

L4Re – L4 Runtime Environment

Contents

1	Fiasco.OC & L4 Runtime Environment (L4Re)	1
1.1	Preface	1
1.2	General System Structure	1
1.3	The Fiasco.OC Microkernel	3
1.3.1	Communication	3
1.3.2	Kernel Objects	3
1.4	L4 Runtime Environment (L4Re)	4
1.5	Introduction to L4Re's concepts	4
1.6	Memory management - Data Spaces and the Region Map	4
1.6.1	User-level paging	4
1.6.2	Data spaces	4
1.6.3	Virtual Memory Handling	5
1.6.4	Memory Allocation	5
1.7	Capabilities and Naming	5
1.8	Initial Environment and Application Bootstrapping	6
1.8.1	Configuring an application before startup	7
1.8.2	Connecting clients and servers	7
1.9	Program Input and Output	8
1.10	Initial Memory Allocator and Factory	8
1.11	Application and Server Building Blocks	8
1.11.1	Creating Additional Application Threads	9
1.11.2	Providing a Service	9
2	Getting Started	11
3	L4Re Servers	13
3.1	Sigma0, the Root Pager	13
3.2	Moe, the Root Task	13
3.3	Ned, the Default Init Process	14

3.4	Io, the Platform and Device Resource Manager	14
3.5	Mag, the GUI Multiplexer	14
3.6	fb-drv, the Low-Level Graphics Driver	14
3.7	Rtc, the Real-Time Clock Server	14
3.8	Moe, the Root-Task	14
3.8.1	Memory Allocator, Generic Factory	15
3.8.2	Name-Space Provider	15
3.8.3	Boot FS	16
3.8.4	Log Subsystem	16
3.8.5	Command-Line Options	16
3.8.5.1	--debug=<debug flags>	16
3.8.5.2	--init=<init process>	16
3.8.5.3	--l4re-dbg=<debug flags>	16
3.8.5.4	--ldr-flags=<loader flags>	16
3.9	Ned, the Init Process	16
3.9.1	Lua Bindings for L4Re	17
3.9.1.1	Capabilities in Lua	17
3.9.1.2	Access to L4Re::Env Capabilities	17
3.9.1.3	Constants	17
3.9.1.4	Application Startup Details	18
3.10	Io, the Io Server	19
4	Pthread Support	23
5	Module Index	25
5.1	Modules	25
6	Namespace Index	29
6.1	Namespace List	29
7	Data Structure Index	31
7.1	Class Hierarchy	31
8	Data Structure Index	37
8.1	Data Structures	37
9	Module Documentation	43
9.1	C++ Exceptions	43
9.2	Small C++ Template Library	44

9.2.1	Function Documentation	46
9.2.1.1	min	46
9.2.1.2	max	46
9.2.1.3	operator new	46
9.3	Client/Server IPC Framework	46
9.4	IPC Streams	47
9.4.1	Function Documentation	47
9.4.1.1	operator>>	47
9.4.1.2	operator>>	48
9.4.1.3	operator>>	49
9.4.1.4	operator>>	49
9.4.1.5	operator>>	50
9.4.1.6	operator<<	50
9.4.1.7	operator<<	51
9.4.1.8	operator<<	52
9.4.1.9	operator<<	53
9.5	IPC Messaging Framework	53
9.5.1	Function Documentation	54
9.5.1.1	buf_cp_out	54
9.5.1.2	buf_cp_in	54
9.5.1.3	msg_ptr	55
9.5.1.4	buf_in	56
9.6	L4Re C++ Interface	56
9.6.1	Detailed Description	58
9.7	L4Re Util C++ Interface	59
9.7.1	Detailed Description	60
9.8	Dataspace interface	60
9.8.1	Detailed Description	61
9.8.2	Function Documentation	61
9.8.2.1	l4re_ds_clear	61
9.8.2.2	l4re_ds_allocate	61
9.8.2.3	l4re_ds_copy_in	61
9.8.2.4	l4re_ds_size	61
9.8.2.5	l4re_ds_flags	61
9.8.2.6	l4re_ds_info	62
9.8.2.7	l4re_ds_phys	62

9.9	Debug interface	62
9.9.1	Function Documentation	63
9.9.1.1	<code>l4re_debug_obj_debug</code>	63
9.10	Event interface	63
9.10.1	Detailed Description	64
9.10.2	Function Documentation	64
9.10.2.1	<code>l4re_event_get_buffer</code>	64
9.10.2.2	<code>l4re_event_get_num_streams</code>	64
9.10.2.3	<code>l4re_event_get_stream_info</code>	64
9.10.2.4	<code>l4re_event_get_stream_info_for_id</code>	65
9.10.2.5	<code>l4re_event_get_axis_info</code>	65
9.11	Log interface	66
9.11.1	Detailed Description	66
9.11.2	Function Documentation	66
9.11.2.1	<code>l4re_log_print</code>	66
9.11.2.2	<code>l4re_log_printn</code>	67
9.11.2.3	<code>l4re_log_print_srv</code>	67
9.11.2.4	<code>l4re_log_printn_srv</code>	68
9.12	Memory allocator	69
9.12.1	Detailed Description	70
9.12.2	Enumeration Type Documentation	70
9.12.2.1	<code>l4re_ma_flags</code>	70
9.12.3	Function Documentation	70
9.12.3.1	<code>l4re_ma_alloc</code>	70
9.12.3.2	<code>l4re_ma_free</code>	71
9.12.3.3	<code>l4re_ma_alloc_srv</code>	72
9.12.3.4	<code>l4re_ma_free_srv</code>	72
9.13	Namespace interface	73
9.13.1	Detailed Description	73
9.13.2	Enumeration Type Documentation	74
9.13.2.1	<code>l4re_ns_register_flags</code>	74
9.13.3	Function Documentation	74
9.13.3.1	<code>l4re_ns_query_to_srv</code>	74
9.13.3.2	<code>l4re_ns_register_obj_srv</code>	74
9.14	Region map interface	74
9.14.1	Detailed Description	75

9.14.2 Enumeration Type Documentation	76
9.14.2.1 <code>l4re_rm_flags_t</code>	76
9.14.3 Function Documentation	76
9.14.3.1 <code>l4re_rm_reserve_area</code>	76
9.14.3.2 <code>l4re_rm_free_area</code>	77
9.14.3.3 <code>l4re_rm_attach</code>	77
9.14.3.4 <code>l4re_rm_detach</code>	78
9.14.3.5 <code>l4re_rm_detach_ds</code>	79
9.14.3.6 <code>l4re_rm_detach_unmap</code>	80
9.14.3.7 <code>l4re_rm_detach_ds_unmap</code>	80
9.14.3.8 <code>l4re_rm_find</code>	81
9.14.3.9 <code>l4re_rm_show_lists</code>	82
9.14.3.10 <code>l4re_rm_reserve_area_srv</code>	82
9.14.3.11 <code>l4re_rm_free_area_srv</code>	83
9.14.3.12 <code>l4re_rm_attach_srv</code>	83
9.14.3.13 <code>l4re_rm_detach_srv</code>	84
9.14.3.14 <code>l4re_rm_find_srv</code>	84
9.15 Capability allocator	85
9.15.1 Detailed Description	85
9.15.2 Function Documentation	85
9.15.2.1 <code>l4re_util_cap_last</code>	85
9.16 Kumem allocator utility	85
9.16.1 Detailed Description	86
9.16.2 Function Documentation	86
9.16.2.1 <code>l4re_util_kumem_alloc</code>	86
9.17 Video API	87
9.17.1 Typedef Documentation	89
9.17.1.1 <code>l4re_video_view_t</code>	89
9.17.2 Enumeration Type Documentation	89
9.17.2.1 <code>l4re_video_goops_info_flags_t</code>	89
9.17.2.2 <code>l4re_video_view_info_flags_t</code>	89
9.17.3 Function Documentation	90
9.17.3.1 <code>l4re_video_goops_info</code>	90
9.17.3.2 <code>l4re_video_goops_refresh</code>	90
9.17.3.3 <code>l4re_video_goops_create_buffer</code>	91
9.17.3.4 <code>l4re_video_goops_delete_buffer</code>	91

9.17.3.5 l4re_video_goops_get_static_buffer	91
9.17.3.6 l4re_video_goops_create_view	91
9.17.3.7 l4re_video_goops_delete_view	92
9.17.3.8 l4re_video_goops_get_view	92
9.17.3.9 l4re_video_view_refresh	92
9.17.3.10 l4re_video_view_get_info	92
9.17.3.11 l4re_video_view_set_info	93
9.17.3.12 l4re_video_view_set_viewport	93
9.17.3.13 l4re_video_view_stack	93
9.18 Console API	93
9.18.1 Detailed Description	94
9.19 Data-Space API	94
9.19.1 Detailed Description	95
9.20 Debugging API	95
9.20.1 Detailed Description	95
9.21 L4Re ELF Auxiliary Information	95
9.21.1 Detailed Description	97
9.21.2 Define Documentation	97
9.21.2.1 L4RE_ELF_AUX_ELEM	97
9.21.2.2 L4RE_ELF_AUX_ELEM_T	97
9.21.3 Enumeration Type Documentation	98
9.21.3.1 "@52	98
9.22 Initial Environment	98
9.22.1 Detailed Description	99
9.22.2 Typedef Documentation	100
9.22.2.1 l4re_env_t	100
9.22.3 Function Documentation	100
9.22.3.1 l4re_env	100
9.22.3.2 l4re_kip	100
9.22.3.3 l4re_env_get_cap	101
9.22.3.4 l4re_env_get_cap_e	101
9.22.3.5 l4re_env_get_cap_l	102
9.23 Event API	103
9.23.1 Detailed Description	103
9.24 Auxiliary data	104
9.25 Logging interface	104

