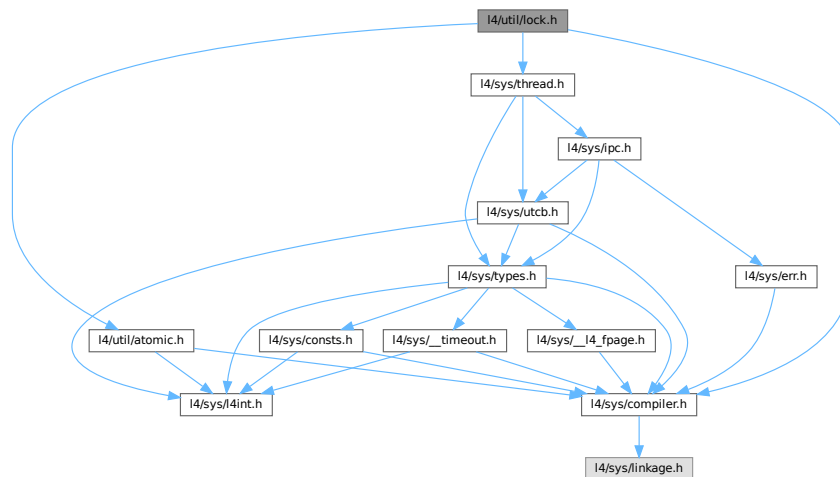
```
00001
00007
00008 /*
00009 * (c) 2003-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00010 * Frank Mehnert <fm3@os.inf.tu-dresden.de>
00011 * economic rights: Technische Universität Dresden (Germany)
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014
00015 #ifndef L4UTIL_L4LA_H
00016 #define L4UTIL_L4LA_H
00017
00018 #include <l4/sys/l4int.h>
00019 #include <l4/sys/compiler.h>
00020
00021 typedef struct l4la_free_t_s
00022 {
00023 struct l4la_free_t_s *next;
00024 l4_size_t size;
00025 } l4la_free_t;
00026
00027 #define L4LA_INITIALIZER { 0 }
00028
00029 L4_BEGIN_DECLS
00030
00035 L4_CV void l4la_free(l4la_free_t **first, void *block, l4_size_t size);
00036
00041 L4_CV void* l4la_alloc(l4la_free_t **first, l4_size_t size, unsigned align);
00042
00045 L4_CV void l4la_dump(l4la_free_t **first);
00046
00049 L4_CV void l4la_init(l4la_free_t **first);
00050
00053 L4_CV l4_size_t l4la_avail(l4la_free_t **first);
00054
00055 L4_END_DECLS
00056
00057 #endif
```

## 17.655 I4/util/lock.h File Reference

Simple lock implementation.

```
#include <l4/sys/thread.h>
#include <l4/sys/compiler.h>
#include <l4/util/atomic.h>
```

Include dependency graph for lock.h:



### 17.655.1 Detailed Description

Simple lock implementation.

Does only work if all thread have the same priority!

Date

02/1997

Author

Michael Hohmuth [hohmuth@os.inf.tu-dresden.de](mailto:hohmuth@os.inf.tu-dresden.de)

Definition in file [lock.h](#).

## 17.656 lock.h

[Go to the documentation of this file.](#)

```

00001 /*****
00009 */
00010 * (c) 2000-2009 Author(s)
00011 * economic rights: Technische Universität Dresden (Germany)
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014
00015 /*****
00016 #ifndef __L4UTIL_LOCK_H__
00017 #define __L4UTIL_LOCK_H__
00018
00019 #include <l4/sys/thread.h>
00020 #include <l4/sys/compiler.h>
00021 #include <l4/util/atomic.h>
00022
00023 L4_BEGIN_DECLS
00024
00025 typedef l4_uint32_t l4util_simple_lock_t;
00026
00027 L4_INLINE int l4_simple_try_lock(l4util_simple_lock_t *lock);
00028 L4_INLINE void l4_simple_unlock(l4util_simple_lock_t *lock);
00029 L4_INLINE int l4_simple_lock_locked(l4util_simple_lock_t *lock);
00030 L4_INLINE void l4_simple_lock_solid(register l4util_simple_lock_t *p);
00031 L4_INLINE void l4_simple_lock(l4util_simple_lock_t * lock);
00032
00033 L4_INLINE int
00034 l4_simple_try_lock(l4util_simple_lock_t *lock)
00035 {
00036 return l4util_xchg32(lock, 1) == 0;
00037 }
00038
00039 L4_INLINE void
00040 l4_simple_unlock(l4util_simple_lock_t *lock)
00041 {
00042 *lock = 0;
00043 }
00044
00045 L4_INLINE int
00046 l4_simple_lock_locked(l4util_simple_lock_t *lock)
00047 {
00048 return (*lock == 0) ? 0 : 1;
00049 }
00050
00051 L4_INLINE void
00052 l4_simple_lock_solid(register l4util_simple_lock_t *p)
00053 {
00054 while (l4_simple_lock_locked(p) || !l4_simple_try_lock(p))
00055 l4_thread_switch(L4_INVALID_CAP);
00056 }
00057
00058 L4_INLINE void
00059 l4_simple_lock(l4util_simple_lock_t * lock)
00060 {
00061 if (!l4_simple_try_lock(lock))
00062 l4_simple_lock_solid(lock);
00063 }
00064
00065 L4_END_DECLS
00066
00067 #endif

```

## 17.657 l4/util/mb\_info.h File Reference

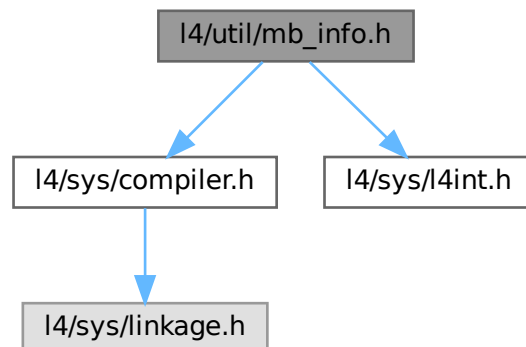
Multiboot info structure as defined by GRUB.

```

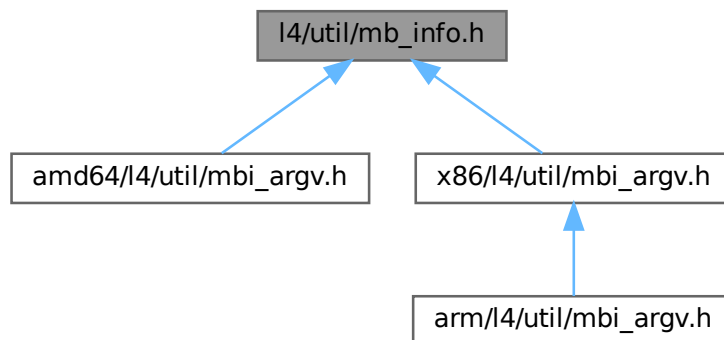
#include <l4/sys/compiler.h>
#include <l4/sys/l4int.h>

```

Include dependency graph for `mb_info.h`:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct `l4util_mb_mod_t`  
The structure type "mod\_list" is used by the `multiboot_info` structure.
- struct `l4util_mb_addr_range_t`  
*INT-15, AX=E820 style "AddressRangeDescriptor" ...with a "size" parameter on the front which is the structure size - 4, pointing to the next one, up until the full buffer length of the memory map has been reached.*
- struct `l4util_mb_drive_t`  
*Drive Info structure.*
- struct `l4util_mb_apm_t`  
*APM BIOS info.*
- struct `l4util_mb_vbe_ctrl_t`



- *VBE controller information.*
- struct [l4util\\_mb\\_vbe\\_mode\\_t](#)
- *VBE mode information.*
- struct [l4util\\_mb\\_info\\_t](#)
- *MultiBoot Info description.*

## Macros

- #define **MB\_ARD\_MEMORY** 1  
*usable memory "Type", all others are reserved.*
- #define **MB\_ART\_MEMORY** 1  
*Address Range Types (ART) from "Advanced Configuration and Power Interface Specification" Rev3.0a (p.*
- #define **MB\_ART\_RESERVED** 2  
*in use or reserved by system*
- #define **MB\_ART\_ACPI** 3  
*ACPI Reclaim Memory (RAM that contains ACPI tables).*
- #define **MB\_ART\_NVS** 4  
*ACPI NVS Memory (must not be used by the OS.*
- #define **MB\_ART\_UNUSABLE** 5  
*memory in which errors have been detected*
- #define [l4util\\_mb\\_for\\_each\\_mmap\\_entry](#)(i, mbi)  
*Iterate over a memory map provided in a Multiboot info.*
- #define **L4UTIL\_MB\_MEMORY** 0x00000001  
*Flags to be set in the 'flags' parameter above.*
- #define **L4UTIL\_MB\_BOOTDEV** 0x00000002  
*is there a boot device set?*
- #define **L4UTIL\_MB\_CMDLINE** 0x00000004  
*is the command-line defined?*
- #define **L4UTIL\_MB\_MODS** 0x00000008  
*are there modules to do something with?*
- #define **L4UTIL\_MB\_AOUT\_SYMS** 0x00000010  
*is there a symbol table loaded?*
- #define **L4UTIL\_MB\_ELF\_SHDR** 0x00000020  
*is there an ELF section header table?*
- #define **L4UTIL\_MB\_MEM\_MAP** 0x00000040  
*is there a full memory map?*
- #define **L4UTIL\_MB\_DRIVE\_INFO** 0x00000080  
*Is there drive info?*
- #define **L4UTIL\_MB\_CONFIG\_TABLE** 0x00000100  
*Is there a config table?*
- #define **L4UTIL\_MB\_BOOT\_LOADER\_NAME** 0x00000200  
*Is there a boot loader name?*
- #define **L4UTIL\_MB\_APM\_TABLE** 0x00000400  
*Is there a APM table?*
- #define **L4UTIL\_MB\_VIDEO\_INFO** 0x00000800  
*Is there video information?*
- #define **L4UTIL\_MB\_VALID** 0x2BADB002UL  
*If we are multiboot-compliant, this value is present in the eax register.*

### 17.657.1 Detailed Description

Multiboot info structure as defined by GRUB.

Definition in file [mb\\_info.h](#).

### 17.657.2 Macro Definition Documentation

#### 17.657.2.1 l4util\_mb\_for\_each\_mmap\_entry

```
#define l4util_mb_for_each_mmap_entry(
 i,
 mbi)
```

**Value:**

```
for (i = l4util_mb_first_mmap_entry(mbi);
 (unsigned long)i < (unsigned long)mbi->mmap_addr + mbi->mmap_length;
 i = l4util_mb_next_mmap_entry(i))
```

Iterate over a memory map provided in a Multiboot info.

#### Parameters

|            |                                                                                                                                               |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <i>i</i>   | Name of a variable of type <a href="#">l4util_mb_addr_range_t</a> * that is consecutively assigned pointers to the entries of the memory map. |
| <i>mbi</i> | Pointer to the <a href="#">l4util_mb_info_t</a> where the memory map can be found.                                                            |

Definition at line 332 of file [mb\\_info.h](#).

#### 17.657.2.2 L4UTIL\_MB\_MEMORY

```
#define L4UTIL_MB_MEMORY 0x00000001
```

Flags to be set in the 'flags' parameter above.

is there basic lower/upper memory information?

Definition at line 344 of file [mb\\_info.h](#).

#### 17.657.2.3 MB\_ART\_MEMORY

```
#define MB_ART_MEMORY 1
```

Address Range Types (ART) from "Advanced Configuration and Power Interface Specification" Rev3.0a (p.

390). Other values are undefined. available, usable RAM

Definition at line 64 of file [mb\\_info.h](#).

## 17.658 mb\_info.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Frank Mehnert <fm3@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011
00012 #ifndef L4UTIL_MB_INFO_H
00013 #define L4UTIL_MB_INFO_H
00014
00015 /*****
00016 * Multiboot (v1)
00017 *****/
00018
00019 #ifndef __ASSEMBLY__
00020
00021 #include <l4/sys/compiler.h>
00022 #include <l4/sys/l4int.h>
00023
00024 /*
00025 * \defgroup l4util_mb_mod Multiboot v1
00026 * \ingroup l4util_api
00027 */
00028
00033 typedef struct
00034 {
00035 l4_uint32_t mod_start;
00036 l4_uint32_t mod_end;
00037 l4_uint32_t cmdline;
00038 l4_uint32_t pad;
00039 } l4util_mb_mod_t;
00040
00041
00048 typedef struct __attribute__((packed))
00049 {
00050 l4_uint32_t struct_size;
00051 l4_uint64_t addr;
00052 l4_uint64_t size;
00053 l4_uint32_t type;
00054 /* unspecified optional padding... */
00055 } l4util_mb_addr_range_t;
00056
00058 #define MB_ARD_MEMORY 1
00059
00064 #define MB_ART_MEMORY 1
00065 #define MB_ART_RESERVED 2
00066 #define MB_ART_ACPI 3
00068 #define MB_ART_NVS 4
00069 #define MB_ART_UNUSABLE 5
00070
00071
00073 typedef struct
00074 {
00075 l4_uint32_t size;
00076 l4_uint8_t drive_number;
00077 l4_uint8_t drive_mode;
00078 l4_uint16_t drive_cylinders;
00079 l4_uint8_t drive_heads;
00080 l4_uint8_t drive_sectors;
00081 l4_uint16_t drive_ports[0];
00082 } l4util_mb_drive_t;
00083
00084 /* Drive Mode. */
00085 #define MB_DI_CHS_MODE 0
00086 #define MB_DI_LBA_MODE 1
00087
00088
00090 typedef struct
00091 {
00092 l4_uint16_t version;
00093 l4_uint16_t cseg;
00094 l4_uint32_t offset;
00095 l4_uint16_t cseg_l6;
00096 l4_uint16_t dseg_l6;
00097 l4_uint16_t flags;
00098 l4_uint16_t cseg_len;
00099 l4_uint16_t cseg_l6_len;
00100 l4_uint16_t dseg_l6_len;
00101 } __attribute__((packed)) l4util_mb_apm_t;
00102 static_assert(sizeof(l4util_mb_apm_t) == 20, "Check l4util_mb_apm_t");
00103

```

```

00104
00106 typedef struct
00107 {
00108 l4_uint8_t signature[4];
00109 l4_uint16_t version;
00110 l4_uint32_t oem_string;
00111 l4_uint32_t capabilities;
00112 l4_uint32_t video_mode;
00113 l4_uint16_t total_memory;
00114 l4_uint16_t oem_software_rev;
00115 l4_uint32_t oem_vendor_name;
00116 l4_uint32_t oem_product_name;
00117 l4_uint32_t oem_product_rev;
00118 l4_uint8_t reserved[222];
00119 l4_uint8_t oem_data[256];
00120 } __attribute__((packed)) l4util_mb_vbe_ctrl_t;
00121 static_assert(sizeof(l4util_mb_vbe_ctrl_t) == 512, "Check l4util_mb_vbe_ctrl_t");
00122
00123
00125 typedef struct
00126 {
00130 l4_uint16_t mode_attributes;
00132 l4_uint8_t win_a_attributes;
00134 l4_uint8_t win_b_attributes;
00136 l4_uint16_t win_granularity;
00138 l4_uint16_t win_size;
00140 l4_uint16_t win_a_segment;
00142 l4_uint16_t win_b_segment;
00144 l4_uint32_t win_func;
00146 l4_uint16_t bytes_per_scanline;
00148
00152 l4_uint16_t x_resolution;
00154 l4_uint16_t y_resolution;
00156 l4_uint8_t x_char_size;
00158 l4_uint8_t y_char_size;
00160 l4_uint8_t number_of_planes;
00162 l4_uint8_t bits_per_pixel;
00164 l4_uint8_t number_of_banks;
00166 l4_uint8_t memory_model;
00168 l4_uint8_t bank_size;
00170 l4_uint8_t number_of_image_pages;
00172 l4_uint8_t reserved0;
00174
00178 l4_uint8_t red_mask_size;
00180 l4_uint8_t red_field_position;
00182 l4_uint8_t green_mask_size;
00184 l4_uint8_t green_field_position;
00186 l4_uint8_t blue_mask_size;
00188 l4_uint8_t blue_field_position;
00190 l4_uint8_t reserved_mask_size;
00192 l4_uint8_t reserved_field_position;
00194 l4_uint8_t direct_color_mode_info;
00196
00200 l4_uint32_t phys_base;
00202 l4_uint32_t reserved1;
00204 l4_uint16_t reversed2;
00206
00210 l4_uint16_t linear_bytes_per_scanline;
00212 l4_uint8_t banked_number_of_image_pages;
00214 l4_uint8_t linear_number_of_image_pages;
00216 l4_uint8_t linear_red_mask_size;
00218 l4_uint8_t linear_red_field_position;
00220 l4_uint8_t linear_green_mask_size;
00222 l4_uint8_t linear_green_field_position;
00224 l4_uint8_t linear_blue_mask_size;
00226 l4_uint8_t linear_blue_field_position;
00228 l4_uint8_t linear_reserved_mask_size;
00230 l4_uint8_t linear_reserved_field_position;
00232 l4_uint32_t max_pixel_clock;
00234 l4_uint8_t reserved3[190];
00236 } __attribute__((packed)) l4util_mb_vbe_mode_t;
00237 static_assert(sizeof(l4util_mb_vbe_mode_t) == 256, "Check l4util_mb_vbe_mode_t");
00238
00239
00246
00247 typedef struct
00248 {
00249 l4_uint32_t flags;
00250 l4_uint32_t mem_lower;
00251 l4_uint32_t mem_upper;
00252 l4_uint32_t boot_device;
00253 l4_uint32_t cmdline;
00254 l4_uint32_t mods_count;
00255 l4_uint32_t mods_addr;
00256
00257 union
00258 {

```

```

00259 struct
00260 {
00262 l4_uint32_t tabsize;
00263 l4_uint32_t strsize;
00264 l4_uint32_t addr;
00265 l4_uint32_t pad;
00266 }
00267 a;
00268
00269 struct
00270 {
00272 l4_uint32_t num;
00273 l4_uint32_t size;
00274 l4_uint32_t addr;
00275 l4_uint32_t shndx;
00276 }
00277 e;
00278 }
00279 syms;
00280
00281 l4_uint32_t mmap_length;
00282 l4_uint32_t mmap_addr;
00283 l4_uint32_t drives_length;
00284 l4_uint32_t drives_addr;
00285 l4_uint32_t config_table;
00286 l4_uint32_t boot_loader_name;
00287 l4_uint32_t apm_table;
00288 l4_uint32_t vbe_ctrl_info;
00289 l4_uint32_t vbe_mode_info;
00290 l4_uint16_t vbe_mode;
00291 l4_uint16_t vbe_interface_seg;
00292 l4_uint16_t vbe_interface_off;
00293 l4_uint16_t vbe_interface_len;
00294 } l4util_mb_info_t;
00295 static_assert(sizeof(l4util_mb_info_t) == 88, "Check l4util_mb_info_t");
00296
00303 static inline l4util_mb_addr_range_t *
00304 l4util_mb_first_mmap_entry(l4util_mb_info_t *mbi)
00305 {
00306 return (l4util_mb_addr_range_t *) (l4_addr_t) mbi->mmap_addr;
00307 }
00308
00318 static inline l4util_mb_addr_range_t *
00319 l4util_mb_next_mmap_entry(l4util_mb_addr_range_t *e)
00320 {
00321 return (l4util_mb_addr_range_t *) ((l4_addr_t) e + e->struct_size
00322 + sizeof(e->struct_size));
00323 }
00324
00332 #define l4util_mb_for_each_mmap_entry(i, mbi) \
00333 for (i = l4util_mb_first_mmap_entry(mbi); \
00334 (unsigned long)i < (unsigned long)mbi->mmap_addr + mbi->mmap_length; \
00335 i = l4util_mb_next_mmap_entry(i)) \
00336
00337 #endif /* ! __ASSEMBLY__ */
00338
00342
00344 #define L4UTIL_MB_MEMORY 0x00000001
00345
00347 #define L4UTIL_MB_BOOTDEV 0x00000002
00348
00350 #define L4UTIL_MB_CMDLINE 0x00000004
00351
00353 #define L4UTIL_MB_MODS 0x00000008
00354
00355 /* These next two are mutually exclusive */
00357 #define L4UTIL_MB_AOUT_SYMS 0x00000010
00358
00360 #define L4UTIL_MB_ELF_SHDR 0x00000020
00361
00363 #define L4UTIL_MB_MEM_MAP 0x00000040
00364
00366 #define L4UTIL_MB_DRIVE_INFO 0x00000080
00367
00369 #define L4UTIL_MB_CONFIG_TABLE 0x00000100
00370
00372 #define L4UTIL_MB_BOOT_LOADER_NAME 0x00000200
00373
00375 #define L4UTIL_MB_APM_TABLE 0x00000400
00376
00378 #define L4UTIL_MB_VIDEO_INFO 0x00000800
00379
00380
00382 #define L4UTIL_MB_VALID 0x2BADB002UL
00383 #define L4UTIL_MB_VALID_ASM 0x2BADB002
00384
00385

```

```

00386 /*****
00387 * Multiboot2
00388 *****/
00389
00390 #ifndef __ASSEMBLY__
00391
00392 typedef struct
00393 {
00394 l4_uint32_t total_size;
00395 l4_uint32_t reserved;
00396 } __attribute__((packed)) l4util_mb2_info_t;
00397
00398 typedef struct
00399 {
00400 char string[0];
00401 } __attribute__((packed)) l4util_mb2_cmdline_tag_t;
00402
00403 typedef struct
00404 {
00405 l4_uint32_t mod_start;
00406 l4_uint32_t mod_end;
00407 char string[];
00408 } __attribute__((packed)) l4util_mb2_module_tag_t;
00409
00410 typedef struct
00411 {
00412 l4_uint64_t base_addr;
00413 l4_uint64_t length;
00414 l4_uint32_t type;
00415 l4_uint32_t reserved;
00416 } __attribute__((packed)) l4util_mb2_memmap_entry_t;
00417
00418 typedef struct
00419 {
00420 l4_uint32_t entry_size;
00421 l4_uint32_t entry_version;
00422 l4util_mb2_memmap_entry_t entries[];
00423 } __attribute__((packed)) l4util_mb2_memmap_tag_t;
00424
00425 typedef struct
00426 {
00427 char data[0];
00428 } __attribute__((packed)) l4util_mb2_rsdp_tag_t;
00429
00430
00431 struct color_info_rgb_t
00432 {
00433 l4_uint8_t framebuffer_red_field_position;
00434 l4_uint8_t framebuffer_red_mask_size;
00435 l4_uint8_t framebuffer_green_field_position;
00436 l4_uint8_t framebuffer_green_mask_size;
00437 l4_uint8_t framebuffer_blue_field_position;
00438 l4_uint8_t framebuffer_blue_mask_size;
00439 } __attribute__((packed));
00440
00441 typedef struct
00442 {
00443 l4_uint64_t framebuffer_addr;
00444 l4_uint32_t framebuffer_pitch;
00445 l4_uint32_t framebuffer_width;
00446 l4_uint32_t framebuffer_height;
00447 l4_uint8_t framebuffer_bpp;
00448 l4_uint8_t framebuffer_type;
00449 l4_uint8_t reserved;
00450
00451 // color_info;
00452 union
00453 {
00454 struct color_info_rgb_t color_info_rgb;
00455 };
00456 } __attribute__((packed)) l4util_mb2_framebuffer_tag_t;
00457
00458 typedef struct
00459 {
00460 l4_uint32_t type;
00461 l4_uint32_t size;
00462
00463 union
00464 {
00465 l4util_mb2_cmdline_tag_t cmdline;
00466 l4util_mb2_module_tag_t module;
00467 l4util_mb2_memmap_tag_t memmap;
00468 l4util_mb2_framebuffer_tag_t fb;
00469 l4util_mb2_rsdp_tag_t rsdp;
00470 };
00471 } __attribute__((packed)) l4util_mb2_tag_t;
00472

```

```

00473 #endif /* ! __ASSEMBLY__ */
00474
00475
00476 #define L4UTIL_MB2_MAGIC 0xE85250D6
00477 #define L4UTIL_MB2_ARCH_I386 0x0
00478
00479 #define L4UTIL_MB2_TERMINATOR_HEADER_TAG 0
00480 #define L4UTIL_MB2_INFO_REQUEST_HEADER_TAG 1
00481 #define L4UTIL_MB2_ENTRY_ADDRESS_HEADER_TAG 3
00482 #define L4UTIL_MB2_FRAMEBUFFER_HEADER_TAG 5
00483 #define L4UTIL_MB2_RELOCATABLE_HEADER_TAG 10
00484
00485 #define L4UTIL_MB2_TAG_FLAG_REQUIRED 0
00486
00487 #define L4UTIL_MB2_TAG_ALIGN_SHIFT 3
00488 #define L4UTIL_MB2_TAG_ALIGN 8
00489
00490 #define L4UTIL_MB2_TERMINATOR_INFO_TAG 0
00491 #define L4UTIL_MB2_BOOT_CMDLINE_INFO_TAG 1
00492 #define L4UTIL_MB2_MODULE_INFO_TAG 3
00493 #define L4UTIL_MB2_MEMORY_MAP_INFO_TAG 6
00494 #define L4UTIL_MB2_FRAMEBUFFER_INFO_TAG 8
00495 #define L4UTIL_MB2_RSDP_OLD_INFO_TAG 14
00496 #define L4UTIL_MB2_RSDP_NEW_INFO_TAG 15
00497 #define L4UTIL_MB2_IMAGE_LOAD_BASE_PHYS_INFO_TAG 21
00498
00499 #define L4UTIL_MB2_RELO_PREFERRED_NONE 0
00500 #define L4UTIL_MB2_RELO_PREFERRED_MIN 1
00501 #define L4UTIL_MB2_RELO_PREFERRED_MAX 2
00502
00503 #endif

```

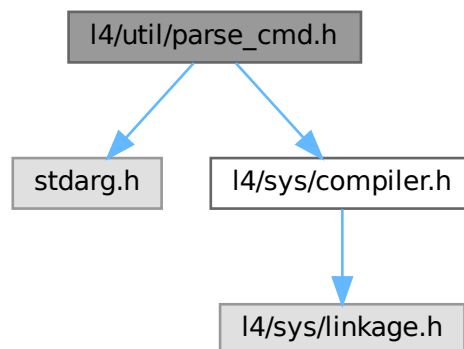
## 17.659 l4/util/parse\_cmd.h File Reference

comfortable command-line parsing

```
#include <stdarg.h>
```

```
#include <l4/sys/compiler.h>
```

Include dependency graph for parse\_cmd.h:



### Typedefs

- typedef void(\* **parse\_cmd\_fn\_t**) (int)  
Function type for `PARSE_CMD_FN`.
- typedef void(\* **parse\_cmd\_fn\_arg\_t**) (int, const char \*, int)  
Function type for `PARSE_CMD_FN_ARG`.

## Enumerations

- enum [parse\\_cmd\\_type](#)

*Types for parsing.*

## Functions

- [L4\\_BEGIN\\_DECLS](#) int [parse\\_cmdline](#) (int \*argc, const char \*\*\*argv, int arg0,...)

*Parse the command-line for specified arguments and store the values into variables.*

## 17.659.1 Detailed Description

comfortable command-line parsing

### Date

2002

### Author

Jork Loeser [jork.loeser@inf.tu-dresden.de](mailto:jork.loeser@inf.tu-dresden.de)

Definition in file [parse\\_cmd.h](#).

## 17.660 [parse\\_cmd.h](#)

[Go to the documentation of this file.](#)

```

00001
00008
00009 /*
00010 * (c) 2003-2009 Author(s)
00011 * economic rights: Technische Universität Dresden (Germany)
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014
00015 #ifndef __PARSE_CMD_H
00016 #define __PARSE_CMD_H
00017
00018 #include <stdarg.h>
00019 #include <l4/sys/compiler.h>
00020
00026
00030 enum parse_cmd_type {
00031 PARSE_CMD_INT,
00032 PARSE_CMD_SWITCH,
00033 PARSE_CMD_STRING,
00034 PARSE_CMD_FN,
00035 PARSE_CMD_FN_ARG,
00036 PARSE_CMD_INC,
00037 PARSE_CMD_DEC,
00038 };
00039
00043 typedef L4_CV void (*parse_cmd_fn_t)(int);
00044
00048 typedef L4_CV void (*parse_cmd_fn_arg_t)(int, const char*, int);
00049
00050 L4_BEGIN_DECLS
00051
00138 L4_CV int parse_cmdline(int *argc, const char***argv, int arg0, ...);
00139 L4_CV int parse_cmdlinev(int *argc, const char***argv, int arg0, va_list va);
00140 L4_CV int parse_cmdline_extra(const char*argv0, const char*line, char delim,
00141 int arg0,...);
00142
00143 L4_END_DECLS
00145
00146 #endif
00147

```



## 17.661 printf\_helpers.h

```

00001 /*
00002 * Copyright (C) 2025 Kernkonzept GmbH.
00003 * Author(s): Frank Mehnert <frank.mehnert@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #pragma once
00009
00010 #include <stddef.h>
00011 #include <stdio.h>
00012 #include <l4/sys/compiler.h>
00013
00030 L4_INLINE int l4util_human_readable_size(char *outstr, size_t outsize,
00031 unsigned long long bytes)
00032 {
00033 static char const *const unitstr = "BKMGT";
00034
00035 int idx = sizeof(unitstr) - 2;
00036 int order;
00037
00038 for (order = idx * 10; order > 10; order -= 10, --idx)
00039 if (bytes > (1ULL « order))
00040 break;
00041
00042 unsigned long long value = bytes » order;
00043 unsigned long long fract = (bytes - (value « order))
00044 / ((1ULL « order) / 10 + 1);
00045
00046 return snprintf(outstr, outsize, "%llu.%1llu %ciB",
00047 value, fract, unitstr[idx]);
00048 }

```

## 17.662 l4/util/rand.h File Reference

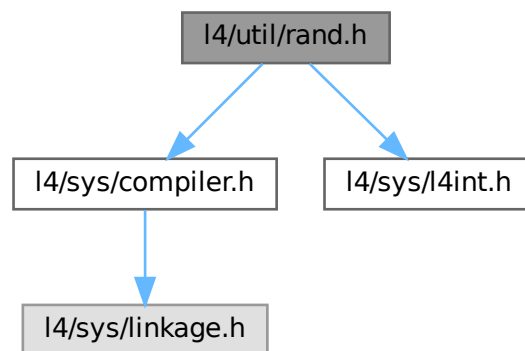
Simple Pseudo-Random Number Generator.