9.25.1	Detailed Description	105
9.26	Memory allocator API	105
9.26.1	Detailed Description	105
9.27	Name-space API	106
9.27.1	Detailed Description	106
9.28	Parent API	106
9.28.1	Detailed Description	107
9.29	L4Re Protocol identifiers	107
9.29.1	Detailed Description	108
9.29.2	Enumeration Type Documentation	108
9.29.2.1	Protocols	108
9.30	Region map API	108
9.30.1	Detailed Description	109
9.31	L4Re Capability API	109
9.31.1	Variable Documentation	110
9.31.1.1	cap_alloc	110
9.32	Kumem utilities	111
9.32.1	Function Documentation	111
9.32.1.1	kumem_alloc	111
9.33	Goos video API	112
9.34	L4Re C Interface	112
9.34.1	Detailed Description	114
9.35	L4Re Util C Interface	114
9.36	Base API	115
9.36.1	Detailed Description	119
9.37	IPC-Gate API	119
9.37.1	Detailed Description	120
9.37.2	Enumeration Type Documentation	120
9.37.2.1	l4_ipc_gate_ops	120
9.37.3	Function Documentation	120
9.37.3.1	l4_ipc_gate_bind_thread	120
9.37.3.2	l4_ipc_gate_get_infos	121
9.38	Basic Macros	122
9.38.1	Detailed Description	123
9.38.2	Define Documentation	123
9.38.2.1	L4_DECLARE_CONSTRUCTOR	123

9.38.2.2	L4_NO_THROW	124
9.38.2.3	L4_EXPORT	124
9.38.2.4	L4_HIDDEN	124
9.39	Fiasco extensions	125
9.39.1	Detailed Description	128
9.39.2	Function Documentation	128
9.39.2.1	fiasco_tbuf_get_status	128
9.39.2.2	fiasco_tbuf_get_status_phys	128
9.39.2.3	fiasco_tbuf_log	128
9.39.2.4	fiasco_tbuf_log_3val	129
9.39.2.5	fiasco_tbuf_log_binary	130
9.39.2.6	fiasco_tbuf_clear	130
9.39.2.7	fiasco_tbuf_dump	131
9.39.2.8	fiasco_watchdog_takeover	131
9.39.2.9	fiasco_watchdog_touch	131
9.39.2.10	fiasco_ldt_set	131
9.39.2.11	fiasco_gdt_set	132
9.39.2.12	fiasco_gdt_get_entry_offset	133
9.40	Fiasco real time scheduling extensions	133
9.40.1	Detailed Description	134
9.40.2	Function Documentation	134
9.40.2.1	slice	134
9.40.2.2	slice	135
9.40.2.3	l4_rt_begin_strictly_periodic	135
9.40.2.4	l4_rt_begin_minimal_periodic	136
9.40.2.5	l4_rt_end_periodic	137
9.40.2.6	l4_rt_remove	137
9.40.2.7	l4_rt_set_period	138
9.40.2.8	l4_rt_next_reservation	139
9.40.2.9	l4_rt_next_period	139
9.40.2.10	l4_preemption_id	140
9.40.2.11	l4_next_period_id	140
9.40.2.12	l4_rt_generic	141
9.41	Flex pages	142
9.41.1	Detailed Description	144
9.41.2	Enumeration Type Documentation	144

9.41.2.1	l4_fpage_consts	144
9.41.2.2	"@60	145
9.41.2.3	L4_fpage_rights	145
9.41.2.4	L4_cap_fpage_rights	145
9.41.2.5	l4_fpage_cacheability_opt_t	146
9.41.2.6	"@61	146
9.41.3	Function Documentation	146
9.41.3.1	l4_fpage	146
9.41.3.2	l4_fpage_all	146
9.41.3.3	l4_fpage_invalid	147
9.41.3.4	l4_iofpage	147
9.41.3.5	l4_obj_fpage	147
9.41.3.6	l4_is_fpage_writable	148
9.41.3.7	l4_fpage_rights	148
9.41.3.8	l4_fpage_type	149
9.41.3.9	l4_fpage_size	149
9.41.3.10	l4_fpage_page	149
9.41.3.11	l4_fpage_set_rights	150
9.41.3.12	l4_fpage_contains	150
9.41.3.13	l4_fpage_max_order	150
9.42	Message Items	151
9.42.1	Detailed Description	152
9.42.2	Enumeration Type Documentation	152
9.42.2.1	l4_msg_item_consts_t	152
9.42.3	Function Documentation	153
9.42.3.1	l4_map_control	153
9.42.3.2	l4_map_obj_control	153
9.43	Timeouts	154
9.43.1	Detailed Description	156
9.43.2	Define Documentation	156
9.43.2.1	L4_IPC_TIMEOUT_0	156
9.43.3	Typedef Documentation	156
9.43.3.1	l4_timeout_s	156
9.43.3.2	l4_timeout_t	156
9.43.4	Enumeration Type Documentation	157
9.43.4.1	l4_timeout_abs_validity	157

9.43.5 Function Documentation	157
9.43.5.1 l4_timeout_rel	157
9.43.5.2 l4_ipc_timeout	157
9.43.5.3 l4_timeout	157
9.43.5.4 l4_snd_timeout	158
9.43.5.5 l4_rcv_timeout	158
9.43.5.6 l4_timeout_rel_get	158
9.43.5.7 l4_timeout_is_absolute	159
9.43.5.8 l4_timeout_get	159
9.43.5.9 l4_timeout_abs	160
9.44 VM API for SVM	161
9.44.1 Detailed Description	161
9.45 VM API for VMX	162
9.45.1 Detailed Description	162
9.45.2 Function Documentation	162
9.45.2.1 l4_vm_vmx_field_len	162
9.45.2.2 l4_vm_vmx_field_ptr	163
9.46 Cache Consistency	163
9.46.1 Detailed Description	164
9.46.2 Function Documentation	164
9.46.2.1 l4_cache_clean_data	164
9.46.2.2 l4_cache_flush_data	165
9.46.2.3 l4_cache_inv_data	165
9.46.2.4 l4_cache_coherent	165
9.46.2.5 l4_cache_dma_coherent	165
9.47 Memory related	166
9.47.1 Detailed Description	167
9.47.2 Define Documentation	167
9.47.2.1 L4_PAGEMASK	167
9.47.2.2 L4_LOG2_PAGESIZE	168
9.47.2.3 L4_SUPERPAGESIZE	168
9.47.2.4 L4_SUPERPAGEMASK	168
9.47.2.5 L4_LOG2_SUPERPAGESIZE	168
9.47.3 Enumeration Type Documentation	168
9.47.3.1 l4_addr_consts_t	168
9.47.4 Function Documentation	168

9.47.4.1	l4_trunc_page	168
9.47.4.2	l4_trunc_size	169
9.47.4.3	l4_round_page	170
9.47.4.4	l4_round_size	170
9.48	Kernel Debugger	171
9.48.1	Detailed Description	172
9.48.2	Define Documentation	173
9.48.2.1	enter_kdebug	173
9.48.2.2	asm_enter_kdebug	173
9.48.2.3	kd_display	173
9.48.2.4	ko	174
9.48.2.5	enter_kdebug	174
9.48.2.6	asm_enter_kdebug	174
9.48.2.7	kd_display	175
9.48.2.8	ko	175
9.48.3	Function Documentation	175
9.48.3.1	l4_debugger_set_object_name	175
9.48.3.2	l4_debugger_global_id	176
9.48.3.3	l4_debugger_kobj_to_id	176
9.48.3.4	outchar	177
9.48.3.5	outstring	177
9.48.3.6	outnstring	177
9.48.3.7	outhex32	178
9.48.3.8	outhex20	178
9.48.3.9	outhex16	178
9.48.3.10	outhex12	178
9.48.3.11	outhex8	178
9.48.3.12	outdec	178
9.48.3.13	l4kd_inchar	179
9.49	Error codes	179
9.49.1	Detailed Description	179
9.49.2	Enumeration Type Documentation	179
9.49.2.1	l4_error_code_t	179
9.50	Factory	180
9.50.1	Detailed Description	181
9.50.2	Function Documentation	181