```

#include <l4/sys/compiler.h>
#include <l4/sys/l4int.h>

```

Include dependency graph for rand.h:



### Functions

- [l4\\_uint32\\_t l4util\\_rand](#) (void)

*Deliver next random number.*

- void `l4util_srand` (`l4_uint32_t` seed)

*Initialize random number generator.*

## 17.662.1 Detailed Description

Simple Pseudo-Random Number Generator.

### Date

1998

### Author

Lars Reuther `reuther@os.inf.tu-dresden.de`

Definition in file [rand.h](#).

## 17.663 rand.h

[Go to the documentation of this file.](#)

```

00001
00008 /*
00009 * (c) 2008-2009 Author(s)
00010 * economic rights: Technische Universität Dresden (Germany)
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013
00014 #ifndef __L4UTIL_RAND_H
00015 #define __L4UTIL_RAND_H
00016
00017 #define L4_RAND_MAX 65535
00018
00019 #include <l4/sys/compiler.h>
00020 #include <l4/sys/l4int.h>
00021
00022 L4_BEGIN_DECLS
00023
00028
00035 L4_CV l4_uint32_t
00036 l4util_rand(void);
00037
00044 L4_CV void
00045 l4util_srand (l4_uint32_t seed);
00046
00047 L4_END_DECLS
00048
00049 #endif /* __L4UTIL_RAND_H */

```

## 17.664 l4/util/splitlog2.h File Reference

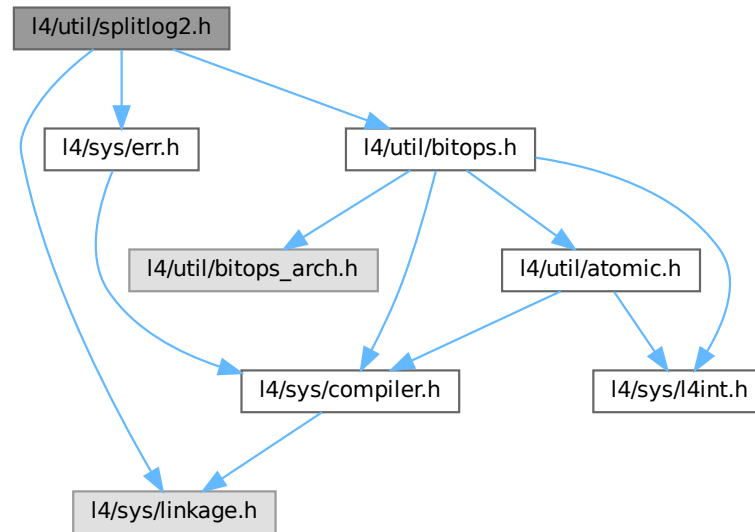
Split a range in log2 aligned and size-aligned chunks.

```

#include <l4/sys/linkage.h>
#include <l4/sys/err.h>

```

#include <l4/util/bitops.h>  
 Include dependency graph for splitlog2.h:



## Functions

- `L4_BEGIN_DECLS` long `l4util_splitlog2_hdl` (`l4_addr_t` start, `l4_addr_t` end, long(\*handler)(`l4_addr_t` s, `l4_addr_t` e, int log2size))  
*Split a range into log2 base and size aligned chunks.*
- `l4_addr_t` `l4util_splitlog2_size` (`l4_addr_t` start, `l4_addr_t` end)  
*Return log2 base and size aligned length of a range.*

### 17.664.1 Detailed Description

Split a range in log2 aligned and size-aligned chunks.

Definition in file [splitlog2.h](#).

## 17.665 splitlog2.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00007 * economic rights: Technische Universität Dresden (Germany)
00008 * License: see LICENSE.spdx (in this directory or the directories above)
00009 */
00010 #ifndef __L4UTIL__INCLUDE__SPLITLOG2_H__
00011 #define __L4UTIL__INCLUDE__SPLITLOG2_H__
00012
00013 #include <l4/sys/linkage.h>
00014 #include <l4/sys/err.h>
00015 #include <l4/util/bitops.h>

```

```

00016
00017 L4_BEGIN_DECLS
00018
00031 L4_INLINE long
00032 l4util_splitlog2_hdl(l4_addr_t start, l4_addr_t end,
00033 long (*handler)(l4_addr_t s, l4_addr_t e, int log2size));
00034
00043 L4_INLINE l4_addr_t
00044 l4util_splitlog2_size(l4_addr_t start, l4_addr_t end);
00045
00046 L4_END_DECLS
00047
00048 /* Implementation */
00049
00050 L4_INLINE long
00051 l4util_splitlog2_hdl(l4_addr_t start, l4_addr_t end,
00052 long (*handler)(l4_addr_t s, l4_addr_t e, int log2size))
00053 {
00054 if (end < start)
00055 return -L4_EINVAL;
00056 while (start <= end)
00057 {
00058 long retval;
00059 int len2 = l4util_splitlog2_size(start, end);
00060 l4_addr_t len = 1UL << len2;
00061 if ((retval = handler(start, start + len - 1, len2)))
00062 return retval;
00063 start += len;
00064 }
00065 return 0;
00066 }
00067
00068 L4_INLINE l4_addr_t
00069 l4util_splitlog2_size(l4_addr_t start, l4_addr_t end)
00070 {
00071 int start_bits = l4util_bsf(start);
00072 int len_bits = l4util_bsr(end - start + 1);
00073 if (start_bits != -1 && len_bits > start_bits)
00074 len_bits = start_bits;
00075 return len_bits;
00076 }
00077
00078 }
00079
00080 #endif /* ! __L4UTIL__INCLUDE__SPLITLOG2_H__ */

```

## 17.666 arm/l4/sys/thread.h File Reference

ARM-specific thread related definitions.

### Enumerations

- enum [L4\\_thread\\_ex\\_regs\\_flags\\_arm](#) { [L4\\_THREAD\\_EX\\_REGS\\_ARM\\_SET\\_EL\\_MASK](#) = 0x3 << 24 , [L4\\_THREAD\\_EX\\_REGS\\_ARM\\_SET\\_EL\\_KEEP](#) = 0x0 << 24 , [L4\\_THREAD\\_EX\\_REGS\\_ARM\\_SET\\_EL\\_EL0](#) = 0x1 << 24 , [L4\\_THREAD\\_EX\\_REGS\\_ARM\\_SET\\_EL\\_EL1](#) = 0x2 << 24 }

*Arm specific [L4::Thread::ex\\_regs\(\)](#) flags.*

### Functions

- [l4\\_msgtag\\_t l4\\_thread\\_arm\\_set\\_tpidruru](#) ([l4\\_cap\\_idx\\_t](#) thread, [l4\\_addr\\_t](#) tpidruru) [L4\\_NOTHROW](#)

*Set the [TPIDRURO](#) thread specific register.*

### 17.666.1 Detailed Description

ARM-specific thread related definitions.

Definition in file [thread.h](#).

## 17.667 thread.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2013 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * economic rights: Technische Universität Dresden (Germany)
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #pragma once
00012
00013 #include_next <l4/sys/thread.h>
00014
00015 // Use the full documentation from ARM64. Otherwise \parameters and \return
00016 // would occur twice in the Doxygen documentation of this function.
00020 L4_INLINE l4_msgtag_t
00021 l4_thread_arm_set_tpidruro(l4_cap_idx_t thread, l4_addr_t tpidruro) L4_NOTHROW;
00022
00027 L4_INLINE l4_msgtag_t
00028 l4_thread_arm_set_tpidruro_u(l4_cap_idx_t thread, l4_addr_t tpidruro,
00029 l4_utcb_t *utcb) L4_NOTHROW;
00030
00039 enum l4_thread_ex_regs_flags_arm
00040 {
00042 L4_THREAD_EX_REGS_ARM_SET_EL_MASK = 0x3 << 24,
00044 L4_THREAD_EX_REGS_ARM_SET_EL_KEEP = 0x0 << 24,
00046 L4_THREAD_EX_REGS_ARM_SET_EL_EL0 = 0x1 << 24,
00048 L4_THREAD_EX_REGS_ARM_SET_EL_EL1 = 0x2 << 24,
00049 };
00050
00051 /* IMPLEMENTATION ----- */
00052
00053 L4_INLINE l4_msgtag_t
00054 l4_thread_arm_set_tpidruro_u(l4_cap_idx_t thread, l4_addr_t tpidruro,
00055 l4_utcb_t *utcb) L4_NOTHROW
00056 {
00057 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00058 v->mr[0] = L4_THREAD_ARM_TPIDRURO_OP;
00059 v->mr[1] = tpidruro;
00060 return l4_ipc_call(thread, utcb, l4_msgtag(L4_PROTO_THREAD, 2, 0, 0),
00061 L4_IPC_NEVER);
00062 }
00063
00064 L4_INLINE l4_msgtag_t
00065 l4_thread_arm_set_tpidruro(l4_cap_idx_t thread, l4_addr_t tpidruro) L4_NOTHROW
00066 {
00067 return l4_thread_arm_set_tpidruro_u(thread, tpidruro, l4_utcb());
00068 }

```

## 17.668 arm64/l4/sys/thread.h File Reference

ARM64-specific thread related definitions.

### Enumerations

- enum [l4\\_thread\\_ex\\_regs\\_flags\\_arm64](#) { [L4\\_THREAD\\_EX\\_REGS\\_ARM64\\_SET\\_EL\\_MASK](#) = 0x3 << 24 , [L4\\_THREAD\\_EX\\_REGS\\_ARM64\\_SET\\_EL\\_KEEP](#) = 0x0 << 24 , [L4\\_THREAD\\_EX\\_REGS\\_ARM64\\_SET\\_EL\\_EL0](#) = 0x1 << 24 , [L4\\_THREAD\\_EX\\_REGS\\_ARM64\\_SET\\_EL\\_EL1](#) = 0x2 << 24 }

*Arm64 specific [L4::Thread::ex\\_regs\(\)](#) flags.*

### Functions

- [l4\\_msgtag\\_t l4\\_thread\\_arm\\_set\\_tpidruro](#) ([l4\\_cap\\_idx\\_t](#) thread, [l4\\_addr\\_t](#) tpidruro) [L4\\_NOTHROW](#)

*Set the [TPIDRURO](#) thread specific register.*

## 17.668.1 Detailed Description

ARM64-specific thread related definitions.

Definition in file [thread.h](#).

## 17.669 thread.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2013 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * economic rights: Technische Universität Dresden (Germany)
00008 *
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #pragma once
00012
00013 #include_next <l4/sys/thread.h>
00014
00027 L4_INLINE l4_msgtag_t
00028 l4_thread_arm_set_tpidruro(l4_cap_idx_t thread, l4_addr_t tpidruro) L4_NOTHROW;
00029
00034 L4_INLINE l4_msgtag_t
00035 l4_thread_arm_set_tpidruro_u(l4_cap_idx_t thread, l4_addr_t tpidruro,
00036 l4_utcb_t *utcb) L4_NOTHROW;
00037
00046 enum L4_thread_ex_regs_flags_arm64
00047 {
00049 L4_THREAD_EX_REGS_ARM64_SET_EL_MASK = 0x3 « 24,
00051 L4_THREAD_EX_REGS_ARM64_SET_EL_KEEP = 0x0 « 24,
00053 L4_THREAD_EX_REGS_ARM64_SET_EL_EL0 = 0x1 « 24,
00055 L4_THREAD_EX_REGS_ARM64_SET_EL_EL1 = 0x2 « 24,
00056 };
00057
00058 /* IMPLEMENTATION ----- */
00059
00060 L4_INLINE l4_msgtag_t
00061 l4_thread_arm_set_tpidruro_u(l4_cap_idx_t thread, l4_addr_t tpidruro,
00062 l4_utcb_t *utcb) L4_NOTHROW
00063 {
00064 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00065 v->mr[0] = L4_THREAD_ARM_TPIDRURO_OP;
00066 v->mr[1] = tpidruro;
00067 return l4_ipc_call(thread, utcb, l4_msgtag(L4_PROTO_THREAD, 2, 0, 0),
00068 L4_IPC_NEVER);
00069 }
00070
00071 L4_INLINE l4_msgtag_t
00072 l4_thread_arm_set_tpidruro(l4_cap_idx_t thread, l4_addr_t tpidruro) L4_NOTHROW
00073 {
00074 return l4_thread_arm_set_tpidruro_u(thread, tpidruro, l4_utcb());
00075 }

```

## 17.670 l4/sys/thread.h File Reference

Common thread related definitions.

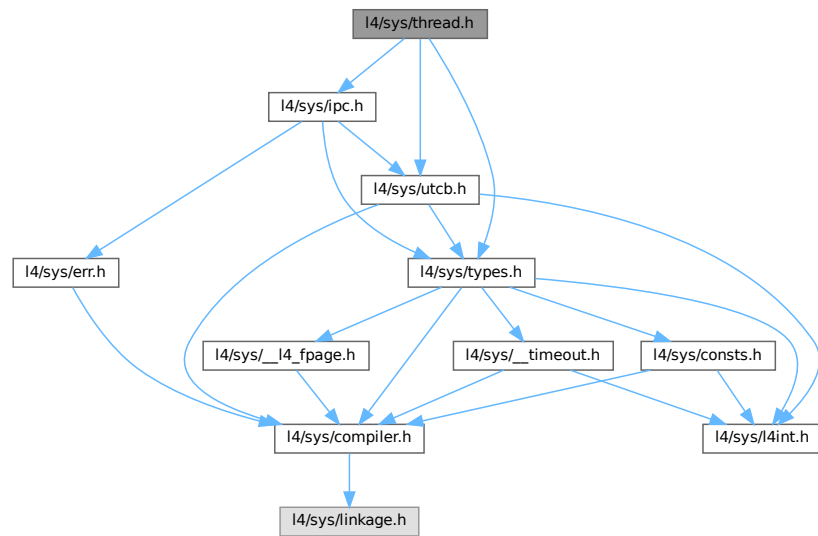
```

#include <l4/sys/types.h>
#include <l4/sys/utcb.h>

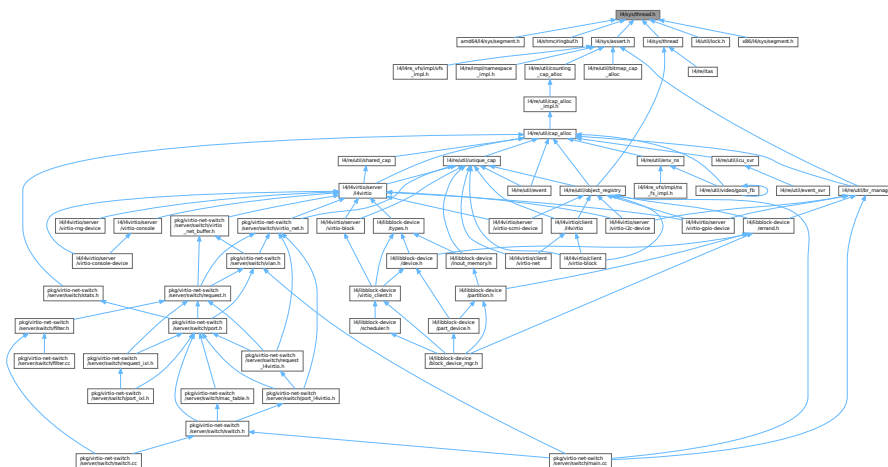
```

```
#include <l4/sys/ipc.h>
```

Include dependency graph for thread.h:



This graph shows which files directly or indirectly include this file:



## Enumerations

- enum [L4\\_thread\\_ops](#) {
  - [L4\\_THREAD\\_CONTROL\\_OP](#) = 0UL , [L4\\_THREAD\\_EX\\_REGS\\_OP](#) = 1UL , [L4\\_THREAD\\_SWITCH\\_OP](#) = 2UL , [L4\\_THREAD\\_STATS\\_OP](#) = 3UL ,
  - [L4\\_THREAD\\_VCPU\\_RESUME\\_OP](#) = 4UL , [L4\\_THREAD\\_REGISTER\\_DELETE\\_IRQ\\_OP](#) = 5UL ,
  - [L4\\_THREAD\\_MODIFY\\_SENDER\\_OP](#) = 6UL , [L4\\_THREAD\\_VCPU\\_CONTROL\\_OP](#) = 7UL ,
  - [L4\\_THREAD\\_VCPU\\_CONTROL\\_EXT\\_OP](#) = [L4\\_THREAD\\_VCPU\\_CONTROL\\_OP](#) | 0x10000 , [L4\\_THREAD\\_REGISTER\\_DO](#) = 8UL , [L4\\_THREAD\\_X86\\_GDT\\_OP](#) = 0x10UL , [L4\\_THREAD\\_ARM\\_TPIDRURO\\_OP](#) = 0x10UL ,
  - [L4\\_THREAD\\_AMD64\\_SET\\_SEGMENT\\_BASE\\_OP](#) = 0x12UL , [L4\\_THREAD\\_AMD64\\_GET\\_SEGMENT\\_INFO\\_OP](#) = 0x13UL , [L4\\_THREAD\\_OPCODE\\_MASK](#) = 0xffff }

*Operations on thread objects.*

- enum `L4_thread_control_flags` { `L4_THREAD_CONTROL_SET_PAGER` = 0x0010000 , `L4_THREAD_CONTROL_BIND_TASK` = 0x0200000 , `L4_THREAD_CONTROL_ALIEN` = 0x0400000 , `L4_THREAD_CONTROL_SET_EXC_HANDLER` = 0x1000000 }

*Flags for the thread control operation.*

- enum `L4_thread_control_mr_indices` {  
`L4_THREAD_CONTROL_MR_IDX_FLAGS` = 0 , `L4_THREAD_CONTROL_MR_IDX_PAGER` = 1 ,  
`L4_THREAD_CONTROL_MR_IDX_EXC_HANDLER` = 2 , `L4_THREAD_CONTROL_MR_IDX_FLAG_VALS`  
= 4 ,  
`L4_THREAD_CONTROL_MR_IDX_BIND_UTCB` = 5 , `L4_THREAD_CONTROL_MR_IDX_BIND_TASK` = 6  
}

*Indices for the values in the message register for thread control.*

- enum `L4_thread_ex_regs_flags` { `L4_THREAD_EX_REGS_CANCEL` = 0x10000UL , `L4_THREAD_EX_REGS_TRIGGER_EXC` = 0x20000UL , `L4_THREAD_EX_REGS_ARCH_MASK` = 0xff000000UL }

*Flags for the thread ex-regs operation.*

## Functions

- `l4_msgtag_t l4_thread_ex_regs` (`l4_cap_idx_t` thread, `l4_addr_t` ip, `l4_addr_t` sp, `l4_umword_t` flags) `L4_NOTHROW`

*Exchange basic thread registers.*

- `l4_msgtag_t l4_thread_ex_regs_u` (`l4_cap_idx_t` thread, `l4_addr_t` ip, `l4_addr_t` sp, `l4_umword_t` flags, `l4_utcb_t` \*utcb) `L4_NOTHROW`

*Exchange basic thread registers.*

- `l4_msgtag_t l4_thread_ex_regs_ret` (`l4_cap_idx_t` thread, `l4_addr_t` \*ip, `l4_addr_t` \*sp, `l4_umword_t` \*flags) `L4_NOTHROW`

*Exchange basic thread registers and return previous values.*

- `l4_msgtag_t l4_thread_ex_regs_ret_u` (`l4_cap_idx_t` thread, `l4_addr_t` \*ip, `l4_addr_t` \*sp, `l4_umword_t` \*flags, `l4_utcb_t` \*utcb) `L4_NOTHROW`

*Exchange basic thread registers and return previous values.*

- void `l4_thread_control_start` (void) `L4_NOTHROW`

*Start a thread control API sequence.*

- void `l4_thread_control_pager` (`l4_cap_idx_t` pager) `L4_NOTHROW`

*Set the pager.*

- void `l4_thread_control_exc_handler` (`l4_cap_idx_t` exc\_handler) `L4_NOTHROW`

*Set the exception handler.*

- void `l4_thread_control_bind` (`l4_utcb_t` \*thread\_utcb, `l4_cap_idx_t` task) `L4_NOTHROW`

*Bind the thread to a task.*

- void `l4_thread_control_alien` (int on) `L4_NOTHROW`

*Enable alien mode.*

- `l4_msgtag_t l4_thread_control_commit` (`l4_cap_idx_t` thread) `L4_NOTHROW`

*Commit the thread control parameters.*

- `l4_msgtag_t l4_thread_yield` (void) `L4_NOTHROW`

*Yield current time slice.*

- `l4_msgtag_t l4_thread_switch` (`l4_cap_idx_t` to\_thread) `L4_NOTHROW`

*Switch to another thread (and donate the remaining time slice).*

- `l4_msgtag_t l4_thread_stats_time` (`l4_cap_idx_t` thread, `l4_kernel_clock_t` \*us) `L4_NOTHROW`

*Get consumed time of thread in  $\mu$ s.*

- `l4_msgtag_t l4_thread_vcpu_resume_start` (void) `L4_NOTHROW`

*vCPU return from event handler.*

- `l4_msgtag_t l4_thread_vcpu_resume_commit` (`l4_cap_idx_t` thread, `l4_msgtag_t` tag) `L4_NOTHROW`

*Commit vCPU resume.*



- [l4\\_msgtag\\_t l4\\_thread\\_vcpu\\_control \(l4\\_cap\\_idx\\_t thread, l4\\_addr\\_t vcpu\\_state\) L4\\_NOTHROW](#)  
*Enable the vCPU feature for the thread.*
- [l4\\_msgtag\\_t l4\\_thread\\_vcpu\\_control\\_u \(l4\\_cap\\_idx\\_t thread, l4\\_addr\\_t vcpu\\_state, l4\\_utcb\\_t \\*utcb\) L4\\_NOTHROW](#)  
*Enable the vCPU feature for the thread.*
- [l4\\_msgtag\\_t l4\\_thread\\_vcpu\\_control\\_ext \(l4\\_cap\\_idx\\_t thread, l4\\_addr\\_t ext\\_vcpu\\_state\) L4\\_NOTHROW](#)  
*Enable the extended vCPU feature for the thread.*
- [l4\\_msgtag\\_t l4\\_thread\\_vcpu\\_control\\_ext\\_u \(l4\\_cap\\_idx\\_t thread, l4\\_addr\\_t ext\\_vcpu\\_state, l4\\_utcb\\_t \\*utcb\) L4\\_NOTHROW](#)  
*Enable the extended vCPU feature for the thread.*
- [l4\\_msgtag\\_t l4\\_thread\\_register\\_del\\_irq \(l4\\_cap\\_idx\\_t thread, l4\\_cap\\_idx\\_t irq\) L4\\_NOTHROW](#)  
*Register an IRQ that will trigger upon deletion events.*
- [l4\\_msgtag\\_t l4\\_thread\\_modify\\_sender\\_start \(void\) L4\\_NOTHROW](#)  
*Start a thread sender modification sequence.*
- [int l4\\_thread\\_modify\\_sender\\_add \(l4\\_umword\\_t match\\_mask, l4\\_umword\\_t match, l4\\_umword\\_t del\\_bits, l4\\_umword\\_t add\\_bits, l4\\_msgtag\\_t \\*tag\) L4\\_NOTHROW](#)  
*Add a modification pattern to a sender modification sequence.*
- [l4\\_msgtag\\_t l4\\_thread\\_modify\\_sender\\_commit \(l4\\_cap\\_idx\\_t thread, l4\\_msgtag\\_t tag\) L4\\_NOTHROW](#)  
*Apply (commit) a sender modification sequence.*
- [l4\\_msgtag\\_t l4\\_thread\\_register\\_doorbell\\_irq \(l4\\_cap\\_idx\\_t thread, l4\\_cap\\_idx\\_t irq\) L4\\_NOTHROW](#)  
*Register an IRQ that will trigger when a forwarded virtual interrupt is pending.*

## 17.670.1 Detailed Description

Common thread related definitions.

Definition in file [thread.h](#).

## 17.671 thread.h

[Go to the documentation of this file.](#)

```

00001
00005 /*
00006 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00007 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00008 * Björn Döbel <doebel@os.inf.tu-dresden.de>,
00009 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00010 * economic rights: Technische Universität Dresden (Germany)
00011 *
00012 * License: see LICENSE.spdx (in this directory or the directories above)
00013 */
00014 #pragma once
00015
00016 #include <l4/sys/types.h>
00017 #include <l4/sys/utcb.h>
00018 #include <l4/sys/ipc.h>
00019
00054
00055
00083 L4_INLINE l4_msgtag_t
00084 l4_thread_ex_regs(l4_cap_idx_t thread, l4_addr_t ip, l4_addr_t sp,
00085 l4_umword_t flags) L4_NOTHROW;
00086
00093 L4_INLINE l4_msgtag_t
00094 l4_thread_ex_regs_u(l4_cap_idx_t thread, l4_addr_t ip, l4_addr_t sp,
00095 l4_umword_t flags, l4_utcb_t *utcb) L4_NOTHROW;
00096
00129 L4_INLINE l4_msgtag_t
00130 l4_thread_ex_regs_ret(l4_cap_idx_t thread, l4_addr_t *ip, l4_addr_t *sp,
00131 l4_umword_t *flags) L4_NOTHROW;
00132

```

```

00139 L4_INLINE l4_msgtag_t
00140 l4_thread_ex_regs_ret_u(l4_cap_idx_t thread, l4_addr_t *ip, l4_addr_t *sp,
00141 l4_umword_t *flags, l4_utcb_t *utcb) L4_NOTHROW;
00142
00143
00144
00170
00190 L4_INLINE void
00191 l4_thread_control_start(void) L4_NOTHROW;
00192
00197 L4_INLINE void
00198 l4_thread_control_start_u(l4_utcb_t *utcb) L4_NOTHROW;
00199
00209 L4_INLINE void
00210 l4_thread_control_pager(l4_cap_idx_t pager) L4_NOTHROW;
00211
00216 L4_INLINE void
00217 l4_thread_control_pager_u(l4_cap_idx_t pager, l4_utcb_t *utcb) L4_NOTHROW;
00218
00228 L4_INLINE void
00229 l4_thread_control_exc_handler(l4_cap_idx_t exc_handler) L4_NOTHROW;
00230
00235 L4_INLINE void
00236 l4_thread_control_exc_handler_u(l4_cap_idx_t exc_handler,
00237 l4_utcb_t *utcb) L4_NOTHROW;
00238
00266 L4_INLINE void
00267 l4_thread_control_bind(l4_utcb_t *thread_utcb,
00268 l4_cap_idx_t task) L4_NOTHROW;
00269
00274 L4_INLINE void
00275 l4_thread_control_bind_u(l4_utcb_t *thread_utcb,
00276 l4_cap_idx_t task, l4_utcb_t *utcb) L4_NOTHROW;
00277
00301 L4_INLINE void
00302 l4_thread_control_alien(int on) L4_NOTHROW;
00303
00308 L4_INLINE void
00309 l4_thread_control_alien_u(l4_utcb_t *utcb, int on) L4_NOTHROW;
00310
00311
00312
00313
00330 L4_INLINE l4_msgtag_t
00331 l4_thread_control_commit(l4_cap_idx_t thread) L4_NOTHROW;
00332
00337 L4_INLINE l4_msgtag_t
00338 l4_thread_control_commit_u(l4_cap_idx_t thread, l4_utcb_t *utcb) L4_NOTHROW;
00339
00346 L4_INLINE l4_msgtag_t
00347 l4_thread_yield(void) L4_NOTHROW;
00348
00357 L4_INLINE l4_msgtag_t
00358 l4_thread_switch(l4_cap_idx_t to_thread) L4_NOTHROW;
00359
00364 L4_INLINE l4_msgtag_t
00365 l4_thread_switch_u(l4_cap_idx_t to_thread, l4_utcb_t *utcb) L4_NOTHROW;
00366
00367
00368
00378 L4_INLINE l4_msgtag_t
00379 l4_thread_stats_time(l4_cap_idx_t thread, l4_kernel_clock_t *us) L4_NOTHROW;
00380
00385 L4_INLINE l4_msgtag_t
00386 l4_thread_stats_time_u(l4_cap_idx_t thread, l4_kernel_clock_t *us,
00387 l4_utcb_t *utcb) L4_NOTHROW;
00388
00389
00400 L4_INLINE l4_msgtag_t
00401 l4_thread_vcpu_resume_start(void) L4_NOTHROW;
00402
00407 L4_INLINE l4_msgtag_t
00408 l4_thread_vcpu_resume_start_u(l4_utcb_t *utcb) L4_NOTHROW;
00409
00457 L4_INLINE l4_msgtag_t
00458 l4_thread_vcpu_resume_commit(l4_cap_idx_t thread,
00459 l4_msgtag_t tag) L4_NOTHROW;
00460
00465 L4_INLINE l4_msgtag_t
00466 l4_thread_vcpu_resume_commit_u(l4_cap_idx_t thread,
00467 l4_msgtag_t tag, l4_utcb_t *utcb) L4_NOTHROW;
00468
00469
00489 L4_INLINE l4_msgtag_t
00490 l4_thread_vcpu_control(l4_cap_idx_t thread, l4_addr_t vcpu_state) L4_NOTHROW;
00491
00499 L4_INLINE l4_msgtag_t