9.50.2.1	l4_factory_create_task	181
9.50.2.2	l4_factory_create_thread	182
9.50.2.3	l4_factory_create_factory	183
9.50.2.4	l4_factory_create_gate	183
9.50.2.5	l4_factory_create_irq	184
9.50.2.6	l4_factory_create_vm	185
9.51	Virtual Machines	185
9.51.1	Detailed Description	186
9.52	Interrupt controller	186
9.52.1	Detailed Description	188
9.52.2	Typedef Documentation	188
9.52.2.1	l4_icu_info_t	188
9.52.3	Enumeration Type Documentation	188
9.52.3.1	L4_icu_flags	188
9.52.4	Function Documentation	188
9.52.4.1	l4_icu_bind	188
9.52.4.2	l4_icu_unbind	189
9.52.4.3	l4_icu_set_mode	190
9.52.4.4	l4_icu_info	190
9.52.4.5	l4_icu_msi_info	191
9.52.4.6	l4_icu_unmask	192
9.52.4.7	l4_icu_mask	192
9.53	Object Invocation	193
9.53.1	Detailed Description	195
9.53.2	Enumeration Type Documentation	195
9.53.2.1	l4_syscall_flags_t	195
9.53.3	Function Documentation	196
9.53.3.1	l4_ipc_send	196
9.53.3.2	l4_ipc_wait	197
9.53.3.3	l4_ipc_receive	198
9.53.3.4	l4_ipc_call	199
9.53.3.5	l4_ipc_reply_and_wait	200
9.53.3.6	l4_ipc_send_and_wait	201
9.53.3.7	l4_ipc	202
9.53.3.8	l4_ipc_sleep	202
9.53.3.9	l4_sndfpage_add	203

9.54 Error Handling	204
9.54.1 Detailed Description	205
9.54.2 Enumeration Type Documentation	205
9.54.2.1 <code>l4_ipc_tcr_error_t</code>	205
9.54.3 Function Documentation	206
9.54.3.1 <code>l4_ipc_error</code>	206
9.54.3.2 <code>l4_error</code>	206
9.54.3.3 <code>l4_ipc_is_snd_error</code>	208
9.54.3.4 <code>l4_ipc_is_rcv_error</code>	208
9.54.3.5 <code>l4_ipc_error_code</code>	208
9.55 Realtime API	209
9.56 IRQs	209
9.56.1 Detailed Description	210
9.56.2 Enumeration Type Documentation	210
9.56.2.1 <code>L4_irq_flow_type</code>	210
9.56.3 Function Documentation	211
9.56.3.1 <code>l4_irq_attach</code>	211
9.56.3.2 <code>l4_irq_chain</code>	211
9.56.3.3 <code>l4_irq_detach</code>	212
9.56.3.4 <code>l4_irq_trigger</code>	213
9.56.3.5 <code>l4_irq_receive</code>	213
9.56.3.6 <code>l4_irq_wait</code>	214
9.56.3.7 <code>l4_irq_unmask</code>	215
9.57 Kernel Objects	215
9.57.1 Detailed Description	217
9.57.2 Define Documentation	217
9.57.2.1 <code>L4_DISABLE_COPY</code>	217
9.57.2.2 <code>L4_KOBJECT_DISABLE_COPY</code>	218
9.57.2.3 <code>L4_KOBJECT</code>	218
9.58 Kernel Interface Page	219
9.58.1 Detailed Description	220
9.58.2 Function Documentation	220
9.58.2.1 <code>l4_kip_version</code>	220
9.58.2.2 <code>l4_kip_version_string</code>	220
9.58.2.3 <code>l4_kernel_info_version_offset</code>	221
9.59 Memory descriptors (C version)	222

9.59.1	Detailed Description	223
9.59.2	Typedef Documentation	223
9.59.2.1	<code>l4_kernel_info_mem_desc_t</code>	223
9.59.3	Enumeration Type Documentation	223
9.59.3.1	<code>l4_mem_type_t</code>	223
9.59.4	Function Documentation	224
9.59.4.1	<code>l4_kernel_info_get_num_mem_descs</code>	224
9.59.4.2	<code>l4_kernel_info_set_mem_desc</code>	224
9.59.4.3	<code>l4_kernel_info_get_mem_desc_start</code>	224
9.59.4.4	<code>l4_kernel_info_get_mem_desc_end</code>	225
9.59.4.5	<code>l4_kernel_info_get_mem_desc_type</code>	225
9.59.4.6	<code>l4_kernel_info_get_mem_desc_subtype</code>	225
9.59.4.7	<code>l4_kernel_info_get_mem_desc_is_virtual</code>	225
9.60	Scheduler	225
9.60.1	Detailed Description	227
9.60.2	Enumeration Type Documentation	227
9.60.2.1	<code>L4_scheduler_ops</code>	227
9.60.3	Function Documentation	227
9.60.3.1	<code>l4_sched_cpu_set</code>	227
9.60.3.2	<code>l4_scheduler_info</code>	228
9.60.3.3	<code>l4_scheduler_run_thread</code>	229
9.60.3.4	<code>l4_scheduler_idle_time</code>	229
9.60.3.5	<code>l4_scheduler_is_online</code>	230
9.61	Task	230
9.61.1	Detailed Description	232
9.61.2	Enumeration Type Documentation	232
9.61.2.1	<code>l4_unmap_flags_t</code>	232
9.61.3	Function Documentation	233
9.61.3.1	<code>l4_task_map</code>	233
9.61.3.2	<code>l4_task_unmap</code>	233
9.61.3.3	<code>l4_task_unmap_batch</code>	234
9.61.3.4	<code>l4_task_delete_obj</code>	235
9.61.3.5	<code>l4_task_release_cap</code>	235
9.61.3.6	<code>l4_task_cap_valid</code>	236
9.61.3.7	<code>l4_task_cap_has_child</code>	236
9.61.3.8	<code>l4_task_cap_equal</code>	237

9.61.3.9	l4_task_add_ku_mem	238
9.62	Thread	238
9.62.1	Detailed Description	240
9.62.2	Enumeration Type Documentation	241
9.62.2.1	L4_thread_ops	241
9.62.2.2	L4_thread_control_flags	242
9.62.2.3	L4_thread_control_mr_indices	242
9.62.2.4	L4_thread_ex_regs_flags	242
9.62.3	Function Documentation	242
9.62.3.1	l4_thread_ex_regs	242
9.62.3.2	l4_thread_ex_regs_ret	243
9.62.3.3	l4_thread_yield	244
9.62.3.4	l4_thread_switch	244
9.62.3.5	l4_thread_stats_time	245
9.62.3.6	l4_thread_vcpu_resume_start	245
9.62.3.7	l4_thread_vcpu_resume_commit	246
9.62.3.8	l4_thread_vcpu_control	247
9.62.3.9	l4_thread_vcpu_control_ext	247
9.62.3.10	l4_thread_register_del_irq	248
9.62.3.11	l4_thread_modify_sender_start	249
9.62.3.12	l4_thread_modify_sender_add	249
9.62.3.13	l4_thread_modify_sender_commit	250
9.63	Thread control	251
9.63.1	Detailed Description	252
9.63.2	Function Documentation	252
9.63.2.1	l4_thread_control_start	252
9.63.2.2	l4_thread_control_pager	253
9.63.2.3	l4_thread_control_exc_handler	253
9.63.2.4	l4_thread_control_bind	254
9.63.2.5	l4_thread_control_alien	255
9.63.2.6	l4_thread_control_ux_host_syscall	255
9.63.2.7	l4_thread_control_commit	256
9.64	Message Tag	257
9.64.1	Detailed Description	259
9.64.2	Typedef Documentation	259
9.64.2.1	l4_mshtag_t	259

9.64.3	Enumeration Type Documentation	259
9.64.3.1	<code>l4_msntag_protocol</code>	259
9.64.3.2	<code>l4_msntag_flags</code>	260
9.64.4	Function Documentation	260
9.64.4.1	<code>l4_msntag</code>	260
9.64.4.2	<code>l4_msntag_label</code>	261
9.64.4.3	<code>l4_msntag_words</code>	262
9.64.4.4	<code>l4_msntag_items</code>	263
9.64.4.5	<code>l4_msntag_flags</code>	263
9.64.4.6	<code>l4_msntag_has_error</code>	263
9.64.4.7	<code>l4_msntag_is_page_fault</code>	264
9.64.4.8	<code>l4_msntag_is_preemption</code>	264
9.64.4.9	<code>l4_msntag_is_sys_exception</code>	265
9.64.4.10	<code>l4_msntag_is_exception</code>	265
9.64.4.11	<code>l4_msntag_is_sigma0</code>	266
9.64.4.12	<code>l4_msntag_is_io_page_fault</code>	267
9.65	Capabilities	267
9.65.1	Detailed Description	268
9.65.2	Typedef Documentation	269
9.65.2.1	<code>l4_cap_idx_t</code>	269
9.65.3	Enumeration Type Documentation	269
9.65.3.1	<code>l4_cap_consts_t</code>	269
9.65.3.2	<code>l4_default_caps_t</code>	269
9.65.4	Function Documentation	270
9.65.4.1	<code>cap_cast</code>	270
9.65.4.2	<code>cap_reinterpret_cast</code>	270
9.65.4.3	<code>cap_dynamic_cast</code>	271
9.65.4.4	<code>l4_is_invalid_cap</code>	272
9.65.4.5	<code>l4_is_valid_cap</code>	272
9.65.4.6	<code>l4_capability_equal</code>	273
9.66	Virtual Registers (UTCBs)	273
9.66.1	Detailed Description	274
9.66.2	Typedef Documentation	275
9.66.2.1	<code>l4_utcb_t</code>	275
9.66.3	Function Documentation	275
9.66.3.1	<code>l4_utcb_mr</code>	275

9.66.3.2	l4_utcb_br	276
9.66.3.3	l4_utcb_tcr	276
9.67	Message Registers (MRs)	277
9.68	Buffer Registers (BRs)	277
9.68.1	Enumeration Type Documentation	278
9.68.1.1	l4_buffer_desc_consts_t	278
9.69	Thread Control Registers (TCRs)	278
9.70	Exception registers	279
9.70.1	Detailed Description	280
9.70.2	Function Documentation	280
9.70.2.1	l4_utcb_exc	280
9.70.2.2	l4_utcb_exc_pc	280
9.70.2.3	l4_utcb_exc_pc_set	281
9.70.2.4	l4_utcb_exc_is_pf	281
9.71	Virtual Console	281
9.71.1	Detailed Description	282
9.71.2	Enumeration Type Documentation	283
9.71.2.1	L4_vcon_write_consts	283
9.71.2.2	L4_vcon_i_flags	283
9.71.2.3	L4_vcon_o_flags	283
9.71.2.4	L4_vcon_l_flags	283
9.71.2.5	L4_vcon_ops	284
9.71.3	Function Documentation	284
9.71.3.1	l4_vcon_send	284
9.71.3.2	l4_vcon_write	284
9.71.3.3	l4_vcon_read	285
9.71.3.4	l4_vcon_set_attr	286
9.71.3.5	l4_vcon_get_attr	286
9.72	vCPU API	287
9.72.1	Detailed Description	288
9.72.2	Enumeration Type Documentation	288
9.72.2.1	L4_vcpu_state_flags	288
9.72.2.2	L4_vcpu_sticky_flags	289
9.72.2.3	L4_vcpu_state_offset	289
9.73	Fiasco-UX Virtual devices	289
9.73.1	Detailed Description	290

9.73.2 Enumeration Type Documentation	290
9.73.2.1 l4_vhw_entry_type	290
9.74 Memory operations	290
9.74.1 Detailed Description	291
9.74.2 Enumeration Type Documentation	291
9.74.2.1 L4_mem_op_widths	291
9.74.3 Function Documentation	291
9.74.3.1 l4_mem_read	291
9.74.3.2 l4_mem_write	291
9.75 ARM Virtual Registers (UTCB)	292
9.76 VM API for TZ	292
9.76.1 Detailed Description	293
9.76.2 Function Documentation	293
9.76.2.1 l4_vm_run	293
9.77 amd64 Virtual Registers (UTCB)	294
9.78 x86 Virtual Registers (UTCB)	295
9.78.1 Enumeration Type Documentation	295
9.78.1.1 L4_utcb_consts_x86	295
9.79 CPU related functions	296
9.79.1 Function Documentation	296
9.79.1.1 l4util_cpu_has_cpuid	296
9.79.1.2 l4util_cpu_capabilities	297
9.79.1.3 l4util_cpu_capabilities_nocheck	297
9.80 Functions to manipulate the local IDT	298
9.81 Timestamp Counter	299
9.81.1 Function Documentation	300
9.81.1.1 l4_rdtsc	300
9.81.1.2 l4_rdtsc_32	301
9.81.1.3 l4_rdpmc	301
9.81.1.4 l4_rdpmc_32	301
9.81.1.5 l4_tsc_to_ns	301
9.81.1.6 l4_tsc_to_us	302
9.81.1.7 l4_tsc_to_s_and_ns	302
9.81.1.8 l4_ns_to_tsc	302
9.81.1.9 l4_busy_wait_ns	303
9.81.1.10 l4_busy_wait_us	303

9.81.1.11 <code>l4_calibrate_tsc</code>	304
9.81.1.12 <code>l4_tsc_init</code>	305
9.81.1.13 <code>l4_get_hz</code>	306
9.82 Atomic Instructions	306
9.82.1 Function Documentation	308
9.82.1.1 <code>l4util_cmpxchg64</code>	308
9.82.1.2 <code>l4util_cmpxchg32</code>	309
9.82.1.3 <code>l4util_cmpxchg16</code>	309
9.82.1.4 <code>l4util_cmpxchg8</code>	309
9.82.1.5 <code>l4util_cmpxchg</code>	310
9.82.1.6 <code>l4util_xchg32</code>	310
9.82.1.7 <code>l4util_xchg16</code>	311
9.82.1.8 <code>l4util_xchg8</code>	311
9.82.1.9 <code>l4util_xchg</code>	311
9.82.1.10 <code>l4util_add8</code>	311
9.82.1.11 <code>l4util_add8_res</code>	312
9.82.1.12 <code>l4util_inc8</code>	312
9.82.1.13 <code>l4util_inc8_res</code>	312
9.82.1.14 <code>l4util_atomic_add</code>	312
9.82.1.15 <code>l4util_atomic_inc</code>	312
9.83 Internal functions	313
9.84 Bit Manipulation	313
9.84.1 Function Documentation	314
9.84.1.1 <code>l4util_set_bit</code>	314
9.84.1.2 <code>l4util_clear_bit</code>	315
9.84.1.3 <code>l4util_complement_bit</code>	315
9.84.1.4 <code>l4util_test_bit</code>	316
9.84.1.5 <code>l4util_bts</code>	316
9.84.1.6 <code>l4util_btr</code>	316
9.84.1.7 <code>l4util_btc</code>	317
9.84.1.8 <code>l4util_bsr</code>	317
9.84.1.9 <code>l4util_bsf</code>	318
9.84.1.10 <code>l4util_find_first_set_bit</code>	318
9.84.1.11 <code>l4util_find_first_zero_bit</code>	319
9.84.1.12 <code>l4util_next_power2</code>	319
9.85 ELF binary format	319

9.85.1	Detailed Description	336
9.85.2	Define Documentation	336
9.85.2.1	EL_CLASS	336
9.85.2.2	EL_CLASS	336
9.85.2.3	ELFCLASSNONE	336
9.85.2.4	ELFCLASSNONE	336
9.85.2.5	EL_DATA	337
9.85.2.6	EL_DATA	337
9.85.2.7	ELFDATANONE	337
9.85.2.8	ELFDATANONE	337
9.85.2.9	ELFDATA2LSB	337
9.85.2.10	ELFDATA2LSB	337
9.85.2.11	ELFDATA2MSB	337
9.85.2.12	ELFDATA2MSB	338
9.85.2.13	EL_VERSION	338
9.85.2.14	EL_VERSION	338
9.85.2.15	EL_OSABI	338
9.85.2.16	EL_OSABI	338
9.85.2.17	ELFOSABI_SYSV	338
9.85.2.18	ELFOSABI_SYSV	338
9.85.2.19	ELFOSABI_HPUX	339
9.85.2.20	ELFOSABI_HPUX	339
9.85.2.21	ELFOSABI_NETBSD	339
9.85.2.22	ELFOSABI_LINUX	339
9.85.2.23	ELFOSABI_SOLARIS	339
9.85.2.24	ELFOSABI_AIX	339
9.85.2.25	ELFOSABI_IRIX	339
9.85.2.26	ELFOSABI_FREEBSD	339
9.85.2.27	ELFOSABI_TRU64	340
9.85.2.28	ELFOSABI_MODESTO	340
9.85.2.29	ELFOSABI_OPENBSD	340
9.85.2.30	EL_PAD	340
9.85.2.31	EL_PAD	340
9.85.2.32	EM_ARC	340
9.85.2.33	SHT_NUM	340
9.85.2.34	SHF_GROUP	340

9.85.2.35 SHF_TLS	341
9.85.2.36 SHF_MASKOS	341
9.85.2.37 PT_LOOS	341
9.85.2.38 PT_HIOS	341
9.85.2.39 PT_LOPROC	341
9.85.2.40 PT_HIPROC	341
9.85.2.41 PT_GNU_EH_FRAME	341
9.85.2.42 PT_GNU_STACK	341
9.85.2.43 PT_GNU_RELRO	341
9.85.2.44 PT_L4_STACK	342
9.85.2.45 PT_L4_KIP	342
9.85.2.46 PT_L4_AUX	342
9.85.2.47 NT_VERSION	342
9.85.2.48 DT_NULL	342
9.85.2.49 DT_LOPROC	342
9.85.2.50 DT_HIPROC	342
9.85.2.51 DF_1_NOW	342
9.85.2.52 DF_1_GLOBAL	343
9.85.2.53 DF_1_GROUP	343
9.85.2.54 DF_1_NODELETE	343
9.85.2.55 DF_1_LOADFLTR	343
9.85.2.56 DF_1_NOOPEN	343
9.85.2.57 DF_1_ORIGIN	343
9.85.2.58 DF_1_DIRECT	343
9.85.2.59 DF_1_INTERPOSE	343
9.85.2.60 DF_1_NODEFLIB	343
9.85.2.61 DF_1_NODUMP	344
9.85.2.62 DF_1_CONFALT	344
9.85.2.63 DF_1_ENDFILTEE	344
9.85.2.64 DF_1_DISPRLDNE	344
9.85.2.65 DF_1_DISPRLPND	344
9.85.2.66 DF_P1_LAZYLOAD	344
9.85.2.67 DF_P1_GROUPPERM	344
9.86 Kernel Interface Page API	345
9.86.1 Define Documentation	345
9.86.1.1 l4util_kip_for_each_feature	345