```

```

00500 l4_thread_vcpu_control_u(l4_cap_idx_t thread, l4_addr_t vcpu_state,
00501 l4_utcb_t *utcb) L4_NOTHROW;
00502
00534 L4_INLINE l4_msgtag_t
00535 l4_thread_vcpu_control_ext(l4_cap_idx_t thread, l4_addr_t ext_vcpu_state) L4_NOTHROW;
00536
00544 L4_INLINE l4_msgtag_t
00545 l4_thread_vcpu_control_ext_u(l4_cap_idx_t thread, l4_addr_t ext_vcpu_state,
00546 l4_utcb_t *utcb) L4_NOTHROW;
00547
00548
00572 L4_INLINE l4_msgtag_t
00573 l4_thread_register_del_irq(l4_cap_idx_t thread, l4_cap_idx_t irq) L4_NOTHROW;
00574
00579 L4_INLINE l4_msgtag_t
00580 l4_thread_register_del_irq_u(l4_cap_idx_t thread, l4_cap_idx_t irq,
00581 l4_utcb_t *utcb) L4_NOTHROW;
00582
00604 L4_INLINE l4_msgtag_t
00605 l4_thread_modify_sender_start(void) L4_NOTHROW;
00606
00611 L4_INLINE l4_msgtag_t
00612 l4_thread_modify_sender_start_u(l4_utcb_t *u) L4_NOTHROW;
00613
00638 L4_INLINE int
00639 l4_thread_modify_sender_add(l4_umword_t match_mask,
00640 l4_umword_t match,
00641 l4_umword_t del_bits,
00642 l4_umword_t add_bits,
00643 l4_msgtag_t *tag) L4_NOTHROW;
00644
00649 L4_INLINE int
00650 l4_thread_modify_sender_add_u(l4_umword_t match_mask,
00651 l4_umword_t match,
00652 l4_umword_t del_bits,
00653 l4_umword_t add_bits,
00654 l4_msgtag_t *tag, l4_utcb_t *u) L4_NOTHROW;
00655
00681 L4_INLINE l4_msgtag_t
00682 l4_thread_modify_sender_commit(l4_cap_idx_t thread, l4_msgtag_t tag) L4_NOTHROW;
00683
00688 L4_INLINE l4_msgtag_t
00689 l4_thread_modify_sender_commit_u(l4_cap_idx_t thread, l4_msgtag_t tag,
00690 l4_utcb_t *u) L4_NOTHROW;
00691
00692
00713 L4_INLINE l4_msgtag_t
00714 l4_thread_register_doorbell_irq(l4_cap_idx_t thread,
00715 l4_cap_idx_t irq) L4_NOTHROW;
00716
00721 L4_INLINE l4_msgtag_t
00722 l4_thread_register_doorbell_irq_u(l4_cap_idx_t thread, l4_cap_idx_t irq,
00723 l4_utcb_t *u) L4_NOTHROW;
00724
00725
00732 enum L4_thread_ops
00733 {
00734 L4_THREAD_CONTROL_OP = 0UL,
00735 L4_THREAD_EX_REGS_OP = 1UL,
00736 L4_THREAD_SWITCH_OP = 2UL,
00737 L4_THREAD_STATS_OP = 3UL,
00738 L4_THREAD_VCPU_RESUME_OP = 4UL,
00739 L4_THREAD_REGISTER_DELETE_IRQ_OP = 5UL,
00740 L4_THREAD_MODIFY_SENDER_OP = 6UL,
00741 L4_THREAD_VCPU_CONTROL_OP = 7UL,
00742 L4_THREAD_VCPU_CONTROL_EXT_OP = L4_THREAD_VCPU_CONTROL_OP | 0x10000,
00743 L4_THREAD_REGISTER_DOORBELL_IRQ_OP = 8UL,
00744 L4_THREAD_X86_GDT_OP = 0x10UL,
00745 L4_THREAD_ARM_TPIDRURO_OP = 0x10UL,
00746 L4_THREAD_AMD64_SET_SEGMENT_BASE_OP = 0x12UL,
00747 L4_THREAD_AMD64_GET_SEGMENT_INFO_OP = 0x13UL,
00748 L4_THREAD_OPCODE_MASK = 0xffff,
00749 };
00750
00761 enum L4_thread_control_flags
00762 {
00764 L4_THREAD_CONTROL_SET_PAGER = 0x0010000,
00766 L4_THREAD_CONTROL_BIND_TASK = 0x0200000,
00768 L4_THREAD_CONTROL_ALIEN = 0x0400000,
00770 L4_THREAD_CONTROL_SET_EXC_HANDLER = 0x1000000,
00771 };
00772
00782 enum L4_thread_control_mr_indices
00783 {
00784 L4_THREAD_CONTROL_MR_IDX_FLAGS = 0,
00785 L4_THREAD_CONTROL_MR_IDX_PAGER = 1,
00786 L4_THREAD_CONTROL_MR_IDX_EXC_HANDLER = 2,

```

```

00787 L4_THREAD_CONTROL_MR_IDX_FLAG_VALS = 4,
00788 L4_THREAD_CONTROL_MR_IDX_BIND_UTCB = 5,
00789 L4_THREAD_CONTROL_MR_IDX_BIND_TASK = 6,
00790 };
00791
00797 enum L4_thread_ex_regs_flags
00798 {
00799 L4_THREAD_EX_REGS_CANCEL = 0x10000UL,
00800 L4_THREAD_EX_REGS_TRIGGER_EXCEPTION = 0x20000UL,
00801
00802 L4_THREAD_EX_REGS_ARCH_MASK = 0xff000000UL,
00803 };
00804
00805
00806 /* IMPLEMENTATION ----- */
00807
00808 #include <l4/sys/ipc.h>
00809 #include <l4/sys/types.h>
00810
00811 L4_INLINE l4_msgtag_t
00812 l4_thread_ex_regs_u(l4_cap_idx_t thread, l4_addr_t ip, l4_addr_t sp,
00813 l4_umword_t flags, l4_utcb_t *utcb) L4_NOTHROW
00814 {
00815 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00816 v->mr[0] = L4_THREAD_EX_REGS_OP | flags;
00817 v->mr[1] = ip;
00818 v->mr[2] = sp;
00819 return l4_ipc_call(thread, utcb, l4_msgtag(L4_PROTO_THREAD, 3, 0, 0), L4_IPC_NEVER);
00820 }
00821
00822 L4_INLINE l4_msgtag_t
00823 l4_thread_ex_regs_ret_u(l4_cap_idx_t thread, l4_addr_t *ip, l4_addr_t *sp,
00824 l4_umword_t *flags, l4_utcb_t *utcb) L4_NOTHROW
00825 {
00826 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00827 l4_msgtag_t ret = l4_thread_ex_regs_u(thread, *ip, *sp, *flags, utcb);
00828 if (l4_error_u(ret, utcb))
00829 return ret;
00830
00831 *flags = v->mr[0];
00832 *ip = v->mr[1];
00833 *sp = v->mr[2];
00834 return ret;
00835 }
00836
00837 L4_INLINE void
00838 l4_thread_control_start_u(l4_utcb_t *utcb) L4_NOTHROW
00839 {
00840 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00841 v->mr[L4_THREAD_CONTROL_MR_IDX_FLAGS] = L4_THREAD_CONTROL_OP;
00842 }
00843
00844 L4_INLINE void
00845 l4_thread_control_pager_u(l4_cap_idx_t pager, l4_utcb_t *utcb) L4_NOTHROW
00846 {
00847 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00848 v->mr[L4_THREAD_CONTROL_MR_IDX_FLAGS] |= L4_THREAD_CONTROL_SET_PAGER;
00849 v->mr[L4_THREAD_CONTROL_MR_IDX_PAGER] = pager;
00850 }
00851
00852 L4_INLINE void
00853 l4_thread_control_exc_handler_u(l4_cap_idx_t exc_handler,
00854 l4_utcb_t *utcb) L4_NOTHROW
00855 {
00856 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00857 v->mr[L4_THREAD_CONTROL_MR_IDX_FLAGS] |= L4_THREAD_CONTROL_SET_EXC_HANDLER;
00858 v->mr[L4_THREAD_CONTROL_MR_IDX_EXC_HANDLER] = exc_handler;
00859 }
00860
00861 L4_INLINE void
00862 l4_thread_control_bind_u(l4_utcb_t *thread_utcb, l4_cap_idx_t task,
00863 l4_utcb_t *utcb) L4_NOTHROW
00864 {
00865 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00866 v->mr[L4_THREAD_CONTROL_MR_IDX_FLAGS] |= L4_THREAD_CONTROL_BIND_TASK;
00867 v->mr[L4_THREAD_CONTROL_MR_IDX_BIND_UTCB] = (l4_addr_t)thread_utcb;
00868 v->mr[L4_THREAD_CONTROL_MR_IDX_BIND_TASK] = L4_ITEM_MAP;
00869 v->mr[L4_THREAD_CONTROL_MR_IDX_BIND_TASK + 1] = l4_obj_fpage(task, 0, L4_CAP_FPAGE_RWS).raw;
00870 }
00871
00872 L4_INLINE void
00873 l4_thread_control_alien_u(l4_utcb_t *utcb, int on) L4_NOTHROW
00874 {
00875 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00876 v->mr[L4_THREAD_CONTROL_MR_IDX_FLAGS] |= L4_THREAD_CONTROL_ALIEN;
00877 v->mr[L4_THREAD_CONTROL_MR_IDX_FLAG_VALS] |= on ? L4_THREAD_CONTROL_ALIEN : 0;
00878 }

```

```

00879
00880 L4_INLINE l4_msgtag_t
00881 l4_thread_control_commit_u(l4_cap_idx_t thread, l4_utcb_t *utcb) L4_NOTHROW
00882 {
00883 int items = 0;
00884 if (l4_utcb_mr_u(utcb)->mr[L4_THREAD_CONTROL_MR_IDX_FLAGS] & L4_THREAD_CONTROL_BIND_TASK)
00885 items = 1;
00886 return l4_ipc_call(thread, utcb, l4_msgtag(L4_PROTO_THREAD, 6, items, 0), L4_IPC_NEVER);
00887 }
00888
00889
00890 L4_INLINE l4_msgtag_t
00891 l4_thread_yield(void) L4_NOTHROW
00892 {
00893 l4_ipc_receive(L4_INVALID_CAP, NULL, L4_IPC_BOTH_TIMEOUT_0);
00894 return l4_msgtag(0, 0, 0, 0);
00895 }
00896
00897 /* Preliminary, to be changed */
00898 L4_INLINE l4_msgtag_t
00899 l4_thread_switch_u(l4_cap_idx_t to_thread, l4_utcb_t *utcb) L4_NOTHROW
00900 {
00901 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00902 v->mr[0] = L4_THREAD_SWITCH_OP;
00903 return l4_ipc_call(to_thread, utcb, l4_msgtag(L4_PROTO_THREAD, 1, 0, 0), L4_IPC_NEVER);
00904 }
00905
00906
00907 L4_INLINE l4_msgtag_t
00908 l4_thread_stats_time_u(l4_cap_idx_t thread, l4_kernel_clock_t *us,
00909 l4_utcb_t *utcb) L4_NOTHROW
00910 {
00911 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00912 l4_msgtag_t res;
00913
00914 v->mr[0] = L4_THREAD_STATS_OP;
00915
00916 res = l4_ipc_call(thread, utcb, l4_msgtag(L4_PROTO_THREAD, 1, 0, 0), L4_IPC_NEVER);
00917
00918 if (l4_msgtag_has_error(res))
00919 return res;
00920
00921 *us = v->mr64[l4_utcb_mr64_idx(0)];
00922
00923 return res;
00924 }
00925
00926 L4_INLINE l4_msgtag_t
00927 l4_thread_vcpu_resume_start_u(l4_utcb_t *utcb) L4_NOTHROW
00928 {
00929 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
00930 v->mr[0] = L4_THREAD_VCPU_RESUME_OP;
00931 return l4_msgtag(L4_PROTO_THREAD, 1, 0, 0);
00932 }
00933
00934 L4_INLINE l4_msgtag_t
00935 l4_thread_vcpu_resume_commit_u(l4_cap_idx_t thread,
00936 l4_msgtag_t tag, l4_utcb_t *utcb) L4_NOTHROW
00937 {
00938 return l4_ipc_call(thread, utcb, tag, L4_IPC_NEVER);
00939 }
00940
00941 L4_INLINE l4_msgtag_t
00942 l4_thread_ex_regs(l4_cap_idx_t thread, l4_addr_t ip, l4_addr_t sp,
00943 l4_umword_t flags) L4_NOTHROW
00944 {
00945 return l4_thread_ex_regs_u(thread, ip, sp, flags, l4_utcb());
00946 }
00947
00948 L4_INLINE l4_msgtag_t
00949 l4_thread_ex_regs_ret(l4_cap_idx_t thread, l4_addr_t *ip, l4_addr_t *sp,
00950 l4_umword_t *flags) L4_NOTHROW
00951 {
00952 return l4_thread_ex_regs_ret_u(thread, ip, sp, flags, l4_utcb());
00953 }
00954
00955 L4_INLINE void
00956 l4_thread_control_start(void) L4_NOTHROW
00957 {
00958 l4_thread_control_start_u(l4_utcb());
00959 }
00960
00961 L4_INLINE void
00962 l4_thread_control_pager(l4_cap_idx_t pager) L4_NOTHROW
00963 {
00964 l4_thread_control_pager_u(pager, l4_utcb());
00965 }

```

```

00966
00967 L4_INLINE void
00968 l4_thread_control_exc_handler(l4_cap_idx_t exc_handler) L4_NOTHROW
00969 {
00970 l4_thread_control_exc_handler_u(exc_handler, l4_utcb());
00971 }
00972
00973
00974 L4_INLINE void
00975 l4_thread_control_bind(l4_utcb_t *thread_utcb, l4_cap_idx_t task) L4_NOTHROW
00976 {
00977 l4_thread_control_bind_u(thread_utcb, task, l4_utcb());
00978 }
00979
00980 L4_INLINE void
00981 l4_thread_control_alien(int on) L4_NOTHROW
00982 {
00983 l4_thread_control_alien_u(l4_utcb(), on);
00984 }
00985
00986 L4_INLINE l4_msgtag_t
00987 l4_thread_control_commit(l4_cap_idx_t thread) L4_NOTHROW
00988 {
00989 return l4_thread_control_commit_u(thread, l4_utcb());
00990 }
00991
00992
00993
00994
00995 L4_INLINE l4_msgtag_t
00996 l4_thread_switch(l4_cap_idx_t to_thread) L4_NOTHROW
00997 {
00998 return l4_thread_switch_u(to_thread, l4_utcb());
00999 }
01000
01001
01002
01003
01004 L4_INLINE l4_msgtag_t
01005 l4_thread_stats_time(l4_cap_idx_t thread, l4_kernel_clock_t *us) L4_NOTHROW
01006 {
01007 return l4_thread_stats_time_u(thread, us, l4_utcb());
01008 }
01009
01010 L4_INLINE l4_msgtag_t
01011 l4_thread_vcpu_resume_start(void) L4_NOTHROW
01012 {
01013 return l4_thread_vcpu_resume_start_u(l4_utcb());
01014 }
01015
01016 L4_INLINE l4_msgtag_t
01017 l4_thread_vcpu_resume_commit(l4_cap_idx_t thread,
01018 l4_msgtag_t tag) L4_NOTHROW
01019 {
01020 return l4_thread_vcpu_resume_commit_u(thread, tag, l4_utcb());
01021 }
01022
01023
01024 L4_INLINE l4_msgtag_t
01025 l4_thread_register_del_irq_u(l4_cap_idx_t thread, l4_cap_idx_t irq,
01026 l4_utcb_t *u) L4_NOTHROW
01027 {
01028 l4_msg_regs_t *m = l4_utcb_mr_u(u);
01029 m->mr[0] = L4_THREAD_REGISTER_DELETE_IRQ_OP;
01030 m->mr[1] = l4_map_obj_control(0,0);
01031 m->mr[2] = l4_obj_fpage(irq, 0, L4_CAP_FPAGE_RWS).raw;
01032 return l4_ipc_call(thread, u, l4_msgtag(L4_PROTO_THREAD, 1, 1, 0), L4_IPC_NEVER);
01033 }
01034
01035
01036 L4_INLINE l4_msgtag_t
01037 l4_thread_register_del_irq(l4_cap_idx_t thread, l4_cap_idx_t irq) L4_NOTHROW
01038 {
01039 return l4_thread_register_del_irq_u(thread, irq, l4_utcb());
01040 }
01041
01042
01043 L4_INLINE l4_msgtag_t
01044 l4_thread_vcpu_control_u(l4_cap_idx_t thread, l4_addr_t vcpu_state,
01045 l4_utcb_t *utcb) L4_NOTHROW
01046 {
01047 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
01048 v->mr[0] = L4_THREAD_VCPU_CONTROL_OP;
01049 v->mr[1] = vcpu_state;
01050 return l4_ipc_call(thread, utcb, l4_msgtag(L4_PROTO_THREAD, 2, 0, 0), L4_IPC_NEVER);
01051 }
01052

```

```

01053 L4_INLINE l4_msgtag_t
01054 l4_thread_vcpu_control(l4_cap_idx_t thread, l4_addr_t vcpu_state) L4_NOTHROW
01055 { return l4_thread_vcpu_control_u(thread, vcpu_state, l4_utcb()); }
01056
01057
01058 L4_INLINE l4_msgtag_t
01059 l4_thread_vcpu_control_ext_u(l4_cap_idx_t thread, l4_addr_t ext_vcpu_state,
01060 l4_utcb_t *utcb) L4_NOTHROW
01061 {
01062 l4_msg_regs_t *v = l4_utcb_mr_u(utcb);
01063 v->mr[0] = L4_THREAD_VCPU_CONTROL_EXT_OP;
01064 v->mr[1] = ext_vcpu_state;
01065 return l4_ipc_call(thread, utcb, l4_msgtag(L4_PROTO_THREAD, 2, 0, 0), L4_IPC_NEVER);
01066 }
01067
01068 L4_INLINE l4_msgtag_t
01069 l4_thread_vcpu_control_ext(l4_cap_idx_t thread, l4_addr_t ext_vcpu_state) L4_NOTHROW
01070 { return l4_thread_vcpu_control_ext_u(thread, ext_vcpu_state, l4_utcb()); }
01071
01072 L4_INLINE l4_msgtag_t
01073 l4_thread_modify_sender_start_u(l4_utcb_t *u) L4_NOTHROW
01074 {
01075 l4_msg_regs_t *m = l4_utcb_mr_u(u);
01076 m->mr[0] = L4_THREAD_MODIFY_SENDER_OP;
01077 return l4_msgtag(L4_PROTO_THREAD, 1, 0, 0);
01078 }
01079
01080 L4_INLINE int
01081 l4_thread_modify_sender_add_u(l4_umword_t match_mask,
01082 l4_umword_t match,
01083 l4_umword_t del_bits,
01084 l4_umword_t add_bits,
01085 l4_msgtag_t *tag, l4_utcb_t *u) L4_NOTHROW
01086 {
01087 l4_msg_regs_t *m = l4_utcb_mr_u(u);
01088 unsigned w = l4_msgtag_words(*tag);
01089 if (w >= L4_UTCB_GENERIC_DATA_SIZE - 4)
01090 return -L4_ENOMEM;
01091
01092 m->mr[w] = match_mask;
01093 m->mr[w+1] = match;
01094 m->mr[w+2] = del_bits;
01095 m->mr[w+3] = add_bits;
01096
01097 *tag = l4_msgtag(l4_msgtag_label(*tag), w + 4, 0, 0);
01098
01099 return 0;
01100 }
01101
01102 L4_INLINE l4_msgtag_t
01103 l4_thread_modify_sender_commit_u(l4_cap_idx_t thread, l4_msgtag_t tag,
01104 l4_utcb_t *u) L4_NOTHROW
01105 {
01106 return l4_ipc_call(thread, u, tag, L4_IPC_NEVER);
01107 }
01108
01109 L4_INLINE l4_msgtag_t
01110 l4_thread_modify_sender_start(void) L4_NOTHROW
01111 {
01112 return l4_thread_modify_sender_start_u(l4_utcb());
01113 }
01114
01115 L4_INLINE int
01116 l4_thread_modify_sender_add(l4_umword_t match_mask,
01117 l4_umword_t match,
01118 l4_umword_t del_bits,
01119 l4_umword_t add_bits,
01120 l4_msgtag_t *tag) L4_NOTHROW
01121 {
01122 return l4_thread_modify_sender_add_u(match_mask, match,
01123 del_bits, add_bits, tag, l4_utcb());
01124 }
01125
01126 L4_INLINE l4_msgtag_t
01127 l4_thread_modify_sender_commit(l4_cap_idx_t thread, l4_msgtag_t tag) L4_NOTHROW
01128 {
01129 return l4_thread_modify_sender_commit_u(thread, tag, l4_utcb());
01130 }
01131
01132
01133 L4_INLINE l4_msgtag_t
01134 l4_thread_register_doorbell_irq_u(l4_cap_idx_t thread, l4_cap_idx_t irq,
01135 l4_utcb_t *u) L4_NOTHROW
01136 {
01137 l4_msg_regs_t *m = l4_utcb_mr_u(u);
01138 m->mr[0] = L4_THREAD_REGISTER_DOORBELL_IRQ_OP;
01139 m->mr[1] = l4_map_obj_control(0, 0);

```

```

01140 m->mr[2] = l4_obj_fpage(irq, 0, L4_CAP_FPAGE_RWS).raw;
01141 return l4_ipc_call(thread, u, l4_msgtag(L4_PROTO_THREAD, 1, 1, 0),
01142 L4_IPC_NEVER);
01143 }
01144
01145 L4_INLINE l4_msgtag_t
01146 l4_thread_register_doorbell_irq(l4_cap_idx_t thread,
01147 l4_cap_idx_t irq) L4_NOTHROW
01148 {
01149 return l4_thread_register_doorbell_irq_u(thread, irq, l4_utcb());
01150 }

```

## 17.672 l4/util/thread.h File Reference

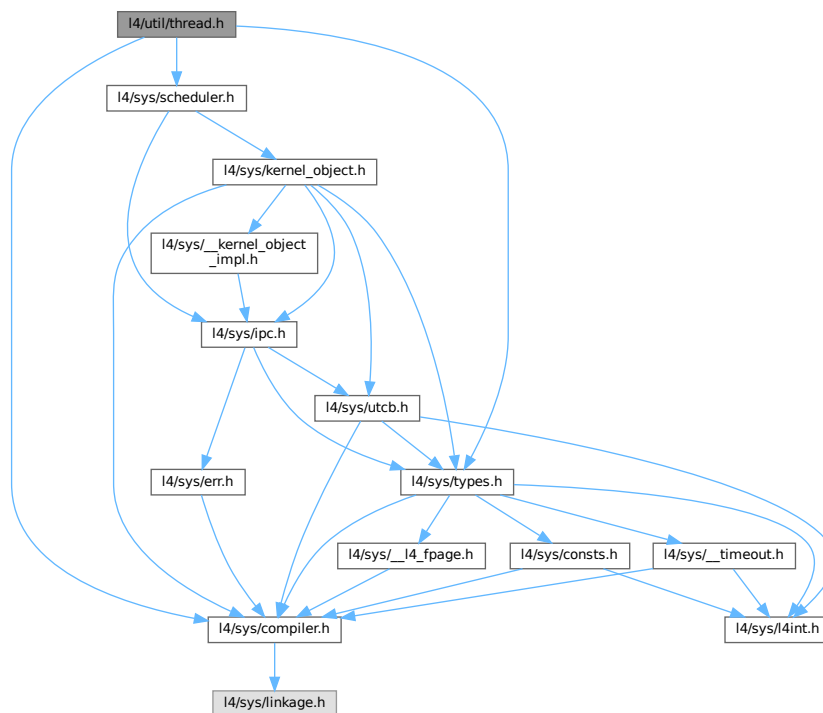
Low-level Thread Functions.

```

#include <l4/sys/compiler.h>
#include <l4/sys/types.h>
#include <l4/sys/scheduler.h>

```

Include dependency graph for thread.h:



### Macros

- `#define __L4UTIL_THREAD_FUNC(name)`  
*Defines a wrapper function that sets up the registers according to the calling conventions for the architecture.*



## 17.672.1 Detailed Description

Low-level Thread Functions.

Date

1997

Author

Sebastian Schönberg

Definition in file [thread.h](#).

## 17.672.2 Macro Definition Documentation

### 17.672.2.1 \_\_L4UTIL\_THREAD\_FUNC

```
#define __L4UTIL_THREAD_FUNC(
 name)
```

**Value:**

```
void L4_NORETURN name(void)
```

Defines a wrapper function that sets up the registers according to the calling conventions for the architecture.

Use this as a function header when starting a low-level thread where only stack and instruction pointer are in a well-defined state.

Example:

```
L4UTIL_THREAD_FUNC(helper_thread) { l4_infinite_loop(); }
```

```
thread_cap->ex_regs((l4_umword_t)helper_thread, stack_addr);
```

Definition at line 71 of file [thread.h](#).

## 17.673 thread.h

[Go to the documentation of this file.](#)

```

00001
00007
00008 /*
00009 * (c) 2003-2009 Author(s)
00010 * economic rights: Technische Universität Dresden (Germany)
00011 * License: see LICENSE.spdx (in this directory or the directories above)
00012 */
00013
00014 #ifndef __L4_THREAD_H
00015 #define __L4_THREAD_H
00016
00017 #include <l4/sys/compiler.h>
00018 #include <l4/sys/types.h>
00019 #include <l4/sys/scheduler.h>
00020
00021 L4_BEGIN_DECLS
00022
00027
00045 L4_CV long
00046 l4util_create_thread(l4_cap_idx_t id, l4_utcb_t *thread_utcb,
00047 l4_cap_idx_t factory,
00048 l4_umword_t pc, l4_umword_t sp, l4_cap_idx_t pager,
00049 l4_cap_idx_t task,
00050 l4_cap_idx_t scheduler, l4_sched_param_t scp) L4_NOTHROW;
00051
00052 L4_END_DECLS
00053
00054 #ifndef L4UTIL_THREAD_FUNC
00071 #define __L4UTIL_THREAD_FUNC(name) void L4_NORETURN name(void)
00072 #define L4UTIL_THREAD_FUNC(name) __L4UTIL_THREAD_FUNC(name)
00073 #define __L4UTIL_THREAD_STATIC_FUNC(name) static L4_NORETURN void name(void)
00074 #define L4UTIL_THREAD_STATIC_FUNC(name) __L4UTIL_THREAD_STATIC_FUNC(name)
00075 #endif
00076
00077 #endif /* __L4_THREAD_H */

```

## 17.674 util.h

```

00001
00004 /*
00005 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00006 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00007 * Frank Mehnert <fm3@os.inf.tu-dresden.de>
00008 * economic rights: Technische Universität Dresden (Germany)
00009 * License: see LICENSE.spdx (in this directory or the directories above)
00010 */
00011 #ifndef __L4UTIL_UTIL_H__
00012 #define __L4UTIL_UTIL_H__
00013
00014 #include <l4/sys/types.h>
00015 #include <l4/sys/compiler.h>
00016 #include <l4/sys/ipc.h>
00017
00021
00022 L4_BEGIN_DECLS
00023
00034 L4_CV l4_timeout_s l4util_micros2l4to(l4_uint64_t us) L4_NOTHROW;
00035
00041 L4_CV void l4_sleep(l4_uint32_t ms) L4_NOTHROW;
00042
00049 L4_CV void l4_usleep(l4_uint64_t us) L4_NOTHROW;
00050
00056 L4_INLINE void l4_sleep_forever(void) L4_NOTHROW L4_NORETURN;
00057
00065 L4_INLINE void
00066 l4_touch_ro(const void *addr, unsigned size) L4_NOTHROW;
00067
00075 L4_INLINE void
00076 l4_touch_rw(const void *addr, unsigned size) L4_NOTHROW;
00077
00078
00079
00080 /*
00081 * Implementations
00082 */
00083
00084 L4_INLINE void

```

```

00085 l4_sleep_forever(void) L4_NOTHROW
00086 {
00087 for (;;)
00088 l4_ipc_sleep(L4_IPC_NEVER);
00089 }
00090
00091 L4_INLINE void
00092 l4_touch_ro(const void *addr, unsigned size) L4_NOTHROW
00093 {
00094 l4_addr_t b, e;
00095
00096 b = l4_trunc_page((l4_addr_t)addr);
00097 e = l4_trunc_page((l4_addr_t)addr + size - 1);
00098
00099 for (; b <= e; b += L4_PAGESIZE)
00100 (void) (*(volatile char *)b);
00101 }
00102
00103
00104 L4_INLINE void
00105 l4_touch_rw(const void *addr, unsigned size) L4_NOTHROW
00106 {
00107 l4_addr_t b, e;
00108
00109 b = l4_trunc_page((l4_addr_t)addr);
00110 e = l4_trunc_page((l4_addr_t)addr + size - 1);
00111
00112 for (; b <= e; b += L4_PAGESIZE)
00113 *(volatile char *)b |= 0;
00114 }
00115
00116 L4_END_DECLS
00117
00118 #endif /* __L4UTIL__UTIL_H__ */

```

## 17.675 vbus

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2011 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include <l4/vbus/vbus.h>
00011 #include <l4/vbus/vbus_pm.h>
00012 #include <l4/sys/icu>
00013
00014 #include <l4/re/dataspace>
00015 #include <l4/re/dma_space>
00016 #include <l4/re/event>
00017 #include <l4/re/inhibitor>
00018
00036
00040 namespace L4vbus {
00041
00042 class Vbus;
00043
00049 template<typename DEC>
00050 class Pm
00051 {
00052 private:
00053 DEC const *self() const { return static_cast<DEC const *>(this); }
00054 DEC *self() { return static_cast<DEC *>(this); }
00055 public:
00063 int pm_suspend() const
00064 { return l4vbus_pm_suspend(self()->bus_cap().cap(), self()->dev_handle()); }
00065
00074 int pm_resume() const
00075 { return l4vbus_pm_resume(self()->bus_cap().cap(), self()->dev_handle()); }
00076 };
00077
00078
00083 class Device : public Pm<Device>
00084 {
00085 public:
00089 Device() : _dev(L4VBUS_NULL) {}
00090
00100 Device(L4::Cap<Vbus> bus, l4vbus_device_handle_t dev)
00101 : _bus(bus), _dev(dev) {}
00102

```

```

00107 L4::Cap<Vbus> bus_cap() const { return _bus; }
00108
00116 l4vbus_device_handle_t dev_handle() const { return _dev; }
00117
00118
00148 int device_by_hid(Device *child, char const *hid,
00149 int depth = L4VBUS_MAX_DEPTH,
00150 l4vbus_device_t *devinfo = 0) const
00151 {
00152 child->_bus = _bus;
00153 return l4vbus_get_device_by_hid(_bus.cap(), _dev, &child->_dev, hid,
00154 depth, devinfo);
00155 }
00156
00171 int next_device(Device *child, int depth = L4VBUS_MAX_DEPTH,
00172 l4vbus_device_t *devinfo = 0) const
00173 {
00174 child->_bus = _bus;
00175 return l4vbus_get_next_device(_bus.cap(), _dev, &child->_dev, depth,
00176 devinfo);
00177 }
00178
00189 int device(l4vbus_device_t *devinfo) const
00190 { return l4vbus_get_device(_bus.cap(), _dev, devinfo); }
00191
00209 int get_resource(unsigned res_idx, l4vbus_resource_t *res) const
00210 {
00211 return l4vbus_get_resource(_bus.cap(), _dev, res_idx, res);
00212 }
00213
00223 int is_compatible(char const *cid) const
00224 { return l4vbus_is_compatible(_bus.cap(), _dev, cid); }
00225
00230 bool operator == (Device const &o) const
00231 {
00232 return _bus == o._bus && _dev == o._dev;
00233 }
00234
00239 bool operator != (Device const &o) const
00240 {
00241 return _bus != o._bus || _dev != o._dev;
00242 }
00243
00244 protected:
00245 L4::Cap<Vbus> _bus;
00247 l4vbus_device_handle_t _dev;
00248 };
00249
00260 class Icu : public Device
00261 {
00262 public:
00264 enum Src_types
00265 {
00273 Src_dev_handle = L4VBUS_ICU_SRC_DEV_HANDLE
00274 };
00275
00285 int vicu(L4::Cap<L4::Icu> icu) const
00286 {
00287 return l4vbus_vicu_get_cap(_bus.cap(), _dev, icu.cap());
00288 }
00289 };
00290
00298 class Vbus : public L4::Kobject_3t<Vbus, L4Re::Dataspace, L4Re::Inhibitor, L4Re::Event>
00299 {
00300 public:
00301
00311 int request_ioport(l4vbus_resource_t *res) const
00312 {
00313 return l4vbus_request_ioport(cap(), res);
00314 }
00315
00323 int release_ioport(l4vbus_resource_t *res) const
00324 {
00325 return l4vbus_release_ioport(cap(), res);
00326 }
00327
00336 Device root() const
00337 {
00338 return Device(L4::Cap<Vbus>(cap()), L4VBUS_ROOT_BUS);
00339 }
00340
00357 int assign_dma_domain(unsigned domain_id, unsigned flags,
00358 L4::Cap<L4Re::Dma_space> dma_space) const
00359 {
00360 flags |= L4VBUS_DMAD_L4RE_DMA_SPACE;
00361 flags &= ~L4VBUS_DMAD_KERNEL_DMA_SPACE;
00362 return l4vbus_assign_dma_domain(cap(), domain_id, flags, dma_space.cap());

```

```

00363 }
00364
00382 int assign_dma_domain(unsigned domain_id, unsigned flags,
00383 L4::Cap<L4::Task> dma_space) const
00384 {
00385 flags |= L4VBUS_DMAD_KERNEL_DMA_SPACE;
00386 flags &= ~L4VBUS_DMAD_L4RE_DMA_SPACE;
00387 return l4vbus_assign_dma_domain(cap(), domain_id, flags, dma_space.cap());
00388 }
00389
00390 typedef L4::Typeid::Raw_ipc<Vbus> Rpcs;
00391 };
00392
00393 } // namespace L4vbus

```

## 17.676 l4/vbus/vbus.h File Reference

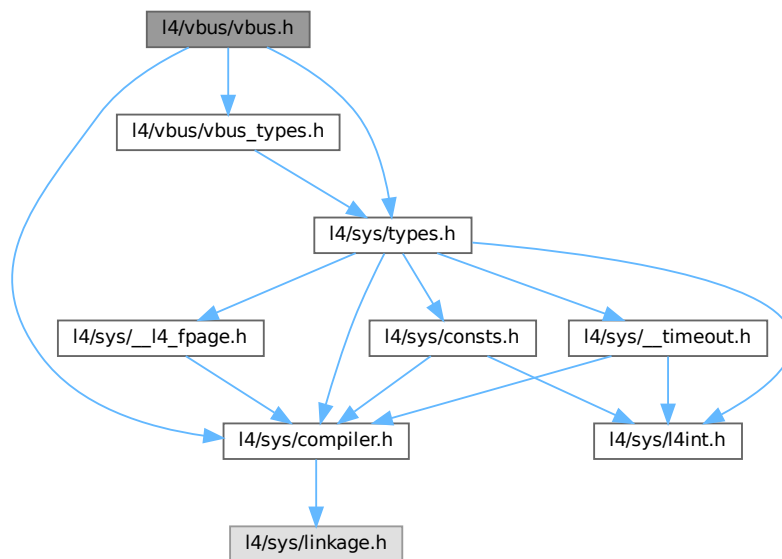
Description of the vbus C API.