9.86.2 Function Documentation	346
9.86.2.1 l4util_kip_kernel_is_ux	346
9.86.2.2 l4util_kip_kernel_has_feature	346
9.86.2.3 l4util_kip_kernel_abi_version	346
9.86.2.4 l4util_memdesc_vm_high	346
9.87 Comfortable Command Line Parsing	347
9.87.1 Function Documentation	347
9.87.1.1 parse_cmdline	347
9.88 Priority related functions	349
9.89 Random number support	349
9.89.1 Function Documentation	350
9.89.1.1 l4util_rand	350
9.89.1.2 l4util_srand	350
9.90 Machine Restarting Function	350
9.91 Low-Level Thread Functions	351
9.92 Utility Functions	353
9.92.1 Function Documentation	355
9.92.1.1 l4_sleep_forever	355
9.92.1.2 l4util_splitlog2_hdl	355
9.92.1.3 l4util_splitlog2_size	356
9.92.1.4 l4util_micros2l4to	357
9.93 IA32 Port I/O API	357
9.93.1 Function Documentation	358
9.93.1.1 l4util_in8	358
9.93.1.2 l4util_in16	359
9.93.1.3 l4util_in32	359
9.93.1.4 l4util_ins8	359
9.93.1.5 l4util_ins16	359
9.93.1.6 l4util_ins32	360
9.93.1.7 l4util_out8	360
9.93.1.8 l4util_out16	360
9.93.1.9 l4util_out32	360
9.93.1.10 l4util_outs8	361
9.93.1.11 l4util_outs16	361
9.93.1.12 l4util_outs32	361
9.94 Bitmap graphics and fonts	361

9.94.1	Detailed Description	362
9.95	Functions for rendering bitmap data in frame buffers	362
9.95.1	Typedef Documentation	363
9.95.1.1	gfxbitmap_color_t	363
9.95.1.2	gfxbitmap_color_pix_t	363
9.95.2	Function Documentation	363
9.95.2.1	gfxbitmap_convert_color	363
9.95.2.2	gfxbitmap_fill	364
9.95.2.3	gfxbitmap_bmap	364
9.95.2.4	gfxbitmap_set	364
9.95.2.5	gfxbitmap_copy	365
9.96	Functions for rendering bitmap fonts to frame buffers	365
9.96.1	Enumeration Type Documentation	366
9.96.1.1	"@97	366
9.96.2	Function Documentation	367
9.96.2.1	gfxbitmap_font_init	367
9.96.2.2	gfxbitmap_font_get	367
9.96.2.3	gfxbitmap_font_width	367
9.96.2.4	gfxbitmap_font_height	367
9.96.2.5	gfxbitmap_font_data	368
9.96.2.6	gfxbitmap_font_text	368
9.96.2.7	gfxbitmap_font_text_scale	368
9.97	IO interface	369
9.97.1	Typedef Documentation	370
9.97.1.1	l4io_resource_t	370
9.97.2	Enumeration Type Documentation	370
9.97.2.1	l4io_iomem_flags_t	370
9.97.2.2	l4io_device_types_t	371
9.97.2.3	l4io_resource_types_t	371
9.97.3	Function Documentation	371
9.97.3.1	l4io_request_iomem	371
9.97.3.2	l4io_request_iomem_region	372
9.97.3.3	l4io_release_iomem	372
9.97.3.4	l4io_search_iomem_region	372
9.97.3.5	l4io_request_ioport	373
9.97.3.6	l4io_release_ioport	373

9.97.3.7 l4io_lookup_device	373
9.97.3.8 l4io_lookup_resource	374
9.97.3.9 l4io_request_resource_iomem	374
9.97.3.10 l4io_has_resource	374
9.98 IRQ handling library	375
9.99 Interface using direct functionality.	375
9.99.1 Function Documentation	376
9.99.1.1 l4irq_attach	376
9.99.1.2 l4irq_attach_ft	376
9.99.1.3 l4irq_attach_thread	377
9.99.1.4 l4irq_attach_thread_ft	377
9.99.1.5 l4irq_wait	377
9.99.1.6 l4irq_unmask_and_wait_any	378
9.99.1.7 l4irq_wait_any	378
9.99.1.8 l4irq_unmask	378
9.99.1.9 l4irq_detach	379
9.100 Interface for asynchronous ISR handlers.	379
9.100.1 Detailed Description	379
9.100.2 Function Documentation	380
9.100.2.1 l4irq_request	380
9.100.2.2 l4irq_release	380
9.101 Interface using direct functionality.	380
9.101.1 Function Documentation	381
9.101.1.1 l4irq_attach_cap	381
9.101.1.2 l4irq_attach_cap_ft	381
9.101.1.3 l4irq_attach_thread_cap	382
9.101.1.4 l4irq_attach_thread_cap_ft	382
9.102 Interface for asynchronous ISR handlers with a given IRQ capability.	382
9.102.1 Detailed Description	383
9.102.2 Function Documentation	383
9.102.2.1 l4irq_request_cap	383
9.103 Sigma0 API	383
9.103.1 Detailed Description	385
9.103.2 Enumeration Type Documentation	385
9.103.2.1 l4sigma0_return_flags_t	385
9.103.3 Function Documentation	385

9.103.3.1 l4sigma0_map_kip	385
9.103.3.2 l4sigma0_map_mem	385
9.103.3.3 l4sigma0_map_iomem	386
9.103.3.4 l4sigma0_map_anypage	386
9.103.3.5 l4sigma0_map_tbuf	387
9.103.3.6 l4sigma0_debug_dump	387
9.103.3.7 l4sigma0_new_client	387
9.103.3.8 l4sigma0_map_errstr	387
9.104 Internal constants	388
9.104.1 Detailed Description	389
9.105 vCPU Support Library	389
9.105.1 Detailed Description	391
9.105.2 Enumeration Type Documentation	391
9.105.2.1 l4vcpu_irq_state_t	391
9.105.3 Function Documentation	391
9.105.3.1 l4vcpu_state	391
9.105.3.2 l4vcpu_irq_disable	392
9.105.3.3 l4vcpu_irq_disable_save	393
9.105.3.4 l4vcpu_irq_enable	393
9.105.3.5 l4vcpu_irq_restore	394
9.105.3.6 l4vcpu_halt	395
9.105.3.7 l4vcpu_print_state	396
9.105.3.8 l4vcpu_is_irq_entry	396
9.105.3.9 l4vcpu_is_page_fault_entry	396
9.106 Extended vCPU support	396
9.106.1 Detailed Description	397
9.106.2 Function Documentation	397
9.106.2.1 l4vcpu_ext_alloc	397
9.107 Shared Memory Library	397
9.107.1 Detailed Description	398
9.107.2 Function Documentation	399
9.107.2.1 l4shmc_create	399
9.107.2.2 l4shmc_attach	399
9.107.2.3 l4shmc_attach_to	399
9.107.2.4 l4shmc_connect_chunk_signal	400
9.107.2.5 l4shmc_area_size	400

9.107.2.6 l4shmc_area_size_free	400
9.107.2.7 l4shmc_area_overhead	400
9.107.2.8 l4shmc_chunk_overhead	401
9.108Chunks	401
9.108.1 Function Documentation	402
9.108.1.1 l4shmc_add_chunk	402
9.108.1.2 l4shmc_get_chunk	402
9.108.1.3 l4shmc_get_chunk_to	403
9.108.1.4 l4shmc_iterate_chunk	403
9.108.1.5 l4shmc_chunk_ptr	403
9.108.1.6 l4shmc_chunk_capacity	404
9.108.1.7 l4shmc_chunk_signal	404
9.109Producer	404
9.109.1 Function Documentation	405
9.109.1.1 l4shmc_chunk_try_to_take	405
9.109.1.2 l4shmc_chunk_ready	405
9.109.1.3 l4shmc_chunk_ready_sig	405
9.109.1.4 l4shmc_is_chunk_clear	406
9.110Consumer	406
9.110.1 Function Documentation	407
9.110.1.1 l4shmc_enable_chunk	407
9.110.1.2 l4shmc_wait_chunk	407
9.110.1.3 l4shmc_wait_chunk_to	407
9.110.1.4 l4shmc_wait_chunk_try	407
9.110.1.5 l4shmc_chunk_consumed	408
9.110.1.6 l4shmc_is_chunk_ready	408
9.110.1.7 l4shmc_chunk_size	408
9.111Signals	409
9.111.1 Function Documentation	410
9.111.1.1 l4shmc_add_signal	410
9.111.1.2 l4shmc_attach_signal	410
9.111.1.3 l4shmc_attach_signal_to	410
9.111.1.4 l4shmc_get_signal_to	411
9.111.1.5 l4shmc_signal_cap	411
9.111.1.6 l4shmc_check_magic	411
9.112Producer	412

9.112.1 Function Documentation	412
9.112.1.1 l4shmc_trigger	412
9.113 Consumer	412
9.113.1 Function Documentation	413
9.113.1.1 l4shmc_enable_signal	413
9.113.1.2 l4shmc_wait_any	413
9.113.1.3 l4shmc_wait_any_try	414
9.113.1.4 l4shmc_wait_any_to	414
9.113.1.5 l4shmc_wait_signal	414
9.113.1.6 l4shmc_wait_signal_to	415
9.113.1.7 l4shmc_wait_signal_try	415
9.114 Integer Types	415
9.114.1 Detailed Description	417
9.114.2 Typedef Documentation	417
9.114.2.1 l4_int8_t	417
9.114.2.2 l4_uint8_t	417
9.114.2.3 l4_int16_t	418
9.114.2.4 l4_uint16_t	418
9.114.2.5 l4_int32_t	418
9.114.2.6 l4_uint32_t	418
9.114.2.7 l4_int64_t	418
9.114.2.8 l4_uint64_t	418
10 Namespace Documentation	419
10.1 cxx Namespace Reference	419
10.1.1 Detailed Description	421
10.2 cxx::Bits Namespace Reference	421
10.2.1 Detailed Description	421
10.3 L4 Namespace Reference	421
10.3.1 Detailed Description	424
10.3.2 Function Documentation	424
10.3.2.1 kobject_typeid	424
10.4 L4::Ipc_svr Namespace Reference	425
10.4.1 Detailed Description	425
10.4.2 Enumeration Type Documentation	425
10.4.2.1 Reply_mode	425
10.5 L4Re Namespace Reference	426

10.5.1	Detailed Description	427
10.6	L4Re::Vfs Namespace Reference	427
10.6.1	Detailed Description	428
11	Data Structure Documentation	429
11.1	L4::Alloc_list Class Reference	429
11.1.1	Detailed Description	429
11.2	L4::Thread::Attr Class Reference	429
11.2.1	Detailed Description	430
11.2.2	Constructor & Destructor Documentation	430
11.2.2.1	Attr	430
11.2.3	Member Function Documentation	430
11.2.3.1	pager	430
11.2.3.2	pager	431
11.2.3.3	exc_handler	431
11.2.3.4	exc_handler	432
11.2.3.5	bind	432
11.2.3.6	ux_host_syscall	433
11.3	L4Re::Util::Auto_cap< T > Struct Template Reference	433
11.3.1	Detailed Description	434
11.4	L4Re::Util::Auto_del_cap< T > Struct Template Reference	434
11.4.1	Detailed Description	435
11.5	cxx::Auto_ptr< T > Class Template Reference	436
11.5.1	Detailed Description	437
11.5.2	Member Typedef Documentation	437
11.5.2.1	Ref_type	437
11.5.3	Constructor & Destructor Documentation	437
11.5.3.1	Auto_ptr	437
11.5.3.2	Auto_ptr	437
11.5.3.3	~Auto_ptr	437
11.5.4	Member Function Documentation	438
11.5.4.1	operator=	438
11.5.4.2	operator*	438
11.5.4.3	operator->	438
11.5.4.4	get	438
11.5.4.5	release	439
11.5.4.6	operator Priv_type *	439

11.6 <code>cxx::Avl_map< Key, Data, Compare, Alloc ></code> Class Template Reference	439
11.6.1 Detailed Description	443
11.6.2 Constructor & Destructor Documentation	443
11.6.2.1 <code>Avl_map</code>	443
11.6.3 Member Function Documentation	443
11.6.3.1 <code>insert</code>	443
11.6.3.2 <code>find_node</code>	444
11.6.3.3 <code>lower_bound_node</code>	444
11.6.3.4 <code>find</code>	444
11.6.3.5 <code>remove</code>	445
11.6.3.6 <code>erase</code>	445
11.6.3.7 <code>operator[]</code>	445
11.6.3.8 <code>operator[]</code>	445
11.7 <code>cxx::Avl_set< Item, Compare, Alloc ></code> Class Template Reference	446
11.7.1 Detailed Description	450
11.7.2 Constructor & Destructor Documentation	450
11.7.2.1 <code>Avl_set</code>	450
11.7.2.2 <code>Avl_set</code>	451
11.7.3 Member Function Documentation	451
11.7.3.1 <code>insert</code>	451
11.7.3.2 <code>remove</code>	452
11.7.3.3 <code>find_node</code>	452
11.7.3.4 <code>lower_bound_node</code>	453
11.7.3.5 <code>begin</code>	453
11.7.3.6 <code>end</code>	453
11.7.3.7 <code>begin</code>	454
11.7.3.8 <code>end</code>	454
11.7.3.9 <code>rbegin</code>	454
11.7.3.10 <code>rend</code>	455
11.7.3.11 <code>rbegin</code>	455
11.7.3.12 <code>rend</code>	455
11.8 <code>cxx::Avl_tree< Node, Get_key, Compare ></code> Class Template Reference	455
11.8.1 Detailed Description	458
11.8.2 Member Typedef Documentation	459
11.8.2.1 <code>Iterator</code>	459
11.8.3 Member Function Documentation	459

11.8.3.1 <code>insert</code>	459
11.8.3.2 <code>remove</code>	460
11.9 <code>cxx::Avl_tree_node</code> Class Reference	461
11.9.1 Detailed Description	464
11.10L4:: <code>Base_exception</code> Class Reference	464
11.10.1 Detailed Description	465
11.11 <code>cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc ></code> Class Template Reference	466
11.11.1 Detailed Description	468
11.11.2 Member Enumeration Documentation	468
11.11.2.1 <code>"@48</code>	468
11.11.3 Member Function Documentation	469
11.11.3.1 <code>total_objects</code>	469
11.11.3.2 <code>free_objects</code>	469
11.12 <code>cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc ></code> Class Template Reference	470
11.12.1 Detailed Description	472
11.12.2 Member Enumeration Documentation	472
11.12.2.1 <code>"@49</code>	472
11.12.3 Member Function Documentation	473
11.12.3.1 <code>alloc</code>	473
11.12.3.2 <code>free</code>	473
11.12.3.3 <code>total_objects</code>	473
11.12.3.4 <code>free_objects</code>	473
11.13L4:: <code>Basic_registry</code> Class Reference	474
11.13.1 Detailed Description	474
11.13.2 Member Typedef Documentation	474
11.13.2.1 <code>Value</code>	474
11.13.3 Member Function Documentation	475
11.13.3.1 <code>dispatch</code>	475
11.14L4Re:: <code>Vfs::Be_file</code> Class Reference	475
11.14.1 Detailed Description	479
11.15L4Re:: <code>Vfs::Be_file_system</code> Class Reference	479
11.15.1 Detailed Description	481
11.15.2 Constructor & Destructor Documentation	482
11.15.2.1 <code>Be_file_system</code>	482
11.15.2.2 <code>~Be_file_system</code>	482
11.15.3 Member Function Documentation	482

11.15.3.1 <code>type</code>	482
11.16 <code>cxx::Bitmap< BITS ></code> Class Template Reference	482
11.16.1 Detailed Description	485
11.16.2 Constructor & Destructor Documentation	485
11.16.2.1 <code>Bitmap</code>	485
11.16.3 Member Function Documentation	485
11.16.3.1 <code>clear_all</code>	485
11.17 <code>cxx::Bitmap_base</code> Class Reference	485
11.17.1 Detailed Description	487
11.17.2 Member Function Documentation	487
11.17.2.1 <code>words</code>	487
11.17.2.2 <code>chars</code>	487
11.17.2.3 <code>bit</code>	487
11.17.2.4 <code>clear_bit</code>	488
11.17.2.5 <code>set_bit</code>	488
11.17.2.6 <code>operator[]</code>	489
11.17.2.7 <code>scan_zero</code>	489
11.18 <code>L4::Bounds_error</code> Class Reference	490
11.18.1 Detailed Description	493
11.19 <code>cxx::Bits::Bst< Node, Get_key, Compare ></code> Class Template Reference	493
11.19.1 Detailed Description	497
11.19.2 Member Function Documentation	497
11.19.2.1 <code>dir</code>	497
11.19.2.2 <code>dir</code>	497
11.19.2.3 <code>begin</code>	498
11.19.2.4 <code>end</code>	498
11.19.2.5 <code>begin</code>	499
11.19.2.6 <code>end</code>	499
11.19.2.7 <code>rbegin</code>	499
11.19.2.8 <code>rend</code>	500
11.19.2.9 <code>rbegin</code>	500
11.19.2.10 <code>rend</code>	500
11.19.2.11 <code>lfind_node</code>	501
11.19.2.12 <code>lower_bound_node</code>	501
11.19.2.13 <code>find</code>	502
11.20 <code>cxx::Bits::Bst_node</code> Class Reference	503