```

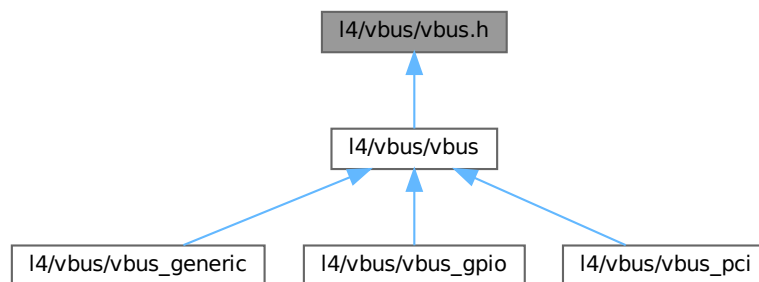
#include <l4/sys/compiler.h>
#include <l4/vbus/vbus_types.h>
#include <l4/sys/types.h>

```

Include dependency graph for vbus.h:



This graph shows which files directly or indirectly include this file:



## Enumerations

- enum { `L4VBUS_NULL` = -1L , `L4VBUS_ROOT_BUS` = 0 }
- Constants for device nodes.
- enum `l4vbus_icu_src_types` { `L4VBUS_ICU_SRC_DEV_HANDLE` = 1ULL << 63 }
- Flags that can be used with the ICU on a vbus device.
- enum `L4vbus_dma_domain_assign_flags` { `L4VBUS_DMAD_UNBIND` = 0 , `L4VBUS_DMAD_BIND` = 1 , `L4VBUS_DMAD_L4RE_DMA_SPACE` = 0 , `L4VBUS_DMAD_KERNEL_DMA_SPACE` = 2 }
- Flags for `l4vbus_assign_dma_domain()`.

## Functions

- `L4_BEGIN_DECLS` int `l4vbus_get_device_by_hid` (`l4_cap_idx_t` vbus, `l4vbus_device_handle_t` parent, `l4vbus_device_handle_t` \*child, char const \*hid, int depth, `l4vbus_device_t` \*devinfo)
- Find a device by the hardware interface identifier (HID).
- int `l4vbus_get_next_device` (`l4_cap_idx_t` vbus, `l4vbus_device_handle_t` parent, `l4vbus_device_handle_t` \*child, int depth, `l4vbus_device_t` \*devinfo)
- Find next child following *child*.
- int `l4vbus_get_device` (`l4_cap_idx_t` vbus, `l4vbus_device_handle_t` dev, `l4vbus_device_t` \*devinfo)
- Obtain detailed information about a Vbus device.
- int `l4vbus_get_resource` (`l4_cap_idx_t` vbus, `l4vbus_device_handle_t` dev, unsigned res\_idx, `l4vbus_resource_t` \*res)
- Obtain the resource description of an individual device resource.
- int `l4vbus_is_compatible` (`l4_cap_idx_t` vbus, `l4vbus_device_handle_t` dev, char const \*cid)
- Check if the given device has a compatibility ID (CID) or HID that matches cid.
- int `l4vbus_get_hid` (`l4_cap_idx_t` vbus, `l4vbus_device_handle_t` dev, char \*hid, unsigned long max\_len)
- Get the HID (hardware identifier) of a device.
- int `l4vbus_get_adr` (`l4_cap_idx_t` vbus, `l4vbus_device_handle_t` dev, `l4_uint32_t` \*adr)
- Get the bus-specific address of a device.
- int `l4vbus_request_ioport` (`l4_cap_idx_t` vbus, `l4vbus_resource_t` const \*res)
- Request an IO port resource.
- int `l4vbus_assign_dma_domain` (`l4_cap_idx_t` vbus, unsigned domain\_id, unsigned flags, `l4_cap_idx_t` dma\_space)
- Bind or unbind a kernel DMA space or a `L4Re::Dma_space` to a DMA domain.
- int `l4vbus_release_ioport` (`l4_cap_idx_t` vbus, `l4vbus_resource_t` const \*res)
- Release a previously requested IO port resource.
- int `l4vbus_vicu_get_cap` (`l4_cap_idx_t` vbus, `l4vbus_device_handle_t` icu, `l4_cap_idx_t` cap)
- Get capability of ICU.

## 17.676.1 Detailed Description

Description of the vbus C API.

Definition in file [vbus.h](#).

## 17.676.2 Enumeration Type Documentation

### 17.676.2.1 anonymous enum

anonymous enum

Constants for device nodes.

#### Enumerator

|                 |                          |
|-----------------|--------------------------|
| L4VBUS_NULL     | NULL device.             |
| L4VBUS_ROOT_BUS | Root device on the vbus. |

Definition at line 20 of file [vbus.h](#).

### 17.676.2.2 l4vbus\_icu\_src\_types

enum [l4vbus\\_icu\\_src\\_types](#)

Flags that can be used with the ICU on a vbus device.

#### Enumerator

|                           |                                                                                                                                                                                                                                                   |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| L4VBUS_ICU_SRC_DEV_HANDLE | Flag to denote that the value should be interpreted as a device handle. This flag may be used in the <code>source</code> parameter in <a href="#">l4_icu_msi_info()</a> to denote that the ICU should interpret the source ID as a device handle. |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Definition at line 26 of file [vbus.h](#).

## 17.677 vbus.h

[Go to the documentation of this file.](#)

```
00001 /*
00002 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00004 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00013 #pragma once
00014
00015 #include <l4/sys/compiler.h>
00016 #include <l4/vbus/vbus_types.h>
00017 #include <l4/sys/types.h>
```

```

00018
00020 enum {
00021 L4VBUS_NULL = -1L,
00022 L4VBUS_ROOT_BUS = 0,
00023 };
00024
00026 enum l4vbus_icu_src_types {
00033 L4VBUS_ICU_SRC_DEV_HANDLE = 1ULL << 63
00034 };
00035
00053
00054 L4_BEGIN_DECLS
00055
00063 int L4_CV
00064 l4vbus_get_device_by_hid(l4_cap_idx_t vbus, l4vbus_device_handle_t parent,
00065 l4vbus_device_handle_t *child, char const *hid,
00066 int depth, l4vbus_device_t *devinfo);
00067
00083 int L4_CV
00084 l4vbus_get_next_device(l4_cap_idx_t vbus, l4vbus_device_handle_t parent,
00085 l4vbus_device_handle_t *child, int depth,
00086 l4vbus_device_t *devinfo);
00087
00101 int L4_CV
00102 l4vbus_get_device(l4_cap_idx_t vbus, l4vbus_device_handle_t dev,
00103 l4vbus_device_t *devinfo);
00104
00113 int L4_CV
00114 l4vbus_get_resource(l4_cap_idx_t vbus, l4vbus_device_handle_t dev,
00115 unsigned res_idx, l4vbus_resource_t *res);
00116
00117
00124 int L4_CV
00125 l4vbus_is_compatible(l4_cap_idx_t vbus, l4vbus_device_handle_t dev,
00126 char const *cid);
00127
00138 int L4_CV
00139 l4vbus_get_hid(l4_cap_idx_t vbus, l4vbus_device_handle_t dev, char *hid,
00140 unsigned long max_len);
00141
00152 int L4_CV
00153 l4vbus_get_adr(l4_cap_idx_t vbus, l4vbus_device_handle_t dev, l4_uint32_t *adr);
00154
00168 int L4_CV
00169 l4vbus_request_ioport(l4_cap_idx_t vbus, l4vbus_resource_t const *res);
00170
00174 enum L4vbus_dma_domain_assign_flags
00175 {
00177 L4VBUS_DMAD_UNBIND = 0,
00179 L4VBUS_DMAD_BIND = 1,
00181 L4VBUS_DMAD_L4RE_DMA_SPACE = 0,
00183 L4VBUS_DMAD_KERNEL_DMA_SPACE = 2,
00184 };
00185
00207 int L4_CV
00208 l4vbus_assign_dma_domain(l4_cap_idx_t vbus, unsigned domain_id,
00209 unsigned flags, l4_cap_idx_t dma_space);
00210
00219 int L4_CV
00220 l4vbus_release_ioport(l4_cap_idx_t vbus, l4vbus_resource_t const *res);
00221
00231 int L4_CV
00232 l4vbus_vicu_get_cap(l4_cap_idx_t vbus, l4vbus_device_handle_t icu,
00233 l4_cap_idx_t cap);
00234
00235 L4_END_DECLS
00236

```

## 17.678 vbus\_generic

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2009 Alexander Warg <warg@os.inf.tu-dresden.de>,
00004 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009
00010 #pragma once
00011
00012 #include <l4/cxx/ipc_stream>
00013 #include <l4/vbus/vbus_types.h>

```



```

00014 #include <l4/vbus/vbus>
00015
00016 inline void
00017 l4vbus_device_msg(l4vbus_device_handle_t handle, l4_uint32_t op,
00018 L4::Ipc::Iostream &s)
00019 {
00020 s « handle « op;
00021 }

```

## 17.679 vbus\_gpio

```

00001 // vi:set ft=c++: -- Mode: C++ --
00002 /*
00003 * (c) 2011 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include <l4/vbus/vbus>
00011 #include <l4/vbus/vbus_gpio.h>
00012
00013 namespace L4vbus {
00014
00015 class Gpio_pin : public Device
00016 {
00017 public:
00018 Gpio_pin(Device const &dev, unsigned pin)
00019 : Device(dev), _pin(pin)
00020 {}
00021
00022 int get() const
00023 {
00024 return l4vbus_gpio_get(_bus.cap(), _dev, _pin);
00025 }
00026
00027 int set(int value) const
00028 {
00029 return l4vbus_gpio_set(_bus.cap(), _dev, _pin, value);
00030 }
00031
00032 int setup(unsigned mode, unsigned value) const
00033 {
00034 return l4vbus_gpio_setup(_bus.cap(), _dev, _pin, mode, value);
00035 }
00036
00037 int config_pull(unsigned mode) const
00038 {
00039 return l4vbus_gpio_config_pull(_bus.cap(), _dev, _pin, mode);
00040 }
00041
00042 int config_pad(unsigned func, unsigned value) const
00043 {
00044 return l4vbus_gpio_config_pad(_bus.cap(), _dev, _pin, func, value);
00045 }
00046
00047 int config_get(unsigned func, unsigned *value) const
00048 {
00049 return l4vbus_gpio_config_get(_bus.cap(), _dev, _pin, func, value);
00050 }
00051
00052 int to_irq() const
00053 {
00054 return l4vbus_gpio_to_irq(_bus.cap(), _dev, _pin);
00055 }
00056
00057 unsigned pin() const { return _pin; }
00058
00059 protected:
00060 Gpio_pin() {}
00061 unsigned _pin;
00062 };
00063
00064 class Gpio_module : public Device
00065 {
00066 public:
00067 Gpio_module(Device dev)
00068 : Device(dev)
00069 {}
00070
00071 struct Pin_slice

```

```

00147 {
00148 Pin_slice(unsigned offset, unsigned mask) : offset(offset), mask(mask) {}
00149 unsigned offset, mask;
00150 };
00151
00166 int setup(Pin_slice const &mask, unsigned mode, unsigned value) const
00167 {
00168 return l4vbus_gpio_multi_setup(_bus.cap(), _dev, mask.offset, mask.mask,
00169 mode, value);
00170 }
00171
00185 int config_pad(Pin_slice const &mask, unsigned func, unsigned value) const
00186 {
00187 return l4vbus_gpio_multi_config_pad(_bus.cap(), _dev, mask.offset,
00188 mask.mask, func, value);
00189 }
00190
00201 int get(unsigned offset, unsigned *data) const
00202 {
00203 return l4vbus_gpio_multi_get(_bus.cap(), _dev, offset, data);
00204 }
00205
00217 int set(Pin_slice const &mask, unsigned data)
00218 {
00219 return l4vbus_gpio_multi_set(_bus.cap(), _dev, mask.offset,
00220 mask.mask, data);
00221 }
00222
00229 Gpio_pin pin(unsigned pin) const
00230 {
00231 return Gpio_pin(*this, pin);
00232 }
00233
00234 protected:
00235 Gpio_module() {}
00236 };
00237
00238 }

```

## 17.680 vbus\_gpio-ops.h

```

00001 /*
00002 * (c) 2011 Alexander Warg <warg@os.inf.tu-dresden.de>
00003 * economic rights: Technische Universität Dresden (Germany)
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #pragma once
00009
00010 #include <l4/vbus/vbus_interfaces.h>
00011
00012 enum L4vbus_gpio_op
00013 {
00014 L4VBUS_GPIO_OP_SETUP = L4VBUS_INTERFACE_GPIO « L4VBUS_IFACE_SHIFT,
00015 L4VBUS_GPIO_OP_CONFIG_PAD,
00016 L4VBUS_GPIO_OP_CONFIG_GET,
00017 L4VBUS_GPIO_OP_GET,
00018 L4VBUS_GPIO_OP_SET,
00019 L4VBUS_GPIO_OP_MULTI_SETUP,
00020 L4VBUS_GPIO_OP_MULTI_CONFIG_PAD,
00021 L4VBUS_GPIO_OP_MULTI_GET,
00022 L4VBUS_GPIO_OP_MULTI_SET,
00023 L4VBUS_GPIO_OP_TO_IRQ,
00024 L4VBUS_GPIO_OP_CONFIG_PULL
00025 };

```

## 17.681 vbus\_gpio.h

```

00001 /*
00002 * (c) 2011 Alexander Warg <warg@os.inf.tu-dresden.de>
00003 * economic rights: Technische Universität Dresden (Germany)
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/sys/compiler.h>

```

```

00010 #include <l4/sys/types.h>
00011 #include <l4/vbus/vbus_types.h>
00012
00018
00019 L4_BEGIN_DECLS
00020
00024 enum L4vbus_gpio_generic_func
00025 {
00026 L4VBUS_GPIO_SETUP_INPUT = 0x100,
00027 L4VBUS_GPIO_SETUP_OUTPUT = 0x200,
00028 L4VBUS_GPIO_SETUP_IRQ = 0x300,
00029 };
00030
00034 enum L4vbus_gpio_pull_modes
00035 {
00036 L4VBUS_GPIO_PIN_PULL_NONE = 0x100,
00037 L4VBUS_GPIO_PIN_PULL_UP = 0x200,
00038 L4VBUS_GPIO_PIN_PULL_DOWN = 0x300,
00039 };
00040
00048 int L4_CV
00049 l4vbus_gpio_setup(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00050 unsigned pin, unsigned mode, int value);
00051
00059 int L4_CV
00060 l4vbus_gpio_config_pull(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00061 unsigned pin, unsigned mode);
00062
00070 int L4_CV
00071 l4vbus_gpio_config_pad(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00072 unsigned pin, unsigned func, unsigned value);
00073
00081 int L4_CV
00082 l4vbus_gpio_config_get(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00083 unsigned pin, unsigned func, unsigned *value);
00084
00092 int L4_CV
00093 l4vbus_gpio_get(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00094 unsigned pin);
00095
00103 int L4_CV
00104 l4vbus_gpio_set(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00105 unsigned pin, int value);
00106
00115 int L4_CV
00116 l4vbus_gpio_multi_setup(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00117 unsigned offset, unsigned mask,
00118 unsigned mode, unsigned value);
00119
00128 int L4_CV
00129 l4vbus_gpio_multi_config_pad(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00130 unsigned offset, unsigned mask,
00131 unsigned func, unsigned value);
00132
00139 int L4_CV
00140 l4vbus_gpio_multi_get(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00141 unsigned offset, unsigned *data);
00142
00151 int L4_CV
00152 l4vbus_gpio_multi_set(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00153 unsigned offset, unsigned mask, unsigned data);
00154
00162 int L4_CV
00163 l4vbus_gpio_to_irq(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00164 unsigned pin);
00165
00167
00168 L4_END_DECLS

```

## 17.682 vbus\_i2c.h

```

00001 /*
00002 * (c) 2009 Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00003 * economic rights: Technische Universität Dresden (Germany)
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/sys/compiler.h>
00010 #include <l4/sys/types.h>
00011 #include <l4/vbus/vbus_types.h>
00012

```

```

00013 L4_BEGIN_DECLS
00014
00015 int L4_CV
00016 l4vbus_i2c_write(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00017 l4_uint16_t addr, l4_uint8_t sub_addr,
00018 l4_uint8_t *buffer, unsigned long size);
00019
00020 int L4_CV
00021 l4vbus_i2c_read(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00022 l4_uint16_t addr, l4_uint8_t sub_addr,
00023 l4_uint8_t *buffer, unsigned long *size);
00024
00025 L4_END_DECLS

```

## 17.683 vbus\_inhibitor.h

```

00001
00006 #pragma once
00007
00008 enum Vbus_inhibitor
00009 {
00010 L4VBUS_INHIBITOR_SUSPEND = 0,
00011 L4VBUS_INHIBITOR_SHUTDOWN = 1,
00012 L4VBUS_INHIBITOR_REBOOT = L4VBUS_INHIBITOR_SHUTDOWN,
00013 L4VBUS_INHIBITOR_WAKEUP = 2,
00014 L4VBUS_INHIBITOR_MAX
00015 };
00016

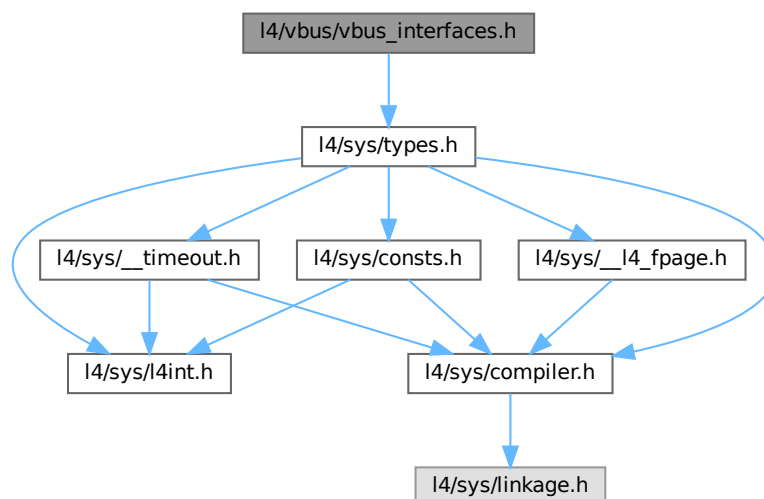
```

## 17.684 I4/vbus/vbus\_interfaces.h File Reference

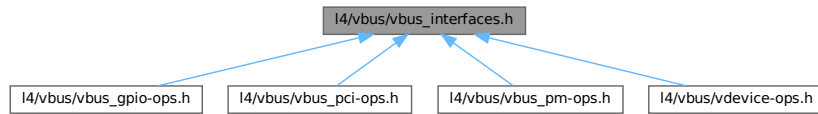
This header contains the definition of VBUS sub-interfaces and convenience functions to work with the interface IDs.

```
#include <l4/sys/types.h>
```

Include dependency graph for vbus\_interfaces.h:



This graph shows which files directly or indirectly include this file:



## Typedefs

- typedef enum [l4vbus\\_iface\\_type\\_t](#) [l4vbus\\_iface\\_type\\_t](#)  
Different sub-interfaces a vbus device may support.

## Enumerations

- enum [l4vbus\\_iface\\_type\\_t](#) {  
[L4VBUS\\_INTERFACE\\_ICU](#) = 0 , [L4VBUS\\_INTERFACE\\_GPIO](#) , [L4VBUS\\_INTERFACE\\_PCI](#) , [L4VBUS\\_INTERFACE\\_PCIDEV](#) ,  
[L4VBUS\\_INTERFACE\\_PM](#) , [L4VBUS\\_INTERFACE\\_BUS](#) , [L4VBUS\\_INTERFACE\\_GENERIC](#) = 0x20 }  
Different sub-interfaces a vbus device may support.
- enum { [L4VBUS\\_IFACE\\_SHIFT](#) = 26 }

## Functions

- unsigned [l4vbus\\_subinterface](#) (unsigned opcode)  
Return the ID of the vbus sub-interface.
- unsigned [l4vbus\\_interface\\_opcode](#) (unsigned opcode)  
Return the function opcode within the sub-interface of the vbus command.
- int [l4vbus\\_subinterface\\_supported](#) ([l4\\_uint32\\_t](#) dev\_type, [l4vbus\\_iface\\_type\\_t](#) iface\_type)  
Check if a vbus device supports a given sub-interface.

### 17.684.1 Detailed Description

This header contains the definition of VBUS sub-interfaces and convenience functions to work with the interface IDs.

Definition in file [vbus\\_interfaces.h](#).

### 17.684.2 Typedef Documentation

#### 17.684.2.1 l4vbus\_iface\_type\_t

```
typedef enum l4vbus_iface_type_t l4vbus_iface_type_t
```

Different sub-interfaces a vbus device may support.

The IPC interface of vbus devices is divided into functional groups of sub-interfaces. Every device must implement the generic interface which provides general device information. According to the type of device, additional functionality may be supported.

The sub-interface constants are first of all used to divide the function opcode space of the interface into these functional groups (see [L4VBUS\\_IFACE\\_SHIFT](#)). They also make up a bitmask that specify the type of the device, i.e. from the point of view of the client a device is defined by the kinds of sub-interfaces it supports.

## 17.684.3 Enumeration Type Documentation

### 17.684.3.1 anonymous enum

anonymous enum

#### Enumerator

|                    |                                                                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| L4VBUS_IFACE_SHIFT | Sub-interface ID shift. Divides the function opcode sent via IPC into a sub-interface ID and the actual function opcode within the sub-interface. |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|

Definition at line 48 of file [vbus\\_interfaces.h](#).

### 17.684.3.2 l4vbus\_iface\_type\_t

enum [l4vbus\\_iface\\_type\\_t](#)

Different sub-interfaces a vbus device may support.

The IPC interface of vbus devices is divided into functional groups of sub-interfaces. Every device must implement the generic interface which provides general device information. According to the type of device, additional functionality may be supported.

The sub-interface constants are first of all used to divide the function opcode space of the interface into these functional groups (see L4VBUS\_IFACE\_SHIFT). They also make up a bitmask that specify the type of the device, i.e. from the point of view of the client a device is defined by the kinds of sub-interfaces it supports.

#### Enumerator

|                          |                            |
|--------------------------|----------------------------|
| L4VBUS_INTERFACE_ICU     | Interrupt Controller.      |
| L4VBUS_INTERFACE_GPIO    | GPIO.                      |
| L4VBUS_INTERFACE_PCI     | PCI.                       |
| L4VBUS_INTERFACE_PCIDEV  | PCI Device.                |
| L4VBUS_INTERFACE_PM      | Power Management.          |
| L4VBUS_INTERFACE_BUS     | VBus.                      |
| L4VBUS_INTERFACE_GENERIC | No specific sub interface. |

Definition at line 29 of file [vbus\\_interfaces.h](#).

## 17.684.4 Function Documentation

### 17.684.4.1 l4vbus\_subinterface\_supported()

```
int l4vbus_subinterface_supported (
 l4_uint32_t dev_type,
 l4vbus_iface_type_t iface_type) [inline]
```

Check if a vbus device supports a given sub-interface.

#### Parameters

---

|                   |                                                              |
|-------------------|--------------------------------------------------------------|
| <i>dev_type</i>   | Device type as reported in <a href="#">l4vbus_device_t</a> . |
| <i>iface_type</i> | Sub-interface type to check for.                             |

## Returns

True if the device supports the sub-interface.

Definition at line 99 of file [vbus\\_interfaces.h](#).

References [L4\\_INLINE](#), and [L4VBUS\\_INTERFACE\\_GENERIC](#).

## 17.685 vbus\_interfaces.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * (c) 2014 Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00003 *
00004 * License: see LICENSE.spdx (in this directory or the directories above)
00005 */
00011 #pragma once
00012
00013 #include <l4/sys/types.h>
00014
00029 typedef enum l4vbus_iface_type_t
00030 {
00032 L4VBUS_INTERFACE_ICU = 0,
00034 L4VBUS_INTERFACE_GPIO,
00036 L4VBUS_INTERFACE_PCI,
00038 L4VBUS_INTERFACE_PCIDEV,
00040 L4VBUS_INTERFACE_PM,
00042 L4VBUS_INTERFACE_BUS,
00044 L4VBUS_INTERFACE_GENERIC = 0x20
00045 } l4vbus_iface_type_t;
00046
00047
00048 enum {
00056 L4VBUS_IFACE_SHIFT = 26
00057 };
00058
00070 L4_INLINE unsigned l4vbus_subinterface(unsigned opcode)
00071 {
00072 return opcode » L4VBUS_IFACE_SHIFT;
00073 }
00074
00086 L4_INLINE unsigned l4vbus_interface_opcode(unsigned opcode)
00087 {
00088 return opcode & ((1 « L4VBUS_IFACE_SHIFT) - 1);
00089 }
00090
00099 L4_INLINE int l4vbus_subinterface_supported(l4_uint32_t dev_type,
00100 l4vbus_iface_type_t iface_type)
00101 {
00102 if (iface_type == L4VBUS_INTERFACE_GENERIC)
00103 return 1;
00104
00105 return (dev_type & (1 « iface_type)) ? 1 : 0;
00106 }

```

## 17.686 vbus\_mcspi.h

```

00001 /*
00002 * (c) 2009 Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00003 * economic rights: Technische Universität Dresden (Germany)
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once

```

```

00008
00009 #include <l4/sys/compiler.h>
00010 #include <l4/sys/types.h>
00011 #include <l4/vbus/vbus_types.h>
00012
00013 L4_BEGIN_DECLS
00014
00015 int L4_CV
00016 l4vbus_mcspi_read(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00017 unsigned channel, l4_umword_t *value);
00018
00019 int L4_CV
00020 l4vbus_mcspi_write(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00021 unsigned channel, l4_umword_t value);
00022
00023 L4_END_DECLS

```

## 17.687 vbus\_pci

```

00001 // vi:set ft=cpp: -*- Mode: C++ -*-
00002 /*
00003 * (c) 2014 Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #pragma once
00009
00010 #include <l4/vbus/vbus>
00011 #include <l4/vbus/vbus_pci.h>
00012
00013 namespace L4vbus {
00014
00015 class Pci_host_bridge : public Device
00016 {
00017 public:
00018 int cfg_read(l4_uint32_t bus, l4_uint32_t devfn, l4_uint32_t reg,
00019 l4_uint32_t *value, l4_uint32_t width) const
00020 {
00021 return l4vbus_pci_cfg_read(bus_cap().cap(), _dev, bus,
00022 devfn, reg, value, width);
00023 }
00024
00025 int cfg_write(l4_uint32_t bus, l4_uint32_t devfn, l4_uint32_t reg,
00026 l4_uint32_t value, l4_uint32_t width) const
00027 {
00028 return l4vbus_pci_cfg_write(bus_cap().cap(), _dev, bus,
00029 devfn, reg, value, width);
00030 }
00031
00032 int irq_enable(l4_uint32_t bus, l4_uint32_t devfn, int pin,
00033 unsigned char *trigger, unsigned char *polarity) const
00034 {
00035 return l4vbus_pci_irq_enable(bus_cap().cap(), _dev, bus,
00036 devfn, pin, trigger, polarity);
00037 }
00038 };
00039
00040 class Pci_dev : public Device
00041 {
00042 public:
00043 int cfg_read(l4_uint32_t reg, l4_uint32_t *value,
00044 l4_uint32_t width) const
00045 {
00046 return l4vbus_pcidev_cfg_read(bus_cap().cap(), _dev, reg, value, width);
00047 }
00048
00049 int cfg_write(l4_uint32_t reg, l4_uint32_t value,
00050 l4_uint32_t width) const
00051 {
00052 return l4vbus_pcidev_cfg_write(bus_cap().cap(), _dev, reg, value, width);
00053 }
00054
00055 int irq_enable(unsigned char *trigger, unsigned char *polarity) const
00056 {
00057 return l4vbus_pcidev_irq_enable(bus_cap().cap(), _dev, trigger, polarity);
00058 }
00059 };

```



```

00140 }
00141
00142 };
00143
00144 }

```

## 17.688 vbus\_pci-ops.h

```

00001 /*
00002 * (c) 2014 Sarah Hoffmann <sarah.hoffmann@kernkonzept.com>
00003 *
00004 * License: see LICENSE.spdx (in this directory or the directories above)
00005 */
00006 #pragma once
00007
00008 #include <14/vbus/vbus_interfaces.h>
00009
00010 enum
00011 {
00012 L4vbus_pciroot_cfg_read = L4VBUS_INTERFACE_PCI « L4VBUS_IFACE_SHIFT,
00013 L4vbus_pciroot_cfg_write,
00014 L4vbus_pciroot_cfg_irq_enable
00015 };
00016
00017 enum
00018 {
00019 L4vbus_pcidev_cfg_read = L4VBUS_INTERFACE_PCIDEV « L4VBUS_IFACE_SHIFT,
00020 L4vbus_pcidev_cfg_write,
00021 L4vbus_pcidev_cfg_irq_enable
00022 };

```

## 17.689 vbus\_pci.h

```

00001 /*
00002 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00008 #pragma once
00009
00010 #include <14/sys/compiler.h>
00011 #include <14/vbus/vbus_types.h>
00012 #include <14/sys/types.h>
00013
00014
00015
00016
00017
00018
00019
00020
00021 L4_BEGIN_DECLS
00022
00023 int L4_CV
00030 l4vbus_pci_cfg_read(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00031 l4_uint32_t bus, l4_uint32_t devfn,
00032 l4_uint32_t reg, l4_uint32_t *value, l4_uint32_t width);
00033
00040 int L4_CV
00041 l4vbus_pci_cfg_write(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00042 l4_uint32_t bus, l4_uint32_t devfn,
00043 l4_uint32_t reg, l4_uint32_t value, l4_uint32_t width);
00044
00051 int L4_CV
00052 l4vbus_pci_irq_enable(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00053 l4_uint32_t bus, l4_uint32_t devfn,
00054 int pin, unsigned char *trigger,
00055 unsigned char *polarity);
00056
00057
00064 int L4_CV
00065 l4vbus_pcidev_cfg_read(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00066 l4_uint32_t reg, l4_uint32_t *value, l4_uint32_t width);
00067
00074 int L4_CV
00075 l4vbus_pcidev_cfg_write(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00076 l4_uint32_t reg, l4_uint32_t value, l4_uint32_t width);
00077
00084 int L4_CV
00085 l4vbus_pcidev_irq_enable(l4_cap_idx_t vbus, l4vbus_device_handle_t handle,
00086 unsigned char *trigger,

```

```

00087 unsigned char *polarity);
00088
00089
00090
00092 L4_END_DECLS

```

## 17.690 vbus\_pm-ops.h

```

00001 /*
00002 * (c) 2013 Alexander Warg <warg@os.inf.tu-dresden.de>
00003 * economic rights: Technische Universität Dresden (Germany)
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007
00008 #pragma once
00009
00010 #include "vbus_interfaces.h"
00011
00012 enum L4vbus_pm_op
00013 {
00014 L4VBUS_PM_OP_SUSPEND = L4VBUS_INTERFACE_PM << L4VBUS_IFACE_SHIFT,
00015 L4VBUS_PM_OP_RESUME,
00016 };
00017

```

## 17.691 vbus\_pm.h

```

00001 /*
00002 * (c) 2013 Alexander Warg <warg@os.inf.tu-dresden.de>
00003 * economic rights: Technische Universität Dresden (Germany)
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00007 #pragma once
00008
00009 #include <l4/sys/compiler.h>
00010 #include <l4/vbus/vbus_types.h>
00011 #include <l4/sys/types.h>
00012
00018
00019 L4_BEGIN_DECLS
00020
00027 int L4_CV
00028 l4vbus_pm_suspend(l4_cap_idx_t vbus, l4vbus_device_handle_t handle);
00029
00036 int L4_CV
00037 l4vbus_pm_resume(l4_cap_idx_t vbus, l4vbus_device_handle_t handle);
00038
00040
00041 L4_END_DECLS

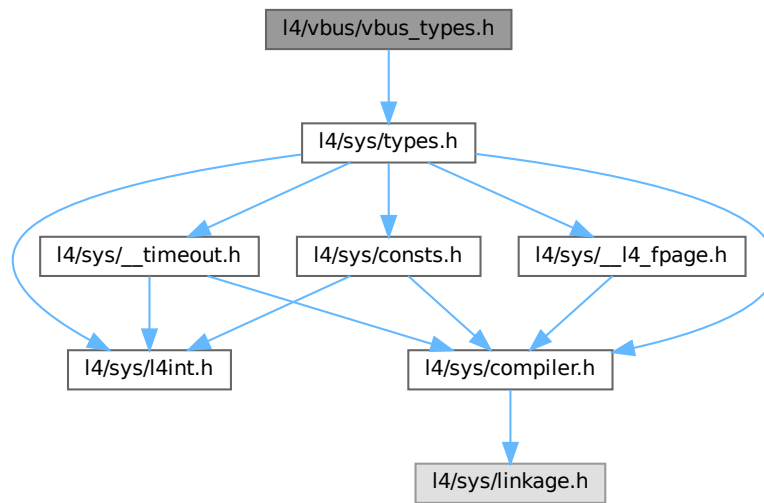
```

## 17.692 l4/vbus/vbus\_types.h File Reference

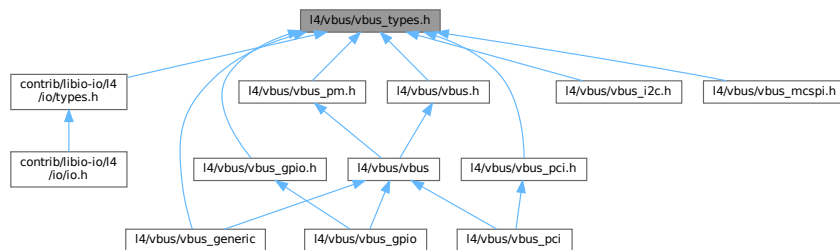
This header file contains descriptions of vbus related data types and constants.

```
#include <l4/sys/types.h>
```

Include dependency graph for vbus\_types.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [l4vbus\\_resource\\_t](#)  
*Description of a single vbus resource.*
- struct [l4vbus\\_device\\_t](#)  
*Detailed information about a vbus device.*

## Typedefs

- typedef [l4\\_mword\\_t](#) [l4vbus\\_device\\_handle\\_t](#)  
*Device handle for a device on the vbus.*
- typedef [l4\\_addr\\_t](#) [l4vbus\\_paddr\\_t](#)  
*Address of resources on the vbus.*

## Enumerations

- enum [l4vbus\\_resource\\_type\\_t](#) {  
[L4VBUS\\_RESOURCE\\_INVALID](#) = 0 , [L4VBUS\\_RESOURCE\\_IRQ](#) , [L4VBUS\\_RESOURCE\\_MEM](#) ,  
[L4VBUS\\_RESOURCE\\_PORT](#) ,  
[L4VBUS\\_RESOURCE\\_BUS](#) , [L4VBUS\\_RESOURCE\\_GPIO](#) , [L4VBUS\\_RESOURCE\\_DMA\\_DOMAIN](#) ,  
[L4VBUS\\_RESOURCE\\_MAX](#) }

*Description of vbus resource types.*

- enum [l4vbus\\_resource\\_flags\\_t](#) {  
[L4VBUS\\_RESOURCE\\_F\\_MEM\\_R](#) = 0x1 , [L4VBUS\\_RESOURCE\\_F\\_MEM\\_W](#) = 0x2 , [L4VBUS\\_RESOURCE\\_F\\_MEM\\_PREFE](#)  
= 0x10 , [L4VBUS\\_RESOURCE\\_F\\_MEM\\_CACHEABLE](#) = 0x20 ,  
[L4VBUS\\_RESOURCE\\_F\\_MEM\\_MMIO\\_READ](#) = 0x2000 , [L4VBUS\\_RESOURCE\\_F\\_MEM\\_MMIO\\_WRITE](#)  
= 0x4000 }

*Description of vbus resource flags.*

- enum [l4vbus\\_device\\_flags\\_t](#) { [L4VBUS\\_DEVICE\\_F\\_CHILDREN](#) = 0x10 }

*Flags describing device properties, see [l4vbus\\_device\\_t](#).*

## 17.692.1 Detailed Description

This header file contains descriptions of vbus related data types and constants.

Definition in file [vbus\\_types.h](#).

## 17.692.2 Enumeration Type Documentation

### 17.692.2.1 l4vbus\_device\_flags\_t

enum [l4vbus\\_device\\_flags\\_t](#)

Flags describing device properties, see [l4vbus\\_device\\_t](#).

#### Enumerator

|                                          |                           |
|------------------------------------------|---------------------------|
| <a href="#">L4VBUS_DEVICE_F_CHILDREN</a> | Device has child devices. |
|------------------------------------------|---------------------------|

Definition at line 92 of file [vbus\\_types.h](#).

### 17.692.2.2 l4vbus\_resource\_flags\_t

enum [l4vbus\\_resource\\_flags\\_t](#)

Description of vbus resource flags.

#### Enumerator

|                                         |                              |
|-----------------------------------------|------------------------------|
| <a href="#">L4VBUS_RESOURCE_F_MEM_R</a> | Memory resource is readable. |
|-----------------------------------------|------------------------------|

|                                    |                                                                                                                                                                                                                         |
|------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| L4VBUS_RESOURCE_F_MEM_W            | Memory resource is writeable.                                                                                                                                                                                           |
| L4VBUS_RESOURCE_F_MEM_PREFETCHABLE | Memory resource is prefetchable. Clients may map it buffered or non-cached.                                                                                                                                             |
| L4VBUS_RESOURCE_F_MEM_CACHEABLE    | Memory resource is cacheable. This implies that the memory resource is prefetchable. If not set, clients must not map it cached. If the resource is neither cacheable nor prefetchable, clients must map it non-cached! |
| L4VBUS_RESOURCE_F_MEM_MMIO_READ    | Reading needs to be performed using the MMIO space protocol.                                                                                                                                                            |
| L4VBUS_RESOURCE_F_MEM_MMIO_WRITE   | Writing needs to be performed using the MMIO space protocol.                                                                                                                                                            |

Definition at line 51 of file [vbus\\_types.h](#).

### 17.692.2.3 l4vbus\_resource\_type\_t

enum [l4vbus\\_resource\\_type\\_t](#)

Description of vbus resource types.

#### Enumerator

|                            |                               |
|----------------------------|-------------------------------|
| L4VBUS_RESOURCE_INVALID    | Invalid type.                 |
| L4VBUS_RESOURCE_IRQ        | Interrupt resource.           |
| L4VBUS_RESOURCE_MEM        | I/O memory resource.          |
| L4VBUS_RESOURCE_PORT       | I/O port resource (x86 only). |
| L4VBUS_RESOURCE_BUS        | Bus resource.                 |
| L4VBUS_RESOURCE_GPIO       | Gpio resource.                |
| L4VBUS_RESOURCE_DMA_DOMAIN | DMA domain.                   |
| L4VBUS_RESOURCE_MAX        | Maximum resource id.          |

Definition at line 39 of file [vbus\\_types.h](#).

## 17.693 vbus\_types.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00013 #pragma once
00014
00015 #include <l4/sys/types.h>
00016
00018 typedef l4_mword_t l4vbus_device_handle_t;
00020 typedef l4_addr_t l4vbus_paddr_t;
00021
00023 typedef struct {

```

```

00025 l4_uint16_t type;
00027 l4_uint16_t flags;
00029 l4vbus_paddr_t start;
00031 l4vbus_paddr_t end;
00033 l4vbus_device_handle_t provider;
00035 l4_uint32_t id;
00036 } l4vbus_resource_t;
00037
00039 enum l4vbus_resource_type_t {
00040 L4VBUS_RESOURCE_INVALID = 0,
00041 L4VBUS_RESOURCE_IRQ,
00042 L4VBUS_RESOURCE_MEM,
00043 L4VBUS_RESOURCE_PORT,
00044 L4VBUS_RESOURCE_BUS,
00045 L4VBUS_RESOURCE_GPIO,
00046 L4VBUS_RESOURCE_DMA_DOMAIN,
00047 L4VBUS_RESOURCE_MAX,
00048 };
00049
00051 enum l4vbus_resource_flags_t {
00053 L4VBUS_RESOURCE_F_MEM_R = 0x1,
00055 L4VBUS_RESOURCE_F_MEM_W = 0x2,
00060 L4VBUS_RESOURCE_F_MEM_PREFETCHABLE = 0x10,
00067 L4VBUS_RESOURCE_F_MEM_CACHEABLE = 0x20,
00069 L4VBUS_RESOURCE_F_MEM_MMIO_READ = 0x2000,
00071 L4VBUS_RESOURCE_F_MEM_MMIO_WRITE = 0x4000,
00072 };
00073
00074 enum l4vbus_consts_t {
00075 L4VBUS_DEV_NAME_LEN = 64,
00076 L4VBUS_MAX_DEPTH = 100,
00077 };
00078
00080 typedef struct {
00082 l4_uint32_t type;
00084 char name[L4VBUS_DEV_NAME_LEN];
00086 unsigned num_resources;
00088 unsigned flags;
00089 } l4vbus_device_t;
00090
00092 enum l4vbus_device_flags_t {
00093 L4VBUS_DEVICE_F_CHILDREN = 0x10,
00094 };

```

## 17.694 vdevice-ops.h

```

00001 /*
00002 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00004 * Torsten Frenzel <frenzel@os.inf.tu-dresden.de>
00005 * economic rights: Technische Universität Dresden (Germany)
00006 *
00007 * License: see LICENSE.spdx (in this directory or the directories above)
00008 */
00009 #pragma once
00010
00011 #include "vbus_interfaces.h"
00012
00013 enum L4vbus_vdevice_op
00014 {
00015 L4vbus_vdevice_hid = L4VBUS_INTERFACE_GENERIC « L4VBUS_IFACE_SHIFT,
00016 L4vbus_vdevice_adr,
00017 L4vbus_vdevice_get_by_hid,
00018 L4vbus_vdevice_get_next,
00019 L4vbus_vdevice_get_resource,
00020 L4vbus_vdevice_get_hid,
00021 L4vbus_vdevice_is_compatible,
00022 L4vbus_vdevice_get,
00023 };
00024
00025 enum {
00026 L4vbus_vbus_request_resource = L4VBUS_INTERFACE_BUS « L4VBUS_IFACE_SHIFT,
00027 L4vbus_vbus_release_resource,
00028 L4vbus_vbus_assign_dma_domain,
00029 };
00030
00031 enum
00032 {
00033 L4vbus_vicu_get_cap = L4VBUS_INTERFACE_ICU « L4VBUS_IFACE_SHIFT
00034 };
00035

```



```

00024 class State
00025 {
00026 public:
00027 State() {}
00028
00034 explicit State(unsigned v) : _s(v) {}
00035
00041 void add(unsigned bits) throw() { _s |= bits; }
00042
00048 void clear(unsigned bits) throw() { _s &= ~bits; }
00049
00055 void set(unsigned v) throw() { _s = v; }
00056
00057 private:
00058 __typeof__(((l4_vcpu_state_t *)0)->state) _s;
00059 };
00060
00065 class Vcpu : private l4_vcpu_state_t
00066 {
00067 public:
00071 void irq_disable() throw()
00072 { l4vcpu_irq_disable(this); }
00073
00078 unsigned irq_disable_save() throw()
00079 { return l4vcpu_irq_disable_save(this); }
00080
00081 l4_vcpu_state_t *s() { return this; }
00082 l4_vcpu_state_t const *s() const { return this; }
00083
00088 State *state() throw()
00089 {
00090 static_assert(sizeof(State) == sizeof(l4_vcpu_state_t::state),
00091 "size mismatch");
00092 return reinterpret_cast<State*>(&(l4_vcpu_state_t::state));
00093 }
00094
00099 State state() const throw()
00100 { return static_cast<State>(l4_vcpu_state_t::state); }
00101
00106 State *saved_state() throw()
00107 {
00108 static_assert(sizeof(State) == sizeof(l4_vcpu_state_t::saved_state),
00109 "size mismatch");
00110 return reinterpret_cast<State*>(&(l4_vcpu_state_t::saved_state));
00111 }
00116 State saved_state() const throw()
00117 { return static_cast<State>(l4_vcpu_state_t::saved_state); }
00118
00122 l4_uint16_t sticky_flags() const throw()
00123 { return l4_vcpu_state_t::sticky_flags; }
00124
00135 void irq_enable(l4_utcb_t *utcb, l4vcpu_event_hndl_t do_event_work_cb,
00136 l4vcpu_setup_ipc_t setup_ipc) throw()
00137 { l4vcpu_irq_enable(this, utcb, do_event_work_cb, setup_ipc); }
00138
00150 void irq_restore(unsigned s, l4_utcb_t *utcb,
00151 l4vcpu_event_hndl_t do_event_work_cb,
00152 l4vcpu_setup_ipc_t setup_ipc) throw()
00153 { l4vcpu_irq_restore(this, s, utcb, do_event_work_cb, setup_ipc); }
00154
00166 void wait_for_event(l4_utcb_t *utcb, l4vcpu_event_hndl_t do_event_work_cb,
00167 l4vcpu_setup_ipc_t setup_ipc) throw()
00168 { l4vcpu_wait_for_event(this, utcb, do_event_work_cb, setup_ipc); }
00169
00174 void task(L4::Cap<L4::Task> const task = L4::Cap<L4::Task>::Invalid) throw()
00175 { user_task = task.cap(); }
00176
00181 int is_page_fault_entry() const
00182 { return l4vcpu_is_page_fault_entry(this); }
00183
00188 int is_irq_entry() const
00189 { return l4vcpu_is_irq_entry(this); }
00190
00195 l4_vcpu_regs_t *r() throw()
00196 { return &(l4_vcpu_state_t::r); }
00197
00202 l4_vcpu_regs_t const *r() const throw()
00203 { return &(l4_vcpu_state_t::r); }
00204
00209 l4_vcpu_ipc_regs_t *i() throw()
00210 { return &(l4_vcpu_state_t::i); }
00211
00216 l4_vcpu_ipc_regs_t const *i() const throw()
00217 { return &(l4_vcpu_state_t::i); }
00218
00225 void entry_sp(l4_umword_t sp)
00226 { l4_vcpu_state_t::entry_sp = sp; }

```



```

00227
00232 void entry_ip(l4_umword_t ip)
00233 { l4_vcpu_state_t::entry_ip = ip; }
00234
00246 L4_CV static int
00247 ext_alloc(Vcpu **vcpu,
00248 l4_addr_t *ext_state,
00249 L4::Cap<L4::Task> task = L4Re::Env::env()->task(),
00250 L4::Cap<L4Re::Rm> rm = L4Re::Env::env()->rm()) throw();
00251
00259 static inline Vcpu *cast(void *x) throw()
00260 { return reinterpret_cast<Vcpu *>(x); }
00261
00269 static inline Vcpu *cast(l4_addr_t x) throw()
00270 { return reinterpret_cast<Vcpu *>(x); }
00271
00275 void print_state(const char *prefix = "") const throw()
00276 { l4vcpu_print_state(this, prefix); }
00277 };
00278
00279
00280 }

```

## 17.697 l4/sys/vcpu.h File Reference

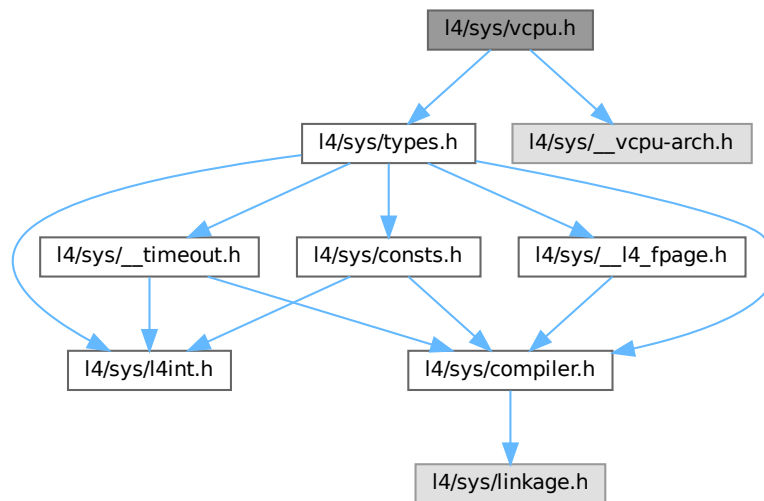
### vCPU API

```

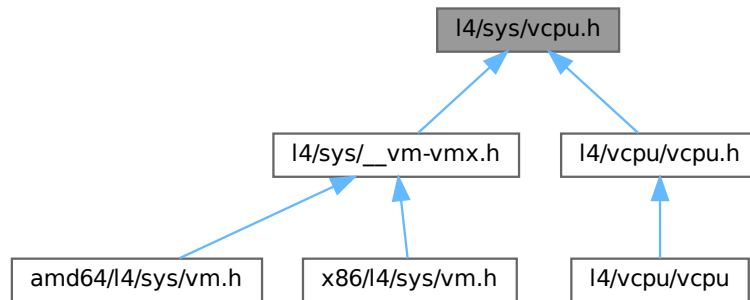
#include <l4/sys/types.h>
#include <l4/sys/__vcpu-arch.h>

```

Include dependency graph for vcpu.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [l4\\_vcpu\\_state\\_t](#)  
*State of a vCPU.*

## Typedefs

- typedef struct l4\_vcpu\_state\_t [l4\\_vcpu\\_state\\_t](#)  
*State of a vCPU.*

## Enumerations

- enum [L4\\_vcpu\\_state\\_flags](#) {  
[L4\\_VCPU\\_F\\_IRQ](#) = 0x01 , [L4\\_VCPU\\_F\\_PAGE\\_FAULTS](#) = 0x02 , [L4\\_VCPU\\_F\\_EXCEPTIONS](#) = 0x04 ,  
[L4\\_VCPU\\_F\\_USER\\_MODE](#) = 0x20 ,  
[L4\\_VCPU\\_F\\_FPU\\_ENABLED](#) = 0x80 }  
*State flags of a vCPU.*
- enum [L4\\_vcpu\\_sticky\\_flags](#) { [L4\\_VCPU\\_SF\\_IRQ\\_PENDING](#) = 0x01 }  
*Sticky flags of a vCPU.*

## Functions

- int [l4\\_vcpu\\_check\\_version](#) ([l4\\_vcpu\\_state\\_t](#) const \*vcpu) [L4\\_NOTHROW](#)  
*Check if a vCPU state has the right version.*

## 17.697.1 Detailed Description

vCPU API

Definition in file [vcpu.h](#).

## 17.697.2 Function Documentation

### 17.697.2.1 l4\_vcpu\_check\_version()

```
int l4_vcpu_check_version (
 l4_vcpu_state_t const * vcpu) [inline]
```

Check if a vCPU state has the right version.

#### Parameters

---

|                   |                                         |
|-------------------|-----------------------------------------|
| <code>vcpu</code> | A pointer to an initialized vCPU state. |
|-------------------|-----------------------------------------|

### Return values

|   |                                                                                                          |
|---|----------------------------------------------------------------------------------------------------------|
| 1 | If the vCPU state has a matching version ID for the current vCPU user-level structures.                  |
| 0 | If the vCPU state has a different (incompatible) version ID than the current vCPU user-level structures. |

Definition at line 191 of file [vcpu.h](#).

References [L4\\_NOTHROW](#), and [L4\\_VCPU\\_STATE\\_VERSION](#).

## 17.698 vcpu.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00003 * Alexander Warg <warg@os.inf.tu-dresden.de>
00004 * economic rights: Technische Universität Dresden (Germany)
00005 *
00006 * License: see LICENSE.spdx (in this directory or the directories above)
00007 */
00012 #pragma once
00013
00014 #include <l4/sys/types.h>
00015 #include <l4/sys/__vcpu-arch.h>
00016
00070
00075 typedef struct l4_vcpu_state_t
00076 {
00077 l4_umword_t version;
00080 l4_umword_t user_data[7];
00081 l4_vcpu_regs_t r;
00082 l4_vcpu_ipc_regs_t i;
00083
00084 l4_uint16_t state;
00085 l4_uint16_t saved_state;
00086 l4_uint16_t sticky_flags;
00087 l4_uint16_t _reserved;
00088
00089 l4_cap_idx_t user_task;
00090
00091 l4_umword_t entry_sp;
00092 l4_umword_t entry_ip;
00093 l4_umword_t reserved_sp;
00094 l4_vcpu_arch_state_t arch_state;
00095 } l4_vcpu_state_t;
00096
00101 enum L4_vcpu_state_flags
00102 {
00114 L4_VCPU_F_IRQ = 0x01,
00115
00129 L4_VCPU_F_PAGE_FAULTS = 0x02,
00130
00142 L4_VCPU_F_EXCEPTIONS = 0x04,
00143
00152 L4_VCPU_F_USER_MODE = 0x20,
00153
00160 L4_VCPU_F_FPU_ENABLED = 0x80,
00161 };
00162
00167 enum L4_vcpu_sticky_flags
00168 {
00171 L4_VCPU_SF_IRQ_PENDING = 0x01,
00172 };
00173
00185 L4_INLINE int
00186 l4_vcpu_check_version(l4_vcpu_state_t const *vcpu) L4_NOTHROW;
00187
00188 /* IMPLEMENTATION: -----*/
00189

```

```

00190 L4_INLINE int
00191 l4_vcpu_check_version(l4_vcpu_state_t const *vcpu) L4_NOTHROW
00192 {
00193 return vcpu->version == L4_VCPU_STATE_VERSION;
00194 }

```

## 17.699 l4/vcpu/vcpu.h File Reference

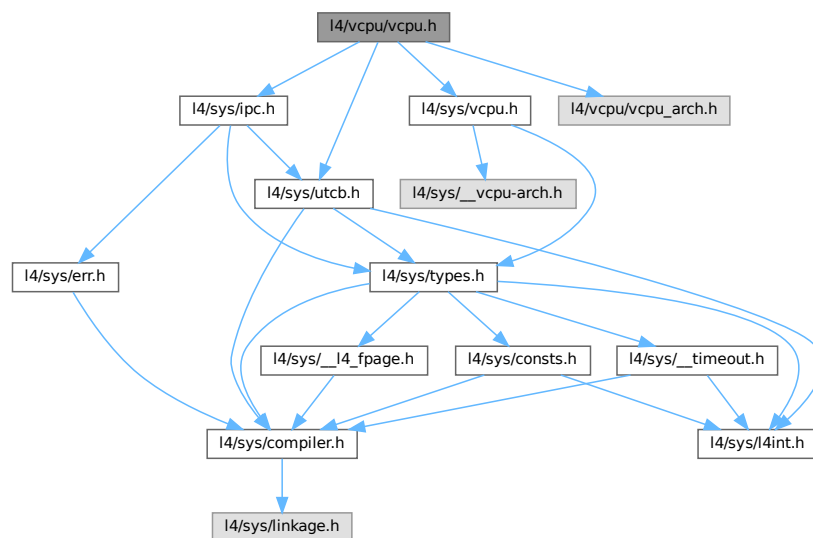
vCPU support library (C interface).