11.20.1 Detailed Description	506
11.21L4::Ipc::Buf_cp_in< T > Class Template Reference	506
11.21.1 Detailed Description	506
11.21.2 Constructor & Destructor Documentation	506
11.21.2.1 Buf_cp_in	506
11.22L4::Ipc::Buf_cp_out< T > Class Template Reference	507
11.22.1 Detailed Description	507
11.22.2 Constructor & Destructor Documentation	507
11.22.2.1 Buf_cp_out	507
11.22.3 Member Function Documentation	508
11.22.3.1 size	508
11.22.3.2 buf	508
11.23L4::Ipc::Buf_in< T > Class Template Reference	509
11.23.1 Detailed Description	509
11.23.2 Constructor & Destructor Documentation	509
11.23.2.1 Buf_in	509
11.24L4::Cap< T > Class Template Reference	510
11.24.1 Detailed Description	512
11.24.2 Constructor & Destructor Documentation	512
11.24.2.1 Cap	512
11.24.2.2 Cap	513
11.24.2.3 Cap	513
11.24.3 Member Function Documentation	513
11.24.3.1 move	513
11.25L4Re::Cap_alloc Class Reference	513
11.25.1 Detailed Description	514
11.25.2 Member Function Documentation	514
11.25.2.1 alloc	514
11.25.2.2 alloc	515
11.25.2.3 free	515
11.25.2.4 get_cap_alloc	516
11.26L4Re::Util::Cap_alloc_base Class Reference	517
11.26.1 Detailed Description	519
11.27L4::Cap_base Class Reference	519
11.27.1 Detailed Description	521
11.27.2 Member Enumeration Documentation	521

11.27.2.1 No_init_type	521
11.27.2.2 Cap_type	521
11.27.3 Constructor & Destructor Documentation	521
11.27.3.1 Cap_base	521
11.27.3.2 Cap_base	522
11.27.4 Member Function Documentation	522
11.27.4.1 cap	522
11.27.4.2 is_valid	524
11.27.4.3 fpage	524
11.27.4.4 snd_base	525
11.27.4.5 validate	526
11.27.4.6 validate	526
11.27.5 Field Documentation	526
11.27.5.1 _c	526
11.28cxx::Bitmap_base::Char< BITS > Class Template Reference	527
11.28.1 Detailed Description	527
11.29L4Re::Video::Color_component Class Reference	527
11.29.1 Detailed Description	528
11.29.2 Constructor & Destructor Documentation	528
11.29.2.1 Color_component	528
11.29.3 Member Function Documentation	528
11.29.3.1 size	528
11.29.3.2 shift	528
11.29.3.3 operator==	529
11.29.3.4 get	529
11.29.3.5 set	529
11.29.3.6 dump	529
11.30L4::Com_error Class Reference	530
11.30.1 Detailed Description	533
11.30.2 Constructor & Destructor Documentation	533
11.30.2.1 Com_error	533
11.31L4::Ipc_svr::Compound_reply Struct Reference	533
11.31.1 Detailed Description	534
11.32L4Re::Console Class Reference	534
11.32.1 Detailed Description	536
11.33L4Re::Util::Counting_cap_alloc< COUNTERTYPE > Class Template Reference	536

11.33.1 Detailed Description	537
11.34L4Re::Dataspace Class Reference	537
11.34.1 Detailed Description	540
11.34.2 Member Enumeration Documentation	541
11.34.2.1 Map_flags	541
11.34.3 Member Function Documentation	541
11.34.3.1 map	541
11.34.3.2 map_region	542
11.34.3.3 clear	543
11.34.3.4 allocate	543
11.34.3.5 copy_in	544
11.34.3.6 phys	545
11.34.3.7 size	545
11.34.3.8 flags	546
11.34.3.9 info	546
11.35L4Re::Util::Dataspace_svr Class Reference	548
11.35.1 Detailed Description	548
11.35.2 Member Function Documentation	548
11.35.2.1 map	548
11.35.2.2 map_hook	549
11.35.2.3 phys	549
11.35.2.4 take	550
11.35.2.5 release	550
11.35.2.6 copy	550
11.35.2.7 clear	550
11.35.2.8 page_shift	551
11.36L4Re::Debug_obj Class Reference	551
11.36.1 Detailed Description	553
11.36.2 Member Function Documentation	554
11.36.2.1 debug	554
11.37L4::Debugger Class Reference	554
11.37.1 Detailed Description	557
11.37.2 Member Function Documentation	557
11.37.2.1 set_object_name	557
11.37.2.2 global_id	557
11.37.2.3 kobj_to_id	557

11.37.2.4 <code>query_log_typeid</code>	558
11.37.2.5 <code>query_log_name</code>	558
11.37.2.6 <code>switch_log</code>	558
11.37.2.7 <code>get_object_name</code>	559
11.38L4:: <code>Ipc_svr::Default_loop_hooks</code> Struct Reference	559
11.38.1 Detailed Description	559
11.39L4:: <code>Ipc_svr::Default_setup_wait</code> Struct Reference	560
11.39.1 Detailed Description	560
11.40L4:: <code>Ipc_svr::Default_timeout</code> Struct Reference	560
11.40.1 Detailed Description	561
11.41cxx:: <code>Bits::Direction</code> Struct Reference	561
11.41.1 Detailed Description	562
11.41.2 Member Enumeration Documentation	562
11.41.2.1 <code>Direction_e</code>	562
11.41.3 Member Function Documentation	562
11.41.3.1 <code>operator!</code>	562
11.42L4Re:: <code>Vfs::Directory</code> Class Reference	563
11.42.1 Detailed Description	565
11.42.2 Member Function Documentation	565
11.42.2.1 <code>faccessat</code>	565
11.42.2.2 <code>mkdir</code>	566
11.42.2.3 <code>unlink</code>	566
11.42.2.4 <code>rename</code>	566
11.42.2.5 <code>link</code>	567
11.42.2.6 <code>symlink</code>	567
11.42.2.7 <code>rmdir</code>	567
11.43L4:: <code>Element_already_exists</code> Class Reference	568
11.43.1 Detailed Description	571
11.44L4:: <code>Element_not_found</code> Class Reference	571
11.44.1 Detailed Description	574
11.45Elf32_Dyn Struct Reference	574
11.45.1 Detailed Description	574
11.45.2 Field Documentation	574
11.45.2.1 <code>d_val</code>	574
11.46Elf32_Ehdr Struct Reference	574
11.46.1 Detailed Description	575

11.46.2 Field Documentation	576
11.46.2.1 e_phnum	576
11.46.2.2 e_shnum	576
11.47 Elf32_Phdr Struct Reference	576
11.47.1 Detailed Description	577
11.48 Elf32_Shdr Struct Reference	577
11.48.1 Detailed Description	578
11.49 Elf32_Sym Struct Reference	578
11.49.1 Detailed Description	578
11.50 Elf64_Dyn Struct Reference	578
11.50.1 Detailed Description	579
11.50.2 Field Documentation	579
11.50.2.1 d_val	579
11.51 Elf64_Ehdr Struct Reference	579
11.51.1 Detailed Description	580
11.51.2 Field Documentation	580
11.51.2.1 e_phnum	580
11.51.2.2 e_shnum	580
11.52 Elf64_Phdr Struct Reference	581
11.52.1 Detailed Description	581
11.53 Elf64_Shdr Struct Reference	581
11.53.1 Detailed Description	582
11.54 Elf64_Sym Struct Reference	582
11.54.1 Detailed Description	583
11.55 L4Re::Env Class Reference	583
11.55.1 Detailed Description	587
11.55.2 Member Function Documentation	587
11.55.2.1 env	587
11.55.2.2 parent	587
11.55.2.3 mem_alloc	587
11.55.2.4 rm	588
11.55.2.5 log	588
11.55.2.6 main_thread	588
11.55.2.7 task	588
11.55.2.8 factory	588
11.55.2.9 first_free_cap	589

11.55.2.10	utcb_area	589
11.55.2.11	first_free_utcb	589
11.55.2.12	initial_caps	589
11.55.2.13	get	589
11.55.2.14	get_cap	590
11.55.2.15	get_cap	590
11.55.2.16	parent	591
11.55.2.17	mem_alloc	591
11.55.2.18	rm	591
11.55.2.19	log	591
11.55.2.20	main_thread	591
11.55.2.21	factory	592
11.55.2.22	first_free_cap	592
11.55.2.23	utcb_area	592
11.55.2.24	first_free_utcb	592
11.55.2.25	scheduler	592
11.55.2.26	scheduler	593
11.55.2.27	initial_caps	593
11.56	L4Re::Event Class Reference	593
11.56.1	Detailed Description	596
11.56.2	Member Function Documentation	596
11.56.2.1	get_buffer	596
11.57	L4Re::Event_buffer_t< PAYLOAD >::Event Struct Reference	596
11.57.1	Detailed Description	597
11.58	L4Re::Util::Event_buffer_consumer_t< PAYLOAD > Class Template Reference	597
11.58.1	Detailed Description	600
11.58.2	Member Function Documentation	600
11.58.2.1	foreach_available_event	600
11.58.2.2	process	601
11.59	L4Re::Event_buffer_t< PAYLOAD > Class Template Reference	601
11.59.1	Detailed Description	604
11.59.2	Constructor & Destructor Documentation	604
11.59.2.1	Event_buffer_t	604
11.59.3	Member Function Documentation	604
11.59.3.1	next	604
11.59.3.2	put	604