```

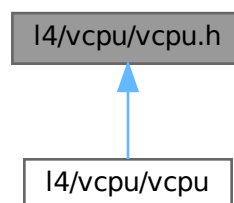
#include <l4/sys/vcpu.h>
#include <l4/sys/utcb.h>
#include <l4/sys/ipc.h>
#include <l4/vcpu/vcpu_arch.h>

```

Include dependency graph for vcpu.h:



This graph shows which files directly or indirectly include this file:



## Functions

- void [l4vcpu\\_irq\\_disable](#) ([l4\\_vcpu\\_state\\_t](#) \*vcpu) [L4\\_NOTHROW](#)  
*Disable a vCPU for event delivery.*
- unsigned [l4vcpu\\_irq\\_disable\\_save](#) ([l4\\_vcpu\\_state\\_t](#) \*vcpu) [L4\\_NOTHROW](#)  
*Disable a vCPU for event delivery and return previous state.*
- void [l4vcpu\\_irq\\_enable](#) ([l4\\_vcpu\\_state\\_t](#) \*vcpu, [l4\\_utcb\\_t](#) \*utcb, [l4vcpu\\_event\\_hndl\\_t](#) do\_event\_work\_cb, [l4vcpu\\_setup\\_ipc\\_t](#) setup\_ipc) [L4\\_NOTHROW](#)  
*Enable a vCPU for event delivery.*
- void [l4vcpu\\_irq\\_restore](#) ([l4\\_vcpu\\_state\\_t](#) \*vcpu, unsigned s, [l4\\_utcb\\_t](#) \*utcb, [l4vcpu\\_event\\_hndl\\_t](#) do\_event\_work\_cb, [l4vcpu\\_setup\\_ipc\\_t](#) setup\_ipc) [L4\\_NOTHROW](#)  
*Restore a previously saved IRQ/event state.*
- void [l4vcpu\\_wait\\_for\\_event](#) ([l4\\_vcpu\\_state\\_t](#) \*vcpu, [l4\\_utcb\\_t](#) \*utcb, [l4vcpu\\_event\\_hndl\\_t](#) do\_event\_work\_cb, [l4vcpu\\_setup\\_ipc\\_t](#) setup\_ipc) [L4\\_NOTHROW](#)  
*Wait for event.*
- void [l4vcpu\\_print\\_state](#) (const [l4\\_vcpu\\_state\\_t](#) \*vcpu, const char \*prefix) [L4\\_NOTHROW](#)  
*Print the state of a vCPU.*
- int [l4vcpu\\_is\\_irq\\_entry](#) ([l4\\_vcpu\\_state\\_t](#) const \*vcpu) [L4\\_NOTHROW](#)  
*Return whether the entry reason was an IRQ/IPC message.*
- int [l4vcpu\\_is\\_page\\_fault\\_entry](#) ([l4\\_vcpu\\_state\\_t](#) const \*vcpu) [L4\\_NOTHROW](#)  
*Return whether the entry reason was a page fault.*
- int [l4vcpu\\_ext\\_alloc](#) ([l4\\_vcpu\\_state\\_t](#) \*\*vcpu, [l4\\_addr\\_t](#) \*ext\_state, [l4\\_cap\\_idx\\_t](#) task, [l4\\_cap\\_idx\\_t](#) regmgr) [L4\\_NOTHROW](#)  
*Allocate state area for an extended vCPU.*

## 17.699.1 Detailed Description

vCPU support library (C interface).

Definition in file [vcpu.h](#).

## 17.700 vcpu.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * (c) 2010 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
00003 * economic rights: Technische Universität Dresden (Germany)
00004 *
00005 * License: see LICENSE.spdx (in this directory or the directories above)
00006 */
00011 #pragma once
00012
00013 #include <l4/sys/vcpu.h>
00014 #include <l4/sys/utcb.h>
00015
00016 __BEGIN_DECLS
00017
00026
00032
00033 typedef void (*l4vcpu_event_hndl_t) (l4_vcpu_state_t *vcpu);
00034 typedef void (*l4vcpu_setup_ipc_t) (l4_utcb_t *utcb);
00035
00042 L4_CV L4_INLINE
00043 void
00044 l4vcpu_irq_disable(l4_vcpu_state_t *vcpu) L4_NOTHROW;
00045
00054 L4_CV L4_INLINE
00055 unsigned
00056 l4vcpu_irq_disable_save(l4_vcpu_state_t *vcpu) L4_NOTHROW;

```

```

00057
00070 L4_CV L4_INLINE
00071 void
00072 l4vcpu_irq_enable(l4_vcpu_state_t *vcpu, l4_utcb_t *utcb,
00073 l4vcpu_event_hndl_t do_event_work_cb,
00074 l4vcpu_setup_ipc_t setup_ipc) L4_NOTHROW;
00075
00090 L4_CV L4_INLINE
00091 void
00092 l4vcpu_irq_restore(l4_vcpu_state_t *vcpu, unsigned s,
00093 l4_utcb_t *utcb,
00094 l4vcpu_event_hndl_t do_event_work_cb,
00095 l4vcpu_setup_ipc_t setup_ipc) L4_NOTHROW;
00096
00110 L4_CV L4_INLINE
00111 void
00112 l4vcpu_wait(l4_vcpu_state_t *vcpu, l4_utcb_t *utcb,
00113 l4_timeout_t to,
00114 l4vcpu_event_hndl_t do_event_work_cb,
00115 l4vcpu_setup_ipc_t setup_ipc) L4_NOTHROW;
00116
00130 L4_CV L4_INLINE
00131 void
00132 l4vcpu_wait_for_event(l4_vcpu_state_t *vcpu, l4_utcb_t *utcb,
00133 l4vcpu_event_hndl_t do_event_work_cb,
00134 l4vcpu_setup_ipc_t setup_ipc) L4_NOTHROW;
00135
00136
00144 L4_CV void
00145 l4vcpu_print_state(const l4_vcpu_state_t *vcpu, const char *prefix) L4_NOTHROW;
00146
00150 L4_CV void
00151 l4vcpu_print_state_arch(const l4_vcpu_state_t *vcpu, const char *prefix) L4_NOTHROW;
00152
00162 L4_CV L4_INLINE
00163 int
00164 l4vcpu_is_irq_entry(l4_vcpu_state_t const *vcpu) L4_NOTHROW;
00165
00174 L4_CV L4_INLINE
00175 int
00176 l4vcpu_is_page_fault_entry(l4_vcpu_state_t const *vcpu) L4_NOTHROW;
00177
00189 L4_CV int
00190 l4vcpu_ext_alloc(l4_vcpu_state_t **vcpu, l4_addr_t *ext_state,
00191 l4_cap_idx_t task, l4_cap_idx_t regmgr) L4_NOTHROW;
00192
00193 /* ===== */
00194 /* Implementations */
00195
00196 #include <l4/sys/ipc.h>
00197 #include <l4/vcpu/vcpu_arch.h>
00198
00199 L4_CV L4_INLINE
00200 void
00201 l4vcpu_irq_disable(l4_vcpu_state_t *vcpu) L4_NOTHROW
00202 {
00203 vcpu->state &= ~L4_VCPU_F_IRQ;
00204 l4_barrier();
00205 }
00206
00207 L4_CV L4_INLINE
00208 unsigned
00209 l4vcpu_irq_disable_save(l4_vcpu_state_t *vcpu) L4_NOTHROW
00210 {
00211 unsigned s = vcpu->state;
00212 l4vcpu_irq_disable(vcpu);
00213 return s;
00214 }
00215
00216 L4_CV L4_INLINE
00217 void
00218 l4vcpu_wait(l4_vcpu_state_t *vcpu, l4_utcb_t *utcb,
00219 l4_timeout_t to,
00220 l4vcpu_event_hndl_t do_event_work_cb,
00221 l4vcpu_setup_ipc_t setup_ipc) L4_NOTHROW
00222 {
00223 l4vcpu_irq_disable(vcpu);
00224 setup_ipc(utcb);
00225 vcpu->i.tag = l4_ipc_wait(utcb, &vcpu->i.label, to);
00226 if (L4_LIKELY(!l4_msgtag_has_error(vcpu->i.tag)))
00227 do_event_work_cb(vcpu);
00228 }
00229
00230 L4_CV L4_INLINE
00231 void
00232 l4vcpu_irq_enable(l4_vcpu_state_t *vcpu, l4_utcb_t *utcb,

```

```

00233 l4vcpu_event_hndl_t do_event_work_cb,
00234 l4vcpu_setup_ipc_t setup_ipc) L4_NOTHROW
00235 {
00236 if (!(vcpu->state & L4_VCPU_F_IRQ))
00237 {
00238 setup_ipc(utcb);
00239 l4_barrier();
00240 }
00241
00242 while (1)
00243 {
00244 vcpu->state |= L4_VCPU_F_IRQ;
00245 l4_barrier();
00246
00247 if (L4_LIKELY(!(vcpu->sticky_flags & L4_VCPU_SF_IRQ_PENDING)))
00248 break;
00249
00250 l4vcpu_wait(vcpu, utcb, L4_IPC_BOTH_TIMEOUT_0,
00251 do_event_work_cb, setup_ipc);
00252 }
00253 }
00254
00255 L4_CV L4_INLINE
00256 void
00257 l4vcpu_irq_restore(l4_vcpu_state_t *vcpu, unsigned s,
00258 l4_utcb_t *utcb,
00259 l4vcpu_event_hndl_t do_event_work_cb,
00260 l4vcpu_setup_ipc_t setup_ipc) L4_NOTHROW
00261 {
00262 if (s & L4_VCPU_F_IRQ)
00263 l4vcpu_irq_enable(vcpu, utcb, do_event_work_cb, setup_ipc);
00264 else if (vcpu->state & L4_VCPU_F_IRQ)
00265 l4vcpu_irq_disable(vcpu);
00266 }
00267
00268 L4_CV L4_INLINE
00269 void
00270 l4vcpu_wait_for_event(l4_vcpu_state_t *vcpu, l4_utcb_t *utcb,
00271 l4vcpu_event_hndl_t do_event_work_cb,
00272 l4vcpu_setup_ipc_t setup_ipc) L4_NOTHROW
00273 {
00274 l4vcpu_wait(vcpu, utcb, L4_IPC_NEVER, do_event_work_cb, setup_ipc);
00275 }
00276
00277 __END_DECLS

```

## 17.701 ipc-invoke.h

```

00001
00009 /*
00010 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00011 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00012 * Björn Döbel <doebel@os.inf.tu-dresden.de>
00013 * economic rights: Technische Universität Dresden (Germany)
00014 *
00015 * License: see LICENSE.spdx (in this directory or the directories above)
00016 */
00017
00018 #pragma once
00019
00020 /*
00021 * Some words about the sysenter entry frame: Since the sysenter instruction
00022 * automatically reloads the instruction pointer (eip) and the stack pointer
00023 * (esp) after kernel entry, we have to save both registers preliminary to
00024 * that instruction. We use ecx to store the user-level esp and save eip onto
00025 * the stack. The ecx register contains the IPC timeout and has to be saved
00026 * onto the stack, too. The ebp register is saved for compatibility reasons
00027 * with the Hazelnut kernel. Both the esp and the ss register are also pushed
00028 * onto the stack to be able to return using the "lret" instruction from the
00029 * sysexit trampoline page if Small Address Spaces are enabled.
00030 */
00031
00032 #ifdef __PIC__
00033 # define L4S_PIC_SAVE "push %%ebx; "
00034 # define L4S_PIC_RESTORE "pop %%ebx; "
00035 # define L4S_PIC_CLOBBER
00036 # define L4S_PIC_SYSCALL , [func] "m" (__l4sys_invoke_indirect)
00037 # if 1
00038 extern void (*__l4sys_invoke_indirect)(void);
00039 # define IPC_SYSENTER "# indirect sys invoke \n\t" \
00040 "call *%[func] \n\t"
00041 # else
00042 # define L4S_PIC_SYSCALL

```



```

00043 # define IPC_SYSENTER "call __l4sys_invoke_direct@plt \n\t"
00044 # endif
00045 # define IPC_SYSENTER_ASM call __l4sys_invoke_direct@plt
00046 #else
00051 #define IPC_SYSENTER "call __l4sys_invoke_direct \n\t"
00056 #define IPC_SYSENTER_ASM call __l4sys_invoke_direct
00061 # define L4S_PIC_SAVE
00066 # define L4S_PIC_RESTORE
00071 # define L4S_PIC_CLOBBER , "ebx"
00072 # define L4S_PIC_SYSCALL
00073
00074 #endif
00079 #define L4_ENTER_KERNEL L4S_PIC_SAVE "push %%ebp; " \
00080 IPC_SYSENTER
00081 " pop %%ebp; " L4S_PIC_RESTORE
00082

```

## 17.702 ipc-l42-gcc3-nopic.h

```

00001
00006 /*
00007 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
00008 * Alexander Warg <warg@os.inf.tu-dresden.de>,
00009 * Frank Mehnert <fm3@os.inf.tu-dresden.de>,
00010 * Jork Löser <jork@os.inf.tu-dresden.de>
00011 * economic rights: Technische Universität Dresden (Germany)
00012 *
00013 * License: see LICENSE.spdx (in this directory or the directories above)
00014 */
00015 #pragma once
00016
00017 #include <l4/sys/consts.h>
00018
00019 L4_INLINE l4_msgtag_t
00020 l4_ipc(l4_cap_idx_t dest, l4_utcb_t *u,
00021 l4_umword_t flags,
00022 l4_umword_t slabel,
00023 l4_msgtag_t tag,
00024 l4_umword_t *rlabel,
00025 l4_timeout_t timeout) L4_NOTHROW
00026 {
00027 l4_umword_t dummy, dummy1, dummy2;
00028
00029 (void)u;
00030
00031 __asm__ __volatile__
00032 (L4_ENTER_KERNEL
00033 :
00034 "=d" (dummy2),
00035 "=S" (slabel),
00036 "=c" (dummy1),
00037 "=D" (dummy),
00038 "=a" (tag.raw)
00039 :
00040 "S" (slabel),
00041 "c" (timeout),
00042 "a" (tag.raw),
00043 "d" (dest | flags)
00044 L4S_PIC_SYSCALL
00045 :
00046 "memory", "cc" L4S_PIC_CLOBBER
00047);
00048
00049 if (rlabel)
00050 *rlabel = slabel;
00051
00052 return tag;
00053 }

```



# Chapter 18

## Examples

### 18.1 hello/server/src/main.c

This is the famous "Hello World!" program.

This is the famous "Hello World!" program.

```
/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 * Frank Mehnert <fm3@os.inf.tu-dresden.de>,
 * Lukas Grützmacher <lg2@os.inf.tu-dresden.de>
 * economic rights: Technische Universität Dresden (Germany)
 *
 * License: see LICENSE.spdx (in this directory or the directories above)
 */
#include <stdio.h>
#include <unistd.h>

int
main(void)
{
 for (;;)
 {
 puts("Hello World!");
 sleep(1);
 }
}
```

### 18.2 examples/sys/ipc/ipc\_example.c

This example shows how two threads can exchange data using the [L4](#) IPC mechanism.

This example shows how two threads can exchange data using the [L4](#) IPC mechanism. One thread is sending an integer to the other thread which is returning the square of the integer. Both values are printed.

```
/*
 * (c) 2008-2009 Author(s)
 * economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
#include <l4/sys/ipc.h>

#include <pthread-l4.h>
#include <unistd.h>
```

```

#include <stdio.h>

static pthread_t t2;

/* Thread1 is the initiator thread, i.e. it initiates the IPC calls. In
 * other words, it takes the client role. It uses L4 IPC mechanisms to send
 * an integer value to thread2 and received a calculation result back. */
static void *thread1_fn(void *arg)
{
 l4_msgtag_t tag;
 int ipc_error;
 unsigned long value = 1;
 (void)arg;

 while (1)
 {
 printf("Sending: %ld\n", value);

 /* Store the value which we want to have squared in the first message
 * register of our UTCB. */
 l4_utcb_mr()->mr[0] = value;

 /* To an L4 IPC call, i.e. send a message to thread2 and wait for a
 * reply from thread2. The '1' in the msgtag denotes that we want to
 * transfer one word of our message registers (i.e. MR0). No timeout. */
 tag = l4_ipc_call(pthread_l4_cap(t2), l4_utcb(),
 l4_msgtag(0, 1, 0, 0), L4_IPC_NEVER);
 /* Check for IPC error, if yes, print out the IPC error code, if not,
 * print the received result. */
 ipc_error = l4_ipc_error(tag, l4_utcb());
 if (ipc_error)
 fprintf(stderr, "thread1: IPC error: %x\n", ipc_error);
 else
 printf("Received: %ld\n", l4_utcb_mr()->mr[0]);

 /* Wait some time and increment our value. */
 sleep(1);
 value++;
 }
 return NULL;
}

/* Thread2 is in the server role, i.e. it waits for requests from others and
 * sends back the calculation results. */
static void *thread2_fn(void *arg)
{
 l4_msgtag_t tag;
 l4_umword_t label;
 int ipc_error;
 (void)arg;

 /* Wait for requests from any thread. No timeout, i.e. wait forever. */
 tag = l4_ipc_wait(l4_utcb(), &label, L4_IPC_NEVER);
 while (1)
 {
 /* Check if we had any IPC failure, if yes, print the error code
 * and just wait again. */
 ipc_error = l4_ipc_error(tag, l4_utcb());
 if (ipc_error)
 {
 fprintf(stderr, "thread2: IPC error: %x\n", ipc_error);
 tag = l4_ipc_wait(l4_utcb(), &label, L4_IPC_NEVER);
 continue;
 }

 /* So, the IPC was ok, now take the value out of message register 0
 * of the UTCB and store the square of it back to it. */
 l4_utcb_mr()->mr[0] = l4_utcb_mr()->mr[0] * l4_utcb_mr()->mr[0];

 /* Send the reply and wait again for new messages.
 * The '1' in the msgtag indicated that we want to transfer 1 word in
 * the message registers (i.e. MR0) */
 tag = l4_ipc_reply_and_wait(l4_utcb(), l4_msgtag(0, 1, 0, 0),
 &label, L4_IPC_NEVER);
 }
 return NULL;
}

int main(void)
{
 // We will have two threads, one is already running the main function, the
 // other (thread2) will be created using pthread_create.

 if (pthread_create(&t2, NULL, thread2_fn, NULL))
 {
 fprintf(stderr, "Thread creation failed\n");
 return 1;
 }
}

```

```

 }

 // Just run thread1 in the main thread
 thread1_fn(NULL);
 return 0;
}

```

## 18.3 examples/sys/ipc/ipc.cfg

Sample configuration file for the IPC example.

Sample configuration file for the IPC example.

```

vim:se ft=lua:

local L4 = require("L4");

L4.default_loader:start({}, "rom/ex_ipc1");

```

## 18.4 examples/sys/start-with-exc/main.c

This example shows how to start a newly created thread with a defined set of CPU registers.

This example shows how to start a newly created thread with a defined set of CPU registers.

```

/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 * Alexander Warg <warg@os.inf.tu-dresden.de>,
 * Björn Döbel <doebel@os.inf.tu-dresden.de>,
 * Frank Mehnert <fm3@os.inf.tu-dresden.de>
 * economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
/*
 * Start a thread with an exception reply. This example does only work on
 * the x86-32 and ARM architectures.
 */

#include <l4/sys/thread.h>
#include <l4/sys/factory.h>
#include <l4/sys/ipc.h>
#include <l4/sys/utcb.h>
#include <l4/util/util.h>
#include <l4/re/env.h>
#include <l4/re/c/util/cap_alloc.h>

#include <stdlib.h>
#include <stdio.h>

/* Stack for the thread to be created. 8kB are enough. */
static char thread_stack[8 « 10];

/* The thread to be created. For illustration it will print out its
 * register set.
 */
static void L4_STICKY(thread_func(l4_umword_t *d))
{
 while (1)
 {
 printf("hey, I'm a thread\n");
 printf("got register values: %ld %ld %ld %ld %ld %ld %ld %ld\n",
 d[7], d[6], d[5], d[4], d[2], d[1], d[0]);
 l4_sleep(800);
 }
}

/* Startup trick for this example. Put all the CPU registers on the stack so
 * that the C function above can get it on the stack. */
asm(
 ".global thread \n"
 "thread: \n"

```

```

#ifdef ARCH_x86
" pusha \n"
" push %esp \n"
" call thread_func \n"
#endif
#ifdef ARCH_arm
" push {r0-r7} \n"
" mov r0, sp \n"
" bl thread_func \n"
#endif
#ifdef ARCH_arm64
" stp x0, x1, [sp, #0]! \n"
" stp x2, x3, [sp, #0]! \n"
" stp x4, x5, [sp, #0]! \n"
" stp x6, x7, [sp, #0]! \n"
" mov x0, sp \n"
" bl thread_func \n"
#endif
);
extern void thread(void);

/* Our main function */
int main(void)
{
 /* Get a capability slot for our new thread. */
 l4_cap_idx_t t1 = l4re_util_cap_alloc();
 l4_utcb_t *u = l4_utcb();
 l4_exc_regs_t *e = l4_utcb_exc_u(u);
 l4_msgtag_t tag;
 int err;

 printf("Example showing how to start a thread with an exception.\n");
 /* We do not want to implement a pager here, take the shortcut. */
 printf("Make sure to start this program with ldr-flags=eager_map\n");

 if (l4_is_invalid_cap(t1))
 return 1;

 /* Create the thread using our default factory */
 tag = l4_factory_create_thread(l4re_env()->factory, t1);
 if (l4_error(tag))
 return 1;

 /* Setup the thread by setting the pager and task. */
 l4_thread_control_start();
 l4_thread_control_pager(l4re_env()->main_thread);
 l4_thread_control_exc_handler(l4re_env()->main_thread);
 l4_thread_control_bind((l4_utcb_t *)l4re_env()->first_free_utcb,
 L4RE_THIS_TASK_CAP);
 tag = l4_thread_control_commit(t1);
 if (l4_error(tag))
 return 2;

 /* Start the thread by finally setting instruction and stack pointer */
 tag = l4_thread_ex_regs(t1,
 (l4_umword_t)thread,
 (l4_umword_t)thread_stack + sizeof(thread_stack),
 L4_THREAD_EX_REGS_TRIGGER_EXCEPTION);

 if (l4_error(tag))
 return 3;

 l4_sched_param_t sp = l4_sched_param(1, 0);
 tag = l4_scheduler_run_thread(l4re_env()->scheduler, t1, &sp);
 if (l4_error(tag))
 return 4;

 /* Receive initial exception from just started thread */
 tag = l4_ipc_receive(t1, u, L4_IPC_NEVER);
 if ((err = l4_ipc_error(tag, u)))
 {
 printf("Umm, ipc error: %x\n", err);
 return 1;
 }

 /* We expect an exception IPC */
 if (!l4_msgtag_is_exception(tag))
 {
 printf("PF?: %lx %lx (not prepared to handle this) %ld\n",
 l4_utcb_mr_u(u)->mr[0], l4_utcb_mr_u(u)->mr[1], l4_msgtag_label(tag));
 return 1;
 }

 /* Fill out the complete register set of the new thread */
 e->sp = (l4_umword_t)(thread_stack + sizeof(thread_stack));
#ifdef ARCH_x86
 e->ip = (l4_umword_t)thread;
 e->edi = 0;

```

```

 e->esi = 1;
 e->ebp = 2;
 e->ebx = 4;
 e->edx = 5;
 e->ecx = 6;
 e->eax = 7;
#endif
#ifdef ARCH_arm
 e->pc = (l4_umword_t)thread;
 e->r[0] = 0;
 e->r[1] = 1;
 e->r[2] = 2;
 e->r[3] = 3;
 e->r[4] = 4;
 e->r[5] = 5;
 e->r[6] = 6;
 e->r[7] = 7;
#endif
#ifdef ARCH_arm64
 e->pc = (l4_umword_t)thread;
 e->r[0] = 0;
 e->r[1] = 1;
 e->r[2] = 2;
 e->r[3] = 3;
 e->r[4] = 4;
 e->r[5] = 5;
 e->r[6] = 6;
 e->r[7] = 7;
#endif
/* Send a complete exception */
tag = l4_msgtag(0, L4_UTCB_EXCEPTION_REGS_SIZE, 0, 0);

/* Send reply and start the thread with the defined CPU register set */
tag = l4_ipc_send(tl, u, tag, L4_IPC_NEVER);
if ((err = l4_ipc_error(tag, u)))
 printf("Error sending IPC: %x\n", err);

/* Idle around */
while (1)
 l4_sleep(10000);

return 0;
}

```

## 18.5 examples/sys/singlestep/main.c

This example shows how a thread can be single stepped on the x86 architecture.

This example shows how a thread can be single stepped on the x86 architecture.

```

/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 * Alexander Warg <warg@os.inf.tu-dresden.de>,
 * Björn Döbel <doebel@os.inf.tu-dresden.de>
 * economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
/*
 * Single stepping example for the x86-32 architecture.
 */
#include <l4/sys/ipc.h>
#include <l4/sys/factory.h>
#include <l4/sys/thread.h>
#include <l4/sys/utcb.h>
#include <l4/sys/kdebug.h>

#include <l4/util/util.h>
#include <l4/re/env.h>
#include <l4/re/c/util/cap_alloc.h>

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

static char thread_stack[8 « 10];

static void thread_func(void)
{

```

```

while (1)
{
 unsigned long d = 0;

 /* Enable single stepping */
 asm volatile("pushf; pop %0; or $256,%0; push %0; popf\n"
 : "=r" (d) : "r" (d));

 /* Some instructions */
 asm volatile("nop");
 asm volatile("nop");
 asm volatile("nop");
 asm volatile("mov $0x12345000, %%edx" : : : "edx"); // a non-existent cap
 asm volatile("int $0x30\n");
 asm volatile("nop");
 asm volatile("nop");
 asm volatile("nop");

 /* Disabled single stepping */
 asm volatile("pushf; pop %0; and $~256,%0; push %0; popf\n"
 : "=r" (d) : "r" (d));

 /* You won't see those */
 asm volatile("nop");
 asm volatile("nop");
 asm volatile("nop");
}

int main(void)
{
 l4_msgtag_t tag;
 int ipc_stat = 0;
 l4_cap_idx_t th = l4re_util_cap_alloc();
 l4_exc_regs_t exc;
 l4_umword_t mr0, mr1;
 l4_utcb_t *u = l4_utcb();

 printf("Singlestep testing\n");

 if (l4_is_invalid_cap(th))
 return 1;

 l4_touch_rw(thread_stack, sizeof(thread_stack));
 l4_touch_ro(thread_func, 1);

 tag = l4_factory_create_thread(l4re_env()->factory, th);
 if (l4_error(tag))
 return 1;

 l4_thread_control_start();
 l4_thread_control_pager(l4re_env()->main_thread);
 l4_thread_control_exc_handler(l4re_env()->main_thread);
 l4_thread_control_bind((l4_utcb_t *)l4re_env()->first_free_utcb,
 L4RE_THIS_TASK_CAP);
 l4_thread_control_alien(1);
 tag = l4_thread_control_commit(th);
 if (l4_error(tag))
 return 2;

 tag = l4_thread_ex_regs(th, (l4_umword_t)thread_func,
 (l4_umword_t)thread_stack + sizeof(thread_stack),
 0);
 if (l4_error(tag))
 return 3;

 l4_sched_param_t sp = l4_sched_param(1, 0);
 tag = l4_scheduler_run_thread(l4re_env()->scheduler, th, &sp);
 if (l4_error(tag))
 return 4;

 /* Pager/Exception loop */
 if (l4_msgtag_has_error(tag = l4_ipc_receive(th, u, L4_IPC_NEVER)))
 {
 printf("l4_ipc_receive failed");
 return 5;
 }

 memcpy(&exc, l4_utcb_exc(), sizeof(exc));
 mr0 = l4_utcb_mr()->mr[0];
 mr1 = l4_utcb_mr()->mr[1];

 for (;;)
 {
 if (l4_msgtag_is_exception(tag))
 {
 printf("PC = %08lx Trap = %08lx Err = %08lx, SP = %08lx SC-Nr: %lx\n",
 l4_utcb_exc_pc(&exc), exc.trapno, exc.err,

```



```

 exc.sp, exc.err » 3);
 if (exc.err » 3)
 {
 if (!(exc.err & 4))
 {
 tag = l4_msgtag(L4_PROTO_ALLOW_SYSCALL,
 L4_UTCB_EXCEPTION_REGS_SIZE, 0, 0);
 if (ipc_stat)
 enter_kdebug("Should not be 1");
 }
 else
 {
 tag = l4_msgtag(L4_PROTO_NONE,
 L4_UTCB_EXCEPTION_REGS_SIZE, 0, 0);
 if (!ipc_stat)
 enter_kdebug("Should not be 0");
 }
 ipc_stat = !ipc_stat;
 }
 l4_sleep(100);
}
else
 printf("Umm, non-handled request: %ld, %08lx %08lx\n",
 l4_msgtag_label(tag), mr0, mr1);

memcpy(l4_utcb_exc(), &exc, sizeof(exc));

/* Reply and wait */
if (l4_msgtag_has_error(tag = l4_ipc_call(th, u, tag, L4_IPC_NEVER)))
{
 printf("l4_ipc_call failed\n");
 return 5;
}
memcpy(&exc, l4_utcb_exc(), sizeof(exc));
mr0 = l4_utcb_mr()->mr[0];
mr1 = l4_utcb_mr()->mr[1];
}

return 0;
}

```

## 18.6 examples/sys/aliens/main.c

This example shows how system call tracing can be done.

This example shows how system call tracing can be done.

```

/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 * Alexander Warg <warg@os.inf.tu-dresden.de>,
 * Björn Döbel <doebel@os.inf.tu-dresden.de>
 * economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
/*
 * Example to show syscall tracing.
 */
#if defined(ARCH_x86) || defined(ARCH_amd64)
// MEASURE only works on x86/amd64
// #define MEASURE
#endif

#include <l4/sys/ipc.h>
#include <l4/sys/thread.h>
#include <l4/sys/factory.h>
#include <l4/sys/utcb.h>
#include <l4/util/util.h>
#include <l4/re/env.h>
#include <l4/re/c/util/cap_alloc.h>
#include <l4/re/c/util/kumem_alloc.h>
#include <l4/sys/debugger.h>

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

/* Architecture specifics */

```

```

#if defined(ARCH_x86) || defined(ARCH_amd64)

static int
is_alien_after_call(l4_exc_regs_t const *exc)
{
 #if defined(ARCH_x86)
 return exc->err & 4;
 #else
 return exc->err == 1;
 #endif
}

static inline void
_print_exc_state(l4_exc_regs_t const *exc)
{
 printf("PC=%08lx SP=%08lx Err=%08lx Trap=%lx, %s syscall, SC-Nr: %lx\n",
 l4_utcb_exc_pc(exc), exc->sp, exc->err,
 exc->trapno, is_alien_after_call(exc) ? " after" : "before",
 exc->err >> 3);
}

#elif defined(ARCH_arm)

static int
is_alien_after_call(l4_exc_regs_t const *exc)
{ return exc->err & 0x40; } // TODO: Should change this to (1 << 16)

static inline void
_print_exc_state(l4_exc_regs_t const *exc)
{
 printf("PC=%08lx SP=%08lx ULR=%08lx CPSR=%08lx Err=%lx/%lx, %s syscall\n",
 l4_utcb_exc_pc(exc), exc->sp, exc->ulr, exc->cpsr,
 exc->err, exc->err >> 26,
 is_alien_after_call(exc) ? " after" : "before");
}

#elif defined(ARCH_arm64)

static int
is_alien_after_call(l4_exc_regs_t const *exc)
{ return exc->err & (1ul << 16); }

static inline void
_print_exc_state(l4_exc_regs_t const *exc)
{
 printf("PC=%08lx SP=%08lx PSTATE=%08lx Err=%lx/%lx, %s syscall\n",
 l4_utcb_exc_pc(exc), exc->sp, exc->pstate,
 exc->err, exc->err >> 26,
 is_alien_after_call(exc) ? " after" : "before");
}

#elif defined(ARCH_mips)

static int
is_alien_after_call(l4_exc_regs_t const *exc)
{ return 0; }

static inline void
_print_exc_state(l4_exc_regs_t const *exc)
{
 printf("PC=%08lx SP=%08lx Cause=%lx, %s syscall\n",
 l4_utcb_exc_pc(exc), exc->sp, exc->cause,
 is_alien_after_call(exc) ? " after" : "before");
}

#elif defined(ARCH_riscv)

static int
is_alien_after_call(l4_exc_regs_t const *exc)
{ return exc->cause == L4_riscv_ec_l4_alien_after_syscall; }

static inline void
_print_exc_state(l4_exc_regs_t const *exc)
{
 printf("PC=%08lx SP=%08lx Cause=%lx, %s syscall\n",
 l4_utcb_exc_pc(exc), exc->sp, exc->cause,
 is_alien_after_call(exc) ? " after" : "before");
}

#else

static int
is_alien_after_call(l4_exc_regs_t const *exc)
{ return exc->err & 1; }

static inline void
_print_exc_state(l4_exc_regs_t const *exc)

```

```

{
 printf("PC=%08lx SP=%08lx, %s syscall\n",
 l4_utcb_exc_pc(exc), exc->sp,
 is_alien_after_call(exc) ? " after" : "before");
}

#endif

/* Measurement mode specifics.
 *
 * In measurement mode the code is less verbose and uses RDTSC for alien exception
 * performance measurement.
 */
#ifdef MEASURE

#include <l4/util/rdtsc.h>

static inline void
calibrate_timer(void)
{
 l4_calibrate_tsc(l4re_kip());
}

static inline void
print_timediff(l4_cpu_time_t start)
{
 e = l4_rdtsc();
 printf("time %lld\n", l4_tsc_to_ns(e - start));
}

static inline void
alien_sleep(void)
{
 l4_sleep(0);
}

static inline void
print_exc_state(l4_exc_regs_t const *exc)
{
 if (0)
 _print_exc_state(exc);
}

#else

static inline void
calibrate_timer(void)
{
}

static inline void
print_timediff(l4_cpu_time_t start)
{
 (void)start;
}

static inline l4_cpu_time_t
l4_rdtsc(void)
{
 return 0;
}

static inline void
alien_sleep(void)
{
 l4_sleep(1000);
}

static inline void
print_exc_state(l4_exc_regs_t const *exc)
{
 _print_exc_state(exc);
}

#endif

static char alien_thread_stack[8 « 10];
static l4_cap_idx_t alien;

static void alien_thread(void)
{
 while (1)
 {
 l4_ipc_call(0x1234 « L4_CAP_SHIFT, l4_utcb(),
 l4_msgtag(0, 0, 0, 0), L4_IPC_NEVER);
 alien_sleep();
 }
}

```

```

 }
}

int main(void)
{
 l4_msgtag_t tag;
 l4_cpu_time_t s;
 l4_utcb_t *u = l4_utcb();
 l4_exc_regs_t exc;
 l4_umword_t mr0, mr1;

 printf("Alien feature testing\n");

 l4_debugger_set_object_name(l4re_env()->main_thread, "alientest");

 /* Start alien thread */
 if (l4_is_invalid_cap(alien = l4re_util_cap_alloc()))
 return 1;

 l4_touch_rw(alien_thread_stack, sizeof(alien_thread_stack));

 tag = l4_factory_create_thread(l4re_env()->factory, alien);
 if (l4_error(tag))
 return 2;

 l4_debugger_set_object_name(alien, "alienth");

 l4_addr_t kumem;
 if (l4re_util_kumem_alloc(&kumem, 0, L4RE_THIS_TASK_CAP, l4re_env()->rm))
 return 3;

 l4_thread_control_start();
 l4_thread_control_pager(l4re_env()->main_thread);
 l4_thread_control_exc_handler(l4re_env()->main_thread);
 l4_thread_control_bind((l4_utcb_t *)kumem, L4RE_THIS_TASK_CAP);
 l4_thread_control_alien(1);
 tag = l4_thread_control_commit(alien);
 if (l4_error(tag))
 return 4;

 tag = l4_thread_ex_regs(alien,
 (l4_umword_t)alien_thread,
 (l4_umword_t)alien_thread_stack + sizeof(alien_thread_stack),
 0);

 if (l4_error(tag))
 return 5;

 l4_sched_param_t sp = l4_sched_param(1, 0);
 tag = l4_scheduler_run_thread(l4re_env()->scheduler, alien, &sp);
 if (l4_error(tag))
 return 6;

 calibrate_timer();

 /* Pager/Exception loop */
 if (l4_msgtag_has_error(tag = l4_ipc_receive(alien, u, L4_IPC_NEVER)))
 {
 printf("l4_ipc_receive failed");
 return 7;
 }

 memcpy(&exc, l4_utcb_exc(), sizeof(exc));
 mr0 = l4_utcb_mr()->mr[0];
 mr1 = l4_utcb_mr()->mr[1];

 for (;;)
 {
 s = l4_rdtsc();

 if (l4_msgtag_is_exception(tag))
 {
 print_exc_state(&exc);
 tag = l4_msgtag(is_alien_after_call(&exc)
 ? 0 : L4_PROTO_ALLOW_SYSCALL,
 L4_UTCB_EXCEPTION_REGS_SIZE, 0, 0);
 }
 else
 printf("Umm, non-handled request (like PF): %lx %lx\n", mr0, mr1);

 memcpy(l4_utcb_exc(), &exc, sizeof(exc));

 /* Reply and wait */
 if (l4_msgtag_has_error(tag = l4_ipc_call(alien, u, tag, L4_IPC_NEVER)))
 {
 printf("l4_ipc_call failed\n");
 return 8;
 }
 }
}

```

```

 memcpy(&exc, l4_utcb_exc(), sizeof(exc));
 mr0 = l4_utcb_mr()->mr[0];
 mr1 = l4_utcb_mr()->mr[1];
 print_timediff(s);
 }

 return 0;
}

```

## 18.7 examples/sys/utcb-ipc/main.c

This example shows how to send IPC using the UTCB to store payload.

This example shows how to send IPC using the UTCB to store payload.

```

/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 * Alexander Warg <warg@os.inf.tu-dresden.de>,
 * Björn Döbel <doebel@os.inf.tu-dresden.de>
 * economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
#include <l4/sys/ipc.h>
#include <l4/sys/thread.h>
#include <l4/sys/factory.h>
#include <l4/sys/utcb.h>
#include <l4/sys/task.h>
#include <l4/sys/vcon.h>
#include <l4/re/env.h>
#include <l4/re/c/util/cap_alloc.h>
#include <l4/re/c/util/kumem_alloc.h>
#include <l4/util/thread.h>

#include <stdio.h>
#include <string.h>

static unsigned char stack2[8 « 10] __attribute__((aligned(8)));
static l4_cap_idx_t thread1_cap, thread2_cap;

static void vlogprintn(const char *s, int l)
{
 if (l > L4_VCON_WRITE_SIZE)
 l = L4_VCON_WRITE_SIZE;

 l4_vcon_send(L4_BASE_LOG_CAP, s, l);
}

static void vlogprint(const char *s)
{
 vlogprintn(s, strlen(s));
}

static void vlogputc(const char c)
{
 vlogprintn(&c, 1);
}

static void thread1(void)
{
 l4_msg_regs_t *mr = l4_utcb_mr();
 l4_msgtag_t tag;
 int i, j;

 printf("Thread1 up (%p)\n", l4_utcb());

 for (i = 0; i < 10; i++)
 {
 for (j = 0; j < L4_UTCB_GENERIC_DATA_SIZE; j++)
 mr->mr[j] = 'A' + (i + j) % ('~' - 'A' + 1);
 tag = l4_msgtag(0, L4_UTCB_GENERIC_DATA_SIZE, 0, 0);
 if (l4_msgtag_has_error(l4_ipc_send(thread2_cap, l4_utcb(), tag, L4_IPC_NEVER)))
 printf("IPC-send error\n");
 }

 mr->mr[0] = 1;
 if (l4_msgtag_has_error(l4_ipc_send(thread2_cap, l4_utcb(), tag, L4_IPC_NEVER)))

```

```

 printf("IPC-send error\n");

 printf("Thread1 done\n");
}

L4UTIL_THREAD_STATIC_FUNC(thread2)
{
 l4_msgtag_t tag;
 l4_msg_regs_t mr;
 unsigned i;

 // No printf() here because this would require a working pthread environment!
 vlogprint("Thread2 up\n");

 while (1)
 {
 if (l4_msgtag_has_error(tag = l4_ipc_receive(thread1_cap, l4_utcb(), L4_IPC_NEVER)))
 vlogprint("IPC receive error\n");
 memcpy(&mr, l4_utcb_mr(), sizeof(mr));
 if (mr.mr[0] == 1) // exit notification
 break;
 vlogprint("Thread2 receive: ");
 for (i = 0; i < l4_msgtag_words(tag); i++)
 vlogprintc((char)mr.mr[i]);
 vlogprint("\n");
 }

 vlogprint("Thread2 done, switching to thread1\n");
 if (l4_msgtag_has_error(l4_ipc_send(thread1_cap, l4_utcb(),
 tag, L4_IPC_NEVER)))
 vlogprint("IPC-send error\n");

 // In theory this could hit if the above IPC send operation doesn't switch
 // to the other thread.
 __builtin_trap();
}

int main(void)
{
 l4_msgtag_t tag;

 thread1_cap = l4re_env()->main_thread;
 thread2_cap = l4re_util_cap_alloc();

 if (l4_is_invalid_cap(thread2_cap))
 {
 printf("Cannot allocate thread2 capability\n");
 return 1;
 }

 tag = l4_factory_create_thread(l4re_env()->factory, thread2_cap);
 if (l4_error(tag))
 {
 printf("Cannot create thread2\n");
 return 2;
 }

 l4_addr_t kumem;
 if (l4re_util_kumem_alloc(&kumem, 0, L4RE_THIS_TASK_CAP, l4re_env()->rm))
 {
 printf("Cannot allocate UTCB for thread2\n");
 return 3;
 }

 l4_thread_control_start();
 l4_thread_control_pager(l4re_env()->rm);
 l4_thread_control_exc_handler(l4re_env()->rm);
 l4_thread_control_bind((l4_utcb_t *)kumem, L4RE_THIS_TASK_CAP);
 tag = l4_thread_control_commit(thread2_cap);
 if (l4_error(tag))
 {
 printf("Cannot set thread2 thread parameters\n");
 return 4;
 }

 tag = l4_thread_ex_regs(thread2_cap,
 (l4_umword_t)thread2,
 (l4_umword_t)(stack2 + sizeof(stack2)), 0);

 if (l4_error(tag))
 {
 printf("Cannot set thread2 IP/SP\n");
 return 5;
 }

 l4_sched_param_t sp = l4_sched_param(1, 0);
 tag = l4_scheduler_run_thread(l4re_env()->scheduler, thread2_cap, &sp);
 if (l4_error(tag))

```

```

 {
 printf("Cannot start thread2\n");
 return 6;
 }

thread1();

if (l4_msgtag_has_error(l4_ipc_receive(thread2_cap, l4_utcb(),
 L4_IPC_NEVER)))
 printf("IPC-receive error\n");

l4_task_unmap(L4RE_THIS_TASK_CAP,
 l4_obj_fpage(thread2_cap, 0, L4_FPAGE_RWX),
 L4_FP_ALL_SPACES);

printf("Terminated thread2. Terminating.\n");
return 0;
}

```

## 18.8 examples/sys/isr/main.c

Example of an interrupt service routine.

Example of an interrupt service routine.

```

/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 * Alexander Warg <warg@os.inf.tu-dresden.de>,
 * Björn Döbel <doebel@os.inf.tu-dresden.de>
 * economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
/*
 * This example shall show how to connect to an interrupt, receive interrupt
 * events and detach again. As the interrupt source we'll use the virtual
 * key interrupt -- pin 0 of the log capability.
 */

#include <l4re/c/util/cap_alloc.h>
#include <l4re/c/namespace.h>
#include <l4/sys/factory.h>
#include <l4/sys/icu.h>
#include <l4/sys/irq.h>
#include <l4/sys/vcon.h>
#include <l4/sys/utcb.h>

#include <stdio.h>

int main(void)
{
 int const irqno = 0;
 l4_cap_idx_t irqcap, icucap;
 l4_vcon_attr_t attr;
 long err;

 icucap = l4re_env()->log;

 /* Get a free capability slot for the ICU capability. */
 if (l4_is_invalid_cap(icucap))
 {
 printf("Did not find the Vlog ICU.\n");
 return 1;
 }

 /* Get another free capability slot for the corresponding IRQ object. */
 if (l4_is_invalid_cap(irqcap = l4re_util_cap_alloc()))
 {
 printf("Cannot allocate capability slot.\n");
 return 1;
 }

 /* Create IRQ object. */
 if ((err = l4_error(l4_factory_create_irq(l4re_env()->factory, irqcap))))
 {
 printf("Could not create IRQ object: %ld (%s).\n", err, l4sys_errtostr(err));
 return 1;
 }

 /*

```

```

 * Bind the recently allocated IRQ object to the IRQ number irqno as provided
 * by the ICU.
 */
if ((err = l4_error(l4_icu_bind(icucap, irqno, irqcap)))
{
 printf("Could not bind IRQ%d to ICU: %ld (%s).\n", irqno, err, l4sys_errtostr(err));
 return 1;
}

if ((err = l4_error(l4_vcon_get_attr(icucap, &attr)))
{
 printf("Could not get Vcon attributes: %ld (%s).\n", err, l4sys_errtostr(err));
 return 1;
}

/* Disable echo at Vcon console. */
attr.l_flags &= ~L4_VCON_ECHO;

if ((err = l4_error(l4_vcon_set_attr(icucap, &attr)))
{
 printf("Could not set Vcon attributes: %ld (%s).\n", err, l4sys_errtostr(err));
 return 1;
}

printf("Vcon echo disabled.\n");

/* Bind ourselves to the IRQ. Define the IPC label which is sent if an IRQ
 * IPC arrives. */
if ((err = l4_error(l4_rcv_ep_bind_thread(irqcap, l4re_env()->main_thread, 0x1234)))
{
 printf("Could not bind to IRQ%d: %ld (%s).\n", irqno, err, l4sys_errtostr(err));
 return 1;
}

printf("Attached to key IRQ %d.\nPress keys now, Shift-Q to exit.\n", irqno);

/* IRQ receive loop. */
while (1)
{
 /* Wait for the interrupt to happen. If we received an IRQ, the label
 * return code is set to 0. If we didn't receive an IRQ, the error flag
 * in the message tag is set and l4_error() reads the IPC error code from
 * the UTCB. */
 l4_umword_t label;
 if ((err = l4_error(l4_irq_wait(irqcap, &label, L4_IPC_NEVER)))
 printf("Could not receive IRQ: %ld (%s).\n", err, l4sys_errtostr(err));
 else
 {
 char buf[128];
 int n;

 if (label != 0x1234)
 {
 printf("Unexpected label %0lx -- ignoring interrupt.\n", label);
 continue;
 }

 /* Process the interrupt -- may do a 'break' */
 printf("Got IRQ with expected label 0x%lx.\n", label);
 n = l4_vcon_read(icucap, buf, sizeof(buf));
 if (n < 0)
 printf("Could not read from Vcon interface: %d (%s).\n", n, l4sys_errtostr(n));
 else
 {
 unsigned i;
 int terminate = 0;
 for (i = 0; i < (unsigned)n && i < sizeof(buf); ++i)
 {
 int c = (unsigned char)buf[i];
 if (c >= 32 && c < 128) // Filter UTF-8 encodings.
 printf("Got key '%c'.\n", c);
 else
 printf("Got keycode %d.\n", c);
 if (buf[i] == 'Q')
 terminate = 1;
 }

 if (terminate)
 break;
 }
 }
}

/* We're done, detach from the interrupt. */
if ((err = l4_error(l4_irq_detach(irqcap)))

```



```

 printf("Could not detach from IRQ: %ld (%s).\n", err, l4sys_strerror(err));

 printf("Application terminated.\n");
 return 0;
}

```

## 18.9 examples/clntsrv/src/server.cc

Client/Server example using C++ infrastructure – Server implementation.

Client/Server example using C++ infrastructure – Server implementation.

```

/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 * Alexander Warg <warg@os.inf.tu-dresden.de>
 * economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
#include <stdio.h>
#include <l4/re/env>
#include <l4/re/util/cap_alloc>
#include <l4/re/util/object_registry>
#include <l4/re/util/br_manager>
#include <l4/sys/cxx/ipc_epiface>

#include "shared.h"

static L4Re::Util::Registry_server<> server;

class Calculation_server : public L4::Epiface_t<Calculation_server, Calc>
{
public:
 int op_sub(Calc::Rights, l4_uint32_t a, l4_uint32_t b, l4_uint32_t &res)
 {
 res = a - b;
 return 0;
 }

 int op_neg(Calc::Rights, l4_uint32_t a, l4_uint32_t &res)
 {
 res = -a;
 return 0;
 }
};

int
main()
{
 static Calculation_server calc;

 // Register calculation server
 if (!server.registry()->register_obj(&calc, "calc_server").is_valid())
 {
 printf("Could not register my service, is there a 'calc_server' in the caps table?\n");
 return 1;
 }

 printf("Welcome to the calculation server!\n"
 "I can do subtractions and negations.\n");

 // Wait for client requests
 server.loop();

 return 0;
}

```

## 18.10 examples/clntsrv/src/client.cc

Client/Server example using C++ infrastructure – Client implementation.

Client/Server example using C++ infrastructure – Client implementation.

```

/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 * Alexander Warg <warg@os.inf.tu-dresden.de>
 * economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
#include <l4/sys/err.h>
#include <l4/sys/types.h>
#include <l4/re/env>
#include <l4/re/util/cap_alloc>

#include <stdio.h>
#include "shared.h"

int
main()
{
 L4::Cap<Calc> server = L4Re::Env::env()->get_cap<Calc>("calc_server");
 if (!server.is_valid())
 {
 printf("Could not get server capability!\n");
 return 1;
 }

 l4_uint32_t val1 = 8;
 l4_uint32_t val2 = 5;

 printf("Asking for %d - %d\n", val1, val2);
 if (server->sub(val1, val2, &val1))
 {
 printf("Error talking to server\n");
 return 1;
 }
 printf("Result of subtract call: %d\n", val1);

 printf("Asking for -%d\n", val1);
 if (server->neg(val1, &val1))
 {
 printf("Error talking to server\n");
 return 1;
 }
 printf("Result of negate call: %d\n", val1);

 return 0;
}

```

## 18.11 examples/clntsrv/src/shared.h

Client/Server example using C++ infrastructure – Shared header file.

Client/Server example using C++ infrastructure – Shared header file.

```

/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
 * economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */

#pragma once

#include <l4/sys/capability>
#include <l4/sys/cxx/ipc_iface>

struct Calc : L4::Kobject_t<Calc, L4::Kobject, 0x44>
{
 L4_INLINE_RPC(int, sub, (l4_uint32_t a, l4_uint32_t b, l4_uint32_t *res));
 L4_INLINE_RPC(int, neg, (l4_uint32_t a, l4_uint32_t *res));
 typedef L4::Typeid::Rpcs<sub_t, neg_t> Rpcs;
};

```

## 18.12 examples/clntsrv/configs/clntsrv.cfg

Sample configuration file for the client/server example.

Sample configuration file for the client/server example.

```
-- vim:set ft=lua:

-- Include L4 functionality
local L4 = require("L4");

-- Some shortcut for less typing
local ld = L4.default_loader;

-- Channel for the two programs to talk to each other.
local calc_server = ld:new_channel();

-- The server program, getting the channel in server mode.
ld:start({ caps = { calc_server = calc_server:svr() },
 log = { "server", "blue" } },
 "rom/ex_clntsrv-server");

-- The client program, getting the 'calc_server' channel to be able to talk
-- to the server. The client will be started with a green log output.
ld:start({ caps = { calc_server = calc_server },
 log = { "client", "green" } },
 "rom/ex_clntsrv-client");
```

## 18.13 examples/libs/l4re/c/ma+rm.c

Coarse grained memory allocation, in C.

Coarse grained memory allocation, in C.

```
/*
 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
 * economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */

#include <l4re/c/mem_alloc.h>
#include <l4re/c/rm.h>
#include <l4re/c/util/cap_alloc.h>
#include <l4re/sys/err.h>
#include <stdio.h>
#include <string.h>

static int allocate_mem(unsigned long size_in_bytes, unsigned long flags,
 void **virt_addr)
{
 int r;
 l4re_ds_t ds;

 /* Allocate a free capability index for our data space */
 ds = l4re_util_cap_alloc();
 if (l4_is_invalid_cap(ds))
 return -L4_ENOMEM;

 size_in_bytes = l4_trunc_page(size_in_bytes);

 /* Allocate memory via a dataspace */
 if ((r = l4re_ma_alloc(size_in_bytes, ds, flags))
 return r;

 /* Make the dataspace visible in our address space */
 *virt_addr = 0;
 if ((r = l4re_rm_attach(virt_addr, size_in_bytes,
 L4RE_RM_F_SEARCH_ADDR | L4RE_RM_F_RWX, ds, 0,
 flags & L4RE_MA_SUPER_PAGES
 ? L4_SUPERPAGESHIFT : L4_PAGESHIFT)))
 {
 /* Free dataspace again */
 l4re_util_cap_free_um(ds);
 return r;
 }
}
```

```

 }

 /* Done, virtual address is in virt_addr */
 return 0;
}

static int free_mem(void *virt_addr)
{
 int r;
 l4re_ds_t ds;

 /* Detach memory from our address space */
 if ((r = l4re_rm_detach_ds(virt_addr, &ds)))
 return r;

 /* Free memory at our memory allocator */
 l4re_util_cap_free_um(ds);

 /* All went ok */
 return 0;
}

int main(void)
{
 void *virt;

 /* Allocate memory: 16k Bytes (usually) */
 if (allocate_mem(4 * L4_PAGESIZE, 0, &virt))
 return 1;

 printf("Allocated memory.\n");

 /* Do something with the memory */
 memset(virt, 0x12, 4 * L4_PAGESIZE);

 printf("Touched memory.\n");

 /* Free memory */
 if (free_mem(virt))
 return 2;

 printf("Freed and done. Bye.\n");

 return 0;
}

```

## 18.14 examples/libs/l4re/c++/mem\_alloc/ma+rm.cc

Coarse grained memory allocation, in C++.

Coarse grained memory allocation, in C++.

```

/*
 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
 * economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */

#include <l4/re/mem_alloc>
#include <l4/re/rm>
#include <l4/re/env>
#include <l4/re/dataspace>
#include <l4/re/util/cap_alloc>
#include <l4/sys/err.h>
#include <cstdio>
#include <cstring>

static int allocate_mem(unsigned long size_in_bytes, unsigned long flags,
 void **virt_addr)
{
 int r;
 L4::Cap<L4Re::Dataspace> d;

 /* Allocate a free capability index for our data space */
 d = L4Re::Util::cap_alloc.alloc<L4Re::Dataspace>();
 if (!d.is_valid())

```

```

 return -L4_ENOMEM;

size_in_bytes = l4_trunc_page(size_in_bytes);

/* Allocate memory via a dataspace */
if ((r = L4Re::Env::env()->mem_alloc()->alloc(size_in_bytes, d, flags)))
 return r;

/* Make the dataspace visible in our address space */
*virt_addr = 0;
if ((r = L4Re::Env::env()->rm()->attach(virt_addr, size_in_bytes,
 L4Re::Rm::F::Search_addr | L4Re::Rm::F::RW,
 L4::Ipc::make_cap_rw(d), 0,
 flags & L4Re::Mem_alloc::Super_pages
 ? L4_SUPERPAGESHIFT : L4_PAGESHIFT)))
 return r;

/* Done, virtual address is in virt_addr */
return 0;
}

static int free_mem(void *virt_addr)
{
 int r;
 L4::Cap<L4Re::Dataspace> ds;

 /* Detach memory from our address space */
 if ((r = L4Re::Env::env()->rm()->detach(virt_addr, &ds)))
 return r;

 /* Release and return capability slot to allocator */
 L4Re::Util::cap_alloc.free(ds, L4Re::Env::env()->task().cap());

 /* All went ok */
 return 0;
}

int main(void)
{
 void *virt;

 /* Allocate memory: 16k Bytes (usually) */
 if (allocate_mem(4 * L4_PAGESIZE, 0, &virt))
 return 1;

 printf("Allocated memory.\n");

 /* Do something with the memory */
 memset(virt, 0x12, 4 * L4_PAGESIZE);

 printf("Touched memory.\n");

 /* Free memory */
 if (free_mem(virt))
 return 2;

 printf("Freed and done. Bye.\n");

 return 0;
}

```

## 18.15 examples/libs/l4re/c++/shared\_ds/ds\_clnt.cc

Sharing memory between applications, client side.

Sharing memory between applications, client side.

```

/*
 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 * Alexander Warg <warg@os.inf.tu-dresden.de>
 * economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */

#include <l4/re/util/cap_alloc> // L4::Cap
#include <l4/re/dataspace> // L4Re::Dataspace
#include <l4/re/rm> // L4::Rm
#include <l4/re/env> // L4::Env

```

```

#include <l4/sys/cache.h>

#include <cstring>
#include <stdio>
#include <unistd.h>

#include "interface.h"

int main()
{
 /*
 * Try to get server interface cap.
 */

 L4::Cap<My_interface> svr = L4Re::Env::env()->get_cap<My_interface>("shm");
 if (!svr.is_valid())
 {
 printf("Could not get the server capability\n");
 return 1;
 }

 /*
 * Alloc data space cap slot
 */
 L4::Cap<L4Re::Dataspace> ds = L4Re::Util::cap_alloc.alloc<L4Re::Dataspace>();
 if (!ds.is_valid())
 {
 printf("Could not get capability slot!\n");
 return 1;
 }

 /*
 * Alloc server notifier IRQ cap slot
 */
 L4::Cap<L4::Irq> irq = L4Re::Util::cap_alloc.alloc<L4::Irq>();
 if (!irq.is_valid())
 {
 printf("Could not get capability slot!\n");
 return 1;
 }

 /*
 * Request shared data-space cap.
 */
 if (svr->get_shared_buffer(ds, irq))
 {
 printf("Could not get shared memory dataspace!\n");
 return 1;
 }

 /*
 * Attach to arbitrary region
 */
 char *addr = 0;
 int err = L4Re::Env::env()->rm()->attach(&addr, ds->size(),
 L4Re::Rm::F::Search_addr | L4Re::Rm::F::RW,
 L4::Ipc::make_cap_rw(ds));

 if (err < 0)
 {
 printf("Error attaching data space: %s\n", l4sys_errtostr(err));
 return 1;
 }

 printf("Content: %s\n", addr);

 // wait a bit for the demo effect
 printf("Sleeping a bit...\n");
 sleep(1);

 /*
 * Fill in new stuff
 */
 memset(addr, 0, ds->size());
 char const * msg = "Hello from client, too!";
 printf("Setting new content in shared memory\n");
 snprintf(addr, strlen(msg)+1, msg);
 l4_cache_clean_data((unsigned long)addr,
 (unsigned long)addr + strlen(msg) + 1);

 // notify the server
 irq->trigger();

 /*
 * Detach region containing addr, result should be Detached_ds (other results
 * only apply if we split regions etc.).
 */
 err = L4Re::Env::env()->rm()->detach(addr, 0);

```

```

 if (err)
 printf("Failed to detach region\n");

 /* Free objects and capabilities, just for completeness. */
 L4Re::Util::cap_alloc.free(ds, L4Re::This_task);
 L4Re::Util::cap_alloc.free(irq, L4Re::This_task);

 return 0;
}

```

## 18.16 examples/libs/l4re/c++/shared\_ds/ds\_srv.cc

Sharing memory between applications, server/creator side.

Sharing memory between applications, server/creator side.

```

/*
 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 * Alexander Warg <warg@os.inf.tu-dresden.de>
 * economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */

#include <l4/re/env>
#include <l4/re/error_helper>
#include <l4/re/namespace>
#include <l4/re/util/cap_alloc>
#include <l4/re/util/object_registry>
#include <l4/re/dataspace>
#include <l4/cxx/ipc_server>
#include <l4/util/util.h>

#include <l4/sys/typeinfo_svr>

#include <cstring>
#include <cstdio>
#include <unistd.h>
#include <pthread.h>
#include <pthread-l4.h>
#include <thread>

#include "interface.h"

class My_server_obj : public L4::Server_object_t<L4::Kobject>
{
private:
 L4::Cap<L4Re::Dataspace> _shm;
 L4::Cap<L4::Irq> _irq;

public:
 explicit My_server_obj(L4::Cap<L4Re::Dataspace> shm, L4::Cap<L4::Irq> irq)
 : _shm(shm), _irq(irq)
 {}

 int dispatch(l4_umword_t obj, L4::Ipc::Iostream &ios);
};

int My_server_obj::dispatch(l4_umword_t obj, L4::Ipc::Iostream &ios)
{
 // we don't care about the original object reference, however
 // we could read out the access rights from the lowest 2 bits
 (void) obj;

 l4_msgtag_t t;
 ios » t; // extract the tag

 switch (t.label())
 {
 case L4::Meta::Protocol:
 // handle the meta protocol requests, implementing the
 // runtime dynamic type system for L4 objects.
 return L4::Util::handle_meta_request<My_interface>(ios);
 case 0:
 // since we have just one operation we have no opcode dispatch,
 // and just return the data-space and the notifier IRQ capabilities
 ios « _shm « _irq;
 return 0;
 }
}

```

```

 default:
 // every other protocol is not supported.
 return -L4_EBADPROTO;
 }
 }

class Shm_observer : public L4::Irq_handler_object
{
private:
 char *_shm;

public:
 explicit Shm_observer(char *_shm)
 : _shm(shm)
 {}

 int dispatch(l4_umword_t obj, L4::Ipc::Iostream &ios);
};

int Shm_observer::dispatch(l4_umword_t obj, L4::Ipc::Iostream &ios)
{
 // We don't care about the original object reference, however
 // we could read out the access rights from the lowest 2 bits
 (void)obj;

 // Since we end up here in this function, we got a 'message' from the IRQ
 // that is bound to us. The 'ios' stream won't contain any valuable info.
 (void)ios;

 printf("Client sent us: %s\n", _shm);

 return 0;
}

enum
{
 DS_SIZE = 4 « 12,
};

static char *get_ds(L4::Cap<L4Re::Dataspace> *_ds)
{
 *_ds = L4Re::Util::cap_alloc.alloc<L4Re::Dataspace>();
 if (!(*_ds).is_valid())
 {
 printf("Dataspace allocation failed.\n");
 return 0;
 }

 int err = L4Re::Env::env()->mem_alloc()->alloc(DS_SIZE, *_ds, 0);
 if (err < 0)
 {
 printf("mem_alloc->alloc() failed.\n");
 L4Re::Util::cap_alloc.free(*_ds);
 return 0;
 }

 /*
 * Attach DS to local address space
 */
 char *_addr = 0;
 err = L4Re::Env::env()->rm()->attach(&_addr, (*_ds)->size(),
 L4Re::Rm::F::Search_addr | L4Re::Rm::F::RW,
 L4::Ipc::make_cap_rw(*_ds));

 if (err < 0)
 {
 printf("Error attaching data space: %s\n", l4sys_errtostr(err));
 L4Re::Util::cap_alloc.free(*_ds);
 return 0;
 }

 /*
 * Success! Write something to DS.
 */
 printf("Attached DS\n");
 static char const * const msg = "[DS] Hello from server!";
 snprintf(_addr, strlen(msg) + 1, msg);

 return _addr;
}

static void *server_thread(void *)
{
 L4::Cap<L4::Thread> l4_thread = Pthread::L4::cap(pthread_self());
 L4Re::Util::Registry_server<> server(l4_thread, L4Re::Env::env()->factory());

 L4::Cap<L4Re::Dataspace> ds;

```



```

char *addr;

if (!(addr = get_ds(&ds)))
 return nullptr;

// First the IRQ handler, because we need it in the My_server_obj object
Shm_observer observer(addr);

// Registering the observer as an IRQ handler, this allocates an
// IRQ object using the factory of our server.
L4::Cap<L4::Irq> irq = server.registry()->register_irq_obj(&observer);

// Now the initial server object shared with the client via our parent.
// it provides the data-space and the IRQ capabilities to a client.
My_server_obj server_obj(ds, irq);

// Registering the server object to the capability 'shm' in our the L4Re::Env.
// This capability must be provided by the parent. (see the shared_ds.lua)
server.registry()->register_obj(&server_obj, "shm");

// Run our server loop.
server.loop();
}

int main()
{
 pthread_attr_t pattr;

 if (pthread_attr_init(&pattr))
 L4Re::throw_error(-L4_ENOMEM, "Initialize pthread attributes");

 pthread_t thr;
 L4Re::chksys(pthread_create(&thr, &pattr, server_thread, nullptr),
 "Create server thread");
 L4Re::chksys(pthread_attr_destroy(&pattr), "Destroy pthread attributes");

 l4_sleep_forever();

 return 0;
}

```

## 18.17 examples/libs/l4re/c++/shared\_ds/shared\_ds.cfg

Sharing memory between applications, configuration file.

Sharing memory between applications, configuration file.

```

-- Include L4 functionality
local L4 = require("L4");

-- Create a channel from the client to the server
local channel = L4.default_loader:new_channel();

-- Start the server, giving the channel with full server rights.
-- The server will have a yellow log output.
L4.default_loader:start(
{
 caps = { shm = channel:svr() },
 log = { "server", "yellow" }
},
"rom/ex_l4re_ds_srv"
);

-- Start the client, giving it the channel with read only rights. The
-- log output will be green.
L4.default_loader:start(
{
 caps = { shm = channel },
 log = { "client", "green" },
 l4re_dbg = L4.Dbg.Warn
},
"rom/ex_l4re_ds_clnt"
);

```

## 18.18 examples/libs/l4re/streammap/server.cc

Client/Server example showing how to map a page to another task – Server implementation.

Client/Server example showing how to map a page to another task – Server implementation. Note that there's also a shared memory library that supplies this functionality in more convenient way.

```

/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 * Alexander Warg <warg@os.inf.tu-dresden.de>
 * economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
#include <stdio.h>
#include <l4/re/env>
#include <l4/re/util/cap_alloc>
#include <l4/re/util/object_registry>
#include <l4/cxx/ipc_server>

#include "shared.h"

static char page_to_map[L4_PAGESIZE] __attribute__((aligned(L4_PAGESIZE)));

static L4Re::Util::Registry_server<> server;

class Smap_server : public L4::Server_object_t<Mapper>
{
public:
 int dispatch(l4_umword_t obj, L4::Ipc::Iostream &ios);
};

int
Smap_server::dispatch(l4_umword_t, L4::Ipc::Iostream &ios)
{
 l4_msgtag_t t;
 ios » t;

 // We're only talking the Map_example protocol
 if (t.label() != Mapper::Protocol)
 return -L4_EBADPROTO;

 L4::Opcode opcode;
 ios » opcode;

 switch (opcode)
 {
 case Mapper::Do_map:
 l4_addr_t snd_base;
 ios » snd_base;
 // put something into the page to read it out at the other side
 snprintf(page_to_map, sizeof(page_to_map), "Hello from the server!");
 printf("Sending to client\n");
 // send page
 ios « L4::Ipc::Snd_fpage::mem((l4_addr_t)page_to_map, L4_PAGESHIFT,
 L4_FPAGE_RO, snd_base);

 return L4_EOK;
 default:
 return -L4_ENOSYS;
 }
}

int
main()
{
 static Smap_server smap;

 // Register server
 if (!server.registry()->register_obj(&smap, "smap").is_valid())
 {
 printf("Could not register my service, read-only namespace?\n");
 return 1;
 }

 printf("Welcome to the memory map example server!\n");

 // Wait for client requests
 server.loop();

 return 0;
}

```

## 18.19 examples/libs/l4re/streammap/client.cc

Client/Server example showing how to map a page to another task – Client implementation.

Client/Server example showing how to map a page to another task – Client implementation. Note that there's also a shared memory library that supplies this functionality in more convenient way.

```

/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 * Alexander Warg <warg@os.inf.tu-dresden.de>
 * economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
#include <l4/sys/err.h>
#include <l4/sys/types.h>
#include <l4/re/env>
#include <l4/re/util/cap_alloc>
#include <l4/cxx/ipc_stream>

#include <stdio.h>

#include "shared.h"

static int
func_smap_call(L4::Cap<void> const &server)
{
 L4::Ipc::Iostream s(l4_utcb());
 l4_addr_t addr = 0;
 int err;

 if ((err = L4Re::Env::env()->rm()->reserve_area(&addr, L4_PAGESIZE,
 L4Re::Rm::F::Search_addr)))
 {
 printf("The reservation of one page within our virtual memory failed with %d\n", err);
 return 1;
 }

 s « L4::Opcode(Mapper::Do_map)
 « (l4_addr_t)addr;
 s « L4::Ipc::Rcv_fpage::mem((l4_addr_t)addr, L4_PAGESHIFT, 0);
 int r = l4_error(s.call(server.cap(), Mapper::Protocol));
 if (r)
 return r; // failure

 printf("String sent by server: %s\n", (char *)addr);

 return 0; // ok
}

int
main()
{
 L4::Cap<void> server = L4Re::Env::env()->get_cap<void>("smap");
 if (!server.is_valid())
 {
 printf("Could not get capability slot!\n");
 return 1;
 }

 printf("Asking for page from server\n");

 if (func_smap_call(server))
 {
 printf("Error talking to server\n");
 return 1;
 }
 printf("It worked!\n");

 L4Re::Util::cap_alloc.free(server, L4Re::This_task);

 return 0;
}

```

## 18.20 examples/libs/l4re/streammap/streammap.cfg

Sample configuration file for the client/server map example.

Sample configuration file for the client/server map example.

```
-- vim:set ft=lua:

-- Include L4 functionality
local L4 = require("L4");

-- Channel for the communication between the server and the client.
local smap_channel = L4.default_loader:new_channel();

-- The server program, using the 'smap' channel in server
-- mode. The log prefix will be 'server', colored yellow.
L4.default_loader:start({ caps = { smap = smap_channel:svr() },
 log = { "server", "yellow" }},
 "rom/ex_smap-server");

-- The client program.
-- It is given the 'smap' channel to be able to talk to the server.
-- The log prefix will be 'client', colored green.
L4.default_loader:start({ caps = { smap = smap_channel },
 log = { "client", "green" }},
 "rom/ex_smap-client");
```

## 18.21 examples/libs/libirq/loop.c

libirq usage example using a self-created thread.

libirq usage example using a self-created thread.

```
/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
 * economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */

#include <l4/irq/irq.h>
#include <l4/util/util.h>
#include <stdio.h>
#include <pthread.h>

enum { IRQ_NO = 17 };

static void isr_handler(void)
{
 printf("Got IRQ %d\n", IRQ_NO);
}

static void *isr_thread(void *data)
{
 l4irq_t *irq;
 (void)data;

 if (!(irq = l4irq_attach(IRQ_NO)))
 return NULL;

 while (1)
 {
 if (l4irq_wait(irq))
 continue;
 isr_handler();
 }

 return NULL;
}

int main(void)
{
 pthread_t thread;

 if (pthread_create(&thread, NULL, isr_thread, NULL))
 return 1;

 l4_sleep_forever();
 return 0;
}
```

## 18.22 examples/libs/libirq/async\_isr.c

libirq usage example using asynchronous ISR handler functionality.

libirq usage example using asynchronous ISR handler functionality.

```

/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
 * economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
/*
 * This example shall show how to use the libirq.
 */

#include <l4/irq/irq.h>
#include <l4/util/util.h>

#include <stdio.h>

enum { IRQ_NO = 17 };

static void isr_handler(void *data)
{
 (void)data;
 printf("Got IRQ %d\n", IRQ_NO);
}

int main(void)
{
 const int seconds = 5;
 l4irq_t *irqdesc;

 if (!(irqdesc = l4irq_request(IRQ_NO, isr_handler, 0, 0xff, 0)))
 {
 printf("Requesting IRQ %d failed\n", IRQ_NO);
 return 1;
 }

 printf("Attached to key IRQ %d\nPress keys now, will terminate in %d seconds\n",
 IRQ_NO, seconds);

 l4_sleep(seconds * 1000);

 if (l4irq_release(irqdesc))
 {
 printf("Failed to release IRQ\n");
 return 1;
 }

 printf("Bye\n");
 return 0;
}

```

## 18.23 examples/sys/migrate/thread\_migrate.cc

Thread migration example.

Thread migration example.

```

/*
 * (c) 2008-2009 Author(s)
 * economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
#include <l4/sys/scheduler>
#include <l4/re/env>
#include <l4/re/util/cap_alloc>

#include <pthread-l4.h>
#include <unistd.h>

```

```

#include <stdio.h>
#include <string.h>

enum { NR_THREADS = 12 };
static L4::Cap<L4::Thread> threads[NR_THREADS];
static L4_umword_t cpu_map, cpu_nrs;

/* Function for the threads. The content is not really relevant, so lets
 * just sleep around a bit. */
static void *thread_fn(void *)
{
 while (1)
 sleep(1);

 return 0;
}

/* Check how many CPUs we have available.
 */
static int check_cpus(void)
{
 l4_sched_cpu_set_t cs = l4_sched_cpu_set(0, 0);

 if (l4_error(L4Re::Env::env()->scheduler()->info(&cpu_nrs, &cs)) < 0)
 return 1;

 cpu_map = cs.map;

 printf("%ld maximal supported CPUs.\n", cpu_nrs);
 if (cpu_nrs >= L4_MWORD_BITS)
 {
 printf("Will only handle %ld CPUs.\n", cpu_nrs);
 cpu_nrs = L4_MWORD_BITS;
 }
 else if (cpu_nrs == 1)
 printf("Only found 1 CPU.\n");

 return cpu_nrs < 2;
}

/* Create a couple of threads and store their capabilities in an array */
static int create_threads(void)
{
 unsigned i;

 for (i = 0; i < NR_THREADS; ++i)
 {
 pthread_t t;

 if (pthread_create(&t, NULL, thread_fn, NULL))
 return 1;

 threads[i] = L4::Cap<L4::Thread>(pthread_l4_cap(t));
 }
 printf("Created %d threads.\n", NR_THREADS);
 return 0;
}

/* Helper function to get the next CPU */
static unsigned get_next_cpu(unsigned c)
{
 unsigned x = c;
 for (;;)
 {
 x = (x + 1) % cpu_nrs;
 if (L4Re::Env::env()->scheduler()->is_online(x))
 return x;
 if (x == c)
 return c;
 }
}

/* Function that shuffles the threads on the available CPUs */
static void shuffle(void)
{
 unsigned start = 0;
 while (1)
 {
 unsigned t;
 unsigned c = start;
 for (t = 0; t < NR_THREADS; ++t)
 {
 l4_sched_param_t sp = l4_sched_param(20);
 c = get_next_cpu(c);
 sp.affinity = l4_sched_cpu_set(c, 0);
 if (l4_error(L4Re::Env::env()->scheduler()->run_thread(threads[t], sp)))
 printf("Error migrating thread%02d to CPU%02d\n", t, c);
 }
 start = c;
 }
}

```

```

 printf("Migrated Thread%02d -> CPU%02d\n", t, c);
 }

 start++;
 if (start == cpu_nrs)
 start = 0;
 sleep(1);
}

int main(void)
{
 if (check_cpus())
 return 1;

 if (create_threads())
 return 1;

 shuffle();

 return 0;
}

```

## 18.24 examples/sys/migrate/thread\_migrate.cfg

Sample configuration file for the thread migration example.

Sample configuration file for the thread migration example.

```

-- vim:set ft=lua:

local L4 = require("L4");

-- The log prefix will be 'migrate', colored green.
L4.default_loader:start({ log = { "migrate", "green" } },
 "rom/ex_thread_migrate");

```

## 18.25 tmpfs/lib/src/fs.cc

Example file system for [L4Re::Vfs](#).

Example file system for [L4Re::Vfs](#).

```

/*
 * (c) 2010 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 * Alexander Warg <warg@os.inf.tu-dresden.de>
 * economic rights: Technische Universität Dresden (Germany)
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU Lesser General Public License 2.1.
 * Please see the COPYING-LGPL-2.1 file for details.
 */

#include <l4/l4re_vfs/backend>
#include <l4/cxx/string>
#include <l4/cxx/avl_tree>

#include <sys/stat.h>
#include <sys/ioctl.h>
#include <dirent.h>

#include <cstdio>
#include <cstdlib>
#include <cstring>

namespace {

using namespace L4Re::Vfs;
using cxx::Ref_ptr;

class File_data
{
public:

```

```

File_data() : _buf(0), _size(0) {}

unsigned long put(unsigned long offset,
 unsigned long bufsize, void *srcbuf);
unsigned long get(unsigned long offset,
 unsigned long bufsize, void *dstbuf);

unsigned long size(unsigned long offset);
unsigned long size() const { return _size; }

~File_data() noexcept { free(_buf); }

private:
 void *_buf;
 unsigned long _size;
};

unsigned long
File_data::put(unsigned long offset, unsigned long bufsize, void *srcbuf)
{
 if (offset + bufsize > _size)
 size(offset + bufsize);

 if (!_buf)
 return 0;

 memcpy((char *)_buf + offset, srcbuf, bufsize);
 return bufsize;
}

unsigned long
File_data::get(unsigned long offset, unsigned long bufsize, void *dstbuf)
{
 unsigned long s = bufsize;

 if (offset > _size)
 return 0;

 if (offset + bufsize > _size)
 s = _size - offset;

 memcpy(dstbuf, (char *)_buf + offset, s);
 return s;
}

unsigned long
File_data::size(unsigned long offset)
{
 if (offset != _size)
 {
 _size = offset;
 _buf = realloc(_buf, _size);
 }

 if (!_buf)
 return 0;
 return -ENOSPC;
}

class Node : public cxx::Avl_tree_node
{
public:
 Node(const char *path, mode_t mode)
 : _ref_cnt(0), _path(strdup(path))
 {
 memset(&_info, 0, sizeof(_info));
 _info.st_mode = mode;
 }

 const char *path() const { return _path; }
 struct stat64 *info() { return &_info; }

 void add_ref() noexcept { ++_ref_cnt; }
 int remove_ref() noexcept { return --_ref_cnt; }

 bool is_dir() const { return S_ISDIR(_info.st_mode); }

 virtual ~Node() { free(_path); }

private:
 int _ref_cnt;
 char *_path;
 struct stat64 _info;
};

struct Node_get_key

```



```

{
 typedef cxx::String Key_type;
 static Key_type key_of(Node const *n)
 { return n->path(); }
};

struct Path_avl_tree_compare
{
 bool operator () (const char *l, const char *r) const
 { return strcmp(l, r) < 0; }
 bool operator () (const cxx::String l, const cxx::String r) const
 {
 int v = strncmp(l.start(), r.start(), cxx::min(l.len(), r.len()));
 return v < 0 || (v == 0 && l.len() < r.len());
 }
};

class Pers_file : public Node
{
public:
 Pers_file(const char *name, mode_t mode)
 : Node(name, (mode & 0777) | __S_IFREG) {}
 File_data const &data() const { return _data; }
 File_data &data() { return _data; }
private:
 File_data _data;
};

class Pers_dir : public Node
{
private:
 typedef cxx::Avl_tree<Node, Node_get_key, Path_avl_tree_compare> Tree;
 Tree _tree;

public:
 Pers_dir(const char *name, mode_t mode)
 : Node(name, (mode & 0777) | __S_IFDIR) {}
 Ref_ptr<Node> find_path(cxx::String);
 bool add_node(Ref_ptr<Node> const &);

 typedef Tree::Const_iterator Const_iterator;
 Const_iterator begin() const { return _tree.begin(); }
 Const_iterator end() const { return _tree.end(); }
};

Ref_ptr<Node> Pers_dir::find_path(cxx::String path)
{
 return cxx::ref_ptr(_tree.find_node(path));
}

bool Pers_dir::add_node(Ref_ptr<Node> const &n)
{
 bool e = _tree.insert(n.ptr()).second;
 if (e)
 n->add_ref();
 return e;
}

class Tmpfs_dir : public Be_file
{
public:
 explicit Tmpfs_dir(Ref_ptr<Pers_dir> const &d) noexcept
 : _dir(d), _getdents_state(false) {}
 int get_entry(const char *, int, mode_t, Ref_ptr<File> *) noexcept override;
 ssize_t getdents(char *, size_t) noexcept override;
 int fstat(struct stat64 *buf) const noexcept override;
 int utime(const struct utimbuf *) noexcept override;
 int fchmod(mode_t) noexcept override;
 int mkdir(const char *, mode_t) noexcept override;
 int unlink(const char *) noexcept override;
 int rename(const char *, const char *) noexcept override;
 int faccessat(const char *, int, int) noexcept override;

private:
 int walk_path(cxx::String const &s,
 Ref_ptr<Node> *ret, cxx::String *remaining = 0);

 Ref_ptr<Pers_dir> _dir;
 bool _getdents_state;
 Pers_dir::Const_iterator _getdents_iter;
};

class Tmpfs_file : public Be_file_pos
{
public:
 explicit Tmpfs_file(Ref_ptr<Pers_file> const &f) noexcept

```

```

 : Be_file_pos(), _file(f) {}

 off64_t size() const noexcept;
 int fstat(struct stat64 *buf) const noexcept override;
 int ftruncate(off64_t p) noexcept override;
 int ioctl(unsigned long, va_list) noexcept override;
 int utime(const struct utimbuf *) noexcept override;
 int fchmod(mode_t) noexcept override;

private:
 ssize_t preadv(const struct iovec *v, int iovcnt, off64_t p) noexcept
 override;
 ssize_t pwritev(const struct iovec *v, int iovcnt, off64_t p) noexcept
 override;
 Ref_ptr<Pers_file> _file;
};

ssize_t Tmpfs_file::preadv(const struct iovec *v, int iovcnt, off64_t p)
 noexcept
{
 if (iovcnt < 0)
 return -EINVAL;

 ssize_t sum = 0;
 for (int i = 0; i < iovcnt; ++i)
 {
 sum += _file->data().get(p, v[i].iov_len, v[i].iov_base);
 p += v[i].iov_len;
 }
 return sum;
}

ssize_t Tmpfs_file::pwritev(const struct iovec *v, int iovcnt, off64_t p)
 noexcept
{
 if (iovcnt < 0)
 return -EINVAL;

 ssize_t sum = 0;
 for (int i = 0; i < iovcnt; ++i)
 {
 sum += _file->data().put(p, v[i].iov_len, v[i].iov_base);
 p += v[i].iov_len;
 }
 return sum;
}

int Tmpfs_file::fstat(struct stat64 *buf) const noexcept
{
 _file->info()->st_size = _file->data().size();
 memcpy(buf, _file->info(), sizeof(*buf));
 return 0;
}

int Tmpfs_file::ftruncate(off64_t p) noexcept
{
 if (p < 0)
 return -EINVAL;

 if (_file->data().size(p) == 0)
 return 0;

 return -EIO; // most likely ENOSPC, but can't report that
}

off64_t Tmpfs_file::size() const noexcept
{
 return _file->data().size();
}

int
Tmpfs_file::ioctl(unsigned long v, va_list args) noexcept
{
 switch (v)
 {
 case FIONREAD: // return amount of data still available
 {
 int *available = va_arg(args, int *);
 *available = _file->data().size() - pos();
 return 0;
 }
 default:
 {
 return -EINVAL;
 }
 }
}

int
Tmpfs_file::utime(const struct utimbuf *times) noexcept
{
 _file->info()->st_atime = times->actime;
 _file->info()->st_mtime = times->modtime;
 return 0;
}

```

```

}

int
Tmpfs_file::fchmod(mode_t m) noexcept
{
 _file->info()->st_mode = m;
 return 0;
}

int
Tmpfs_dir::faccessat(const char *path, int mode, int) noexcept
{
 Ref_ptr<Node> node;
 cxx::String name = path;

 int err = walk_path(name, &node, &name);
 if (err < 0)
 return err;

 if (mode == F_OK) // existence check
 return 0;

 struct stat64 *stats = node->info();

 if ((mode & R_OK) && !(stats->st_mode & S_IRUSR))
 return -EACCES;

 if ((mode & W_OK) && !(stats->st_mode & S_IWUSR))
 return -EACCES;

 if ((mode & X_OK) && !(stats->st_mode & S_IXUSR))
 return -EACCES;

 return 0;
}

int
Tmpfs_dir::get_entry(const char *name, int flags, mode_t mode,
 Ref_ptr<File> *file) noexcept
{
 Ref_ptr<Node> path;
 if (!*name)
 {
 *file = cxx::ref_ptr(this);
 return 0;
 }

 cxx::String n = name;

 int e = walk_path(n, &path, &n);

 if (e == -ENOTDIR)
 return e;

 if (!(flags & O_CREAT) && e < 0)
 return e;

 if ((flags & O_CREAT) && e == -ENOENT)
 {
 Ref_ptr<Node> node(new Pers_file(n.start(), mode));
 // when ENOENT is return, path is always a directory
 bool e = cxx::ref_ptr_static_cast<Pers_dir>(path)->add_node(node);
 if (!e)
 return -ENOMEM;
 path = node;
 }

 if (path->is_dir())
 *file = cxx::ref_ptr(new Tmpfs_dir(cxx::ref_ptr_static_cast<Pers_dir>(path)));
 else
 *file = cxx::ref_ptr(new Tmpfs_file(cxx::ref_ptr_static_cast<Pers_file>(path)));

 if (!*file)
 return -ENOMEM;

 return 0;
}

ssize_t
Tmpfs_dir::getdents(char *buf, size_t sz) noexcept
{
 struct dirent64 *d = (struct dirent64 *)buf;
 ssize_t ret = 0;

 if (!_getdents_state)
 {

```

```

 _getdents_iter = _dir->begin();
 _getdents_state = true;
}
else if (_getdents_iter == _dir->end())
{
 _getdents_state = false;
 return 0;
}

for (; _getdents_iter != _dir->end(); ++_getdents_iter)
{
 unsigned l = strlen(_getdents_iter->path()) + 1;
 if (l > sizeof(d->d_name))
 l = sizeof(d->d_name);

 unsigned n = offsetof (struct dirent64, d_name) + 1;
 n = (n + sizeof(long) - 1) & ~(sizeof(long) - 1);

 if (n > sz)
 break;

 d->d_ino = 1;
 d->d_off = 0;
 memcpy(d->d_name, _getdents_iter->path(), l);
 d->d_reclen = n;
 d->d_type = DT_REG;
 ret += n;
 sz -= n;
 d = (struct dirent64 *) ((unsigned long)d + n);
}

return ret;
}

int
Tmpfs_dir::fstat(struct stat64 *buf) const noexcept
{
 memcpy(buf, _dir->info(), sizeof(*buf));
 return 0;
}

int
Tmpfs_dir::utime(const struct utimbuf *times) noexcept
{
 _dir->info()->st_atime = times->actime;
 _dir->info()->st_mtime = times->modtime;
 return 0;
}

int
Tmpfs_dir::fchmod(mode_t m) noexcept
{
 _dir->info()->st_mode = m;
 return 0;
}

int
Tmpfs_dir::walk_path(cxx::String const &s,
 Ref_ptr<Node> *ret, cxx::String *remaining)
{
 Ref_ptr<Pers_dir> p = _dir;
 cxx::String s = s;
 Ref_ptr<Node> n;

 while (1)
 {
 if (s.len() == 0)
 {
 *ret = p;
 return 0;
 }

 cxx::String::Index sep = s.find("/");

 if (sep - s.start() == 1 && *s.start() == '.')
 {
 s = s.substr(s.start() + 2);
 continue;
 }

 n = p->find_path(s.head(sep - s.start()));

 if (!n)
 {
 *ret = p;
 if (remaining)
 *remaining = s.head(sep - s.start());
 }
 }
}

```

```

 return -ENOENT;
 }

 if (sep == s.end())
 {
 *ret = n;
 return 0;
 }

 if (!n->is_dir())
 return -ENOTDIR;

 s = s.substr(sep + 1);

 p = cxx::ref_ptr_static_cast<Pers_dir>(n);
}

*ret = n;

return 0;
}

int
Tmpfs_dir::mkdir(const char *name, mode_t mode) noexcept
{
 Ref_ptr<Node> node = _dir;
 cxx::String p = cxx::String(name);
 cxx::String path, last = p;
 cxx::String::Index s = p.rfind("/");

 // trim '/'s at the end
 while (p.len() && s == p.end() - 1)
 {
 p.len(p.len() - 1);
 s = p.rfind("/");
 }

 //printf("MKDIR '%s' p=%p %p\n", name, p.start(), s);

 if (s != p.end())
 {
 path = p.head(s);
 last = p.substr(s + 1, p.end() - s);

 int e = walk_path(path, &node);
 if (e < 0)
 return e;
 }

 if (!node->is_dir())
 return -ENOTDIR;

 // due to path walking we can end up with an empty name
 if (p.len() == 0 || p == cxx::String("."))
 return 0;

 Ref_ptr<Pers_dir> dnode = cxx::ref_ptr_static_cast<Pers_dir>(node);

 Ref_ptr<Pers_dir> dir(new Pers_dir(last.start(), mode));
 return dnode->add_node(dir) ? 0 : -EEXIST;
}

int
Tmpfs_dir::unlink(const char *name) noexcept
{
 cxx::Ref_ptr<Node> n;

 int e = walk_path(name, &n);
 if (e < 0)
 return -ENOENT;

 printf("Unimplemented (if file exists): %s(%s)\n", __func__, name);
 return -ENOMEM;
}

int
Tmpfs_dir::rename(const char *old, const char *newn) noexcept
{
 printf("Unimplemented: %s(%s, %s)\n", __func__, old, newn);
 return -ENOMEM;
}

class Tmpfs_fs : public Be_file_system
{

```

```

public:
 Tmpfs_fs() : Be_file_system("tmpfs") {}
 int mount(char const *source, unsigned long mountflags,
 void const *data, cxx::Ref_ptr<File> *dir) noexcept override
 {
 (void)mountflags;
 (void)source;
 (void)data;
 *dir = cxx::ref_ptr(new Tmpfs_dir(cxx::ref_ptr(new Pers_dir("root", 0777))));
 if (!*dir)
 return -ENOMEM;
 return 0;
 }
};

static Tmpfs_fs _tmpfs L4RE_VFS_FILE_SYSTEM_ATTRIBUTE;
}

```

## 18.26 examples/libs/shmc/prodcons.c

Simple shared memory example.

Simple shared memory example.

```

/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
 * economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */

/*
 * This example uses shared memory between two threads, one producer, one
 * consumer.
 */

#include <l4/shmc/shmc.h>

#include <l4/util/util.h>

#include <stdio.h>
#include <string.h>
#include <pthread-l4.h>

#include <l4/sys/thread.h>
#include <l4/sys/debugger.h>
#include <l4/sys/kip.h>
#include <l4/re/env.h>

#define LOG(args...) printf(NAME ": " args)
#define CHK(func) do \
 { \
 long r = (func); \
 if (r) \
 { \
 printf(NAME ": Failure %ld (%s) at line %d.\n", \
 r, l4sys_errtostr(r), __LINE__); \
 return (void *)-1; \
 } \
 } while (0)

static const char some_data[] = "Hi consumer!";

static inline l4_cap_idx_t self(void) { return pthread_l4_cap(pthread_self()); }

#define NAME "PRODUCER"
static void *thread_producer(void *d)
{
 (void)d;
 l4shmc_chunk_t p_one;
 l4shmc_signal_t s_one, s_done;
 l4shmc_area_t shmarea;
 l4_kernel_clock_t try_until;

 l4_debugger_set_object_name(self(), "producer");

 // attach this thread to the shm object

```

```

CHK(l4shmc_attach("testshm", &shmarea));

// add a chunk
CHK(l4shmc_add_chunk(&shmarea, "one", 1024, &p_one));

// add a signal
CHK(l4shmc_add_signal(&shmarea, "testshm_prod", &s_one));

CHK(l4shmc_attach_signal(&shmarea, "testshm_done", self(), &s_done));

// connect chunk and signal
CHK(l4shmc_connect_chunk_signal(&p_one, &s_one));

CHK(l4shmc_mark_client_initialized(&shmarea));

try_until = l4_kip_clock(l4re_kip()) + 10 * 1000000;

for (;;)
{
 l4_umword_t clients;
 l4shmc_get_initialized_clients(&shmarea, &clients);
 if (clients == 3UL)
 break;
 if (l4_kip_clock(l4re_kip()) >= try_until)
 {
 LOG("consumer not initialized within time\n");
 return (void *)-1;
 }
}

LOG("Ready.\n");

while (1)
{
 while (l4shmc_chunk_try_to_take(&p_one))
 printf("Uh, should not happen!\n"); //l4_thread_yield();

 memcpy(l4shmc_chunk_ptr(&p_one), some_data, sizeof(some_data));

 CHK(l4shmc_chunk_ready_sig(&p_one, sizeof(some_data)));

 LOG("Sent data.\n");

 CHK(l4shmc_wait_signal(&s_done));
}

l4_sleep_forever();
return NULL;
}

#undef NAME
#define NAME "CONSUMER"
static void *thread_consumer(void *d)
{
 (void)d;
 l4shmc_area_t shmarea;
 l4shmc_chunk_t p_one;
 l4shmc_signal_t s_one, s_done;
 l4_kernel_clock_t try_until;

 l4_debugger_set_object_name(self(), "consumer");

 // attach to shared memory area
 CHK(l4shmc_attach("testshm", &shmarea));

 // get chunk 'one'
 CHK(l4shmc_get_chunk(&shmarea, "one", &p_one));

 // add a signal
 CHK(l4shmc_add_signal(&shmarea, "testshm_done", &s_done));

 // attach signal to this thread
 CHK(l4shmc_attach_signal(&shmarea, "testshm_prod", self(), &s_one));

 // connect chunk and signal
 CHK(l4shmc_connect_chunk_signal(&p_one, &s_one));

 CHK(l4shmc_mark_client_initialized(&shmarea));

 try_until = l4_kip_clock(l4re_kip()) + 10 * 1000000;

 for (;;)
 {
 l4_umword_t clients;
 l4shmc_get_initialized_clients(&shmarea, &clients);
 if (clients == 3UL)

```

```

 break;
 if (l4_kip_clock(l4re_kip()) >= try_until)
 {
 LOG("producer not initialized within time\n");
 return (void *)-1;
 }
}

LOG("Ready.\n");

while (1)
{
 CHK(l4shmc_wait_chunk(&p_one));

 LOG("Received from chunk one: '%s'.\n",
 (char *)l4shmc_chunk_ptr(&p_one));
 memset(l4shmc_chunk_ptr(&p_one), 0, l4shmc_chunk_size(&p_one));

 CHK(l4shmc_chunk_consumed(&p_one));

 CHK(l4shmc_trigger(&s_done));
}

return NULL;
}

int main(void)
{
 pthread_t one, two;
 long r;

 // create shared memory area
 if ((r = l4shmc_create("testshm")) < 0)
 {
 printf("Error %ld (%s) creating shared memory area\n",
 r, l4sys_errtostr(r));
 return 1;
 }

 // create two threads, one for producer, one for consumer
 pthread_create(&one, 0, thread_producer, 0);
 pthread_create(&two, 0, thread_consumer, 0);

 // now sleep, the two threads are doing the work
 l4_sleep_forever();

 return 0;
}

```