11.60L4Re::Util::Event_buffer_t< PAYLOAD > Class Template Reference	605
11.60.1 Detailed Description	608
11.60.2 Member Function Documentation	608
11.60.2.1 buf	608
11.60.2.2 attach	608
11.60.2.3 detach	609
11.61L4Re::Util::Event_t< PAYLOAD > Class Template Reference	609
11.61.1 Detailed Description	611
11.61.2 Member Enumeration Documentation	611
11.61.2.1 Mode	611
11.61.3 Member Function Documentation	611
11.61.3.1 init	611
11.61.3.2 buffer	612
11.61.3.3 irq	612
11.62L4::Exception_tracer Class Reference	612
11.62.1 Detailed Description	614
11.63L4::Factory Class Reference	614
11.63.1 Detailed Description	617
11.63.2 Member Function Documentation	617
11.63.2.1 create	617
11.63.2.2 create_task	618
11.63.2.3 create_thread	619
11.63.2.4 create_factory	620
11.63.2.5 create_gate	620
11.63.2.6 create_irq	621
11.63.2.7 create_vm	622
11.64L4Re::Vfs::File Class Reference	623
11.64.1 Detailed Description	625
11.65L4Re::Vfs::File_system Class Reference	625
11.65.1 Detailed Description	627
11.65.2 Member Function Documentation	628
11.65.2.1 type	628
11.65.2.2 mount	628
11.66L4Re::Vfs::Fs Class Reference	629
11.66.1 Detailed Description	630
11.66.2 Member Function Documentation	630

11.66.2.1 <code>get_file</code>	630
11.66.2.2 <code>alloc_fd</code>	631
11.66.2.3 <code>set_fd</code>	631
11.66.2.4 <code>free_fd</code>	631
11.66.2.5 <code>mount</code>	631
11.67L4Re::Vfs::Generic_file Class Reference	632
11.67.1 Detailed Description	634
11.67.2 Member Function Documentation	634
11.67.2.1 <code>unlock_all_locks</code>	634
11.67.2.2 <code>fstat64</code>	635
11.67.2.3 <code>fchmod</code>	635
11.67.2.4 <code>get_status_flags</code>	635
11.67.2.5 <code>set_status_flags</code>	635
11.68gfxbitmap_offset Struct Reference	636
11.68.1 Detailed Description	636
11.69L4Re::Video::Goos Class Reference	636
11.69.1 Detailed Description	639
11.69.2 Member Enumeration Documentation	639
11.69.2.1 Flags	639
11.69.3 Member Function Documentation	640
11.69.3.1 <code>info</code>	640
11.69.3.2 <code>get_static_buffer</code>	640
11.69.3.3 <code>create_buffer</code>	640
11.69.3.4 <code>delete_buffer</code>	641
11.69.3.5 <code>create_view</code>	641
11.69.3.6 <code>delete_view</code>	641
11.69.3.7 <code>view</code>	641
11.70L4Re::Util::Video::Goos_svr Class Reference	642
11.70.1 Detailed Description	644
11.70.2 Member Function Documentation	644
11.70.2.1 <code>get_fb</code>	644
11.70.2.2 <code>screen_info</code>	645
11.70.2.3 <code>view_info</code>	645
11.70.2.4 <code>refresh</code>	645
11.70.2.5 <code>dispatch</code>	646
11.70.2.6 <code>init_infos</code>	646

11.71L4::Icu Class Reference	646
11.71.1 Detailed Description	649
11.71.2 Member Function Documentation	649
11.71.2.1 bind	649
11.71.2.2 unbind	650
11.71.2.3 info	651
11.71.2.4 msi_info	651
11.71.2.5 mask	652
11.71.2.6 unmask	653
11.71.2.7 set_mode	653
11.72L4::Ipc_svr::Ignore_errors Struct Reference	654
11.72.1 Detailed Description	655
11.73L4Re::Video::Goos::Info Struct Reference	655
11.73.1 Detailed Description	657
11.73.2 Member Function Documentation	657
11.73.2.1 auto_refresh	657
11.74L4Re::Video::View::Info Struct Reference	658
11.74.1 Detailed Description	661
11.74.2 Field Documentation	661
11.74.2.1 flags	661
11.75L4::Icu::Info Class Reference	661
11.75.1 Detailed Description	663
11.76L4::Invalid_capability Class Reference	663
11.76.1 Detailed Description	666
11.76.2 Constructor & Destructor Documentation	666
11.76.2.1 Invalid_capability	666
11.76.3 Member Function Documentation	666
11.76.3.1 cap	666
11.77L4::IOModifier Class Reference	666
11.77.1 Detailed Description	666
11.78L4::Ipc::Iostream Class Reference	667
11.78.1 Detailed Description	670
11.78.2 Constructor & Destructor Documentation	670
11.78.2.1 Iostream	670
11.78.3 Member Function Documentation	671
11.78.3.1 reset	671

11.78.3.2 call	671
11.78.3.3 reply_and_wait	672
11.78.3.4 reply_and_wait	673
11.79L4::Ipc_gate Class Reference	674
11.79.1 Detailed Description	676
11.79.2 Member Function Documentation	676
11.79.2.1 bind_thread	676
11.79.2.2 get_infos	676
11.80L4::Irq Class Reference	676
11.80.1 Detailed Description	679
11.80.2 Member Function Documentation	679
11.80.2.1 attach	679
11.80.2.2 chain	680
11.80.2.3 detach	681
11.80.2.4 receive	681
11.80.2.5 wait	682
11.80.2.6 unmask	683
11.80.2.7 trigger	683
11.81L4::Ipc::Istream Class Reference	684
11.81.1 Detailed Description	688
11.81.2 Constructor & Destructor Documentation	688
11.81.2.1 Istream	688
11.81.3 Member Function Documentation	688
11.81.3.1 reset	688
11.81.3.2 get	688
11.81.3.3 skip	689
11.81.3.4 get	689
11.81.3.5 get	689
11.81.3.6 tag	690
11.81.3.7 tag	690
11.81.3.8 wait	691
11.81.3.9 wait	691
11.81.3.10 receive	692
11.82L4Re::Util::Item_alloc_base Class Reference	693
11.82.1 Detailed Description	694
11.83cxx::List_item::Iter Class Reference	695

11.83.1 Detailed Description	698
11.83.2 Member Function Documentation	698
11.83.2.1 remove_me	698
11.84cxx::List< D, Alloc >::Iter Class Reference	699
11.84.1 Detailed Description	701
11.85L4::Kobject Class Reference	701
11.85.1 Detailed Description	702
11.85.2 Member Function Documentation	702
11.85.2.1 cap	702
11.85.2.2 dec_refcnt	702
11.85.3 Friends And Related Function Documentation	703
11.85.3.1 kobject_typeid	703
11.86L4::Kobject_2t< Derived, Base1, Base2, PROTO > Class Template Reference	704
11.86.1 Detailed Description	705
11.86.2 Friends And Related Function Documentation	705
11.86.2.1 kobject_typeid	705
11.87L4::Kobject_t< Derived, Base, PROTO > Class Template Reference	705
11.87.1 Detailed Description	706
11.87.2 Friends And Related Function Documentation	706
11.87.2.1 kobject_typeid	706
11.88I4_buf_regs_t Struct Reference	707
11.88.1 Detailed Description	707
11.89I4_exc_regs_t Struct Reference	707
11.89.1 Detailed Description	710
11.89.2 Field Documentation	710
11.89.2.1 flags	710
11.90I4_fpage_t Union Reference	710
11.90.1 Detailed Description	711
11.91I4_icu_info_t Struct Reference	711
11.91.1 Detailed Description	712
11.91.2 Field Documentation	712
11.91.2.1 features	712
11.92I4_kernel_info_mem_desc_t Struct Reference	712
11.92.1 Detailed Description	712
11.93I4_kernel_info_t Struct Reference	712
11.93.1 Detailed Description	715

11.94	14_msg_regs_t Struct Reference	715
11.94.1	Detailed Description	715
11.95	14_mshtag_t Struct Reference	715
11.95.1	Detailed Description	717
11.95.2	Member Function Documentation	717
11.95.2.1	flags	717
11.96	14_rt_preemption_t Union Reference	717
11.96.1	Detailed Description	718
11.97	14_rt_preemption_val32_t Struct Reference	718
11.97.1	Detailed Description	719
11.98	14_rt_preemption_val_t Struct Reference	719
11.98.1	Detailed Description	719
11.99	14_sched_cpu_set_t Struct Reference	720
11.99.1	Detailed Description	720
11.100	14_sched_param_t Struct Reference	720
11.100.1	Detailed Description	721
11.101	14_snd_fpage_t Struct Reference	722
11.101.1	Detailed Description	722
11.102	14_thread_regs_t Struct Reference	723
11.102.1	Detailed Description	724
11.103	14_timeout_s Struct Reference	724
11.103.1	Detailed Description	724
11.104	14_timeout_t Union Reference	724
11.104.1	Detailed Description	725
11.105	14_tracebuffer_status_t Struct Reference	726
11.105.1	Detailed Description	730
11.105.2	Field Documentation	730
11.105.2.1	ltracebuffer0	730
11.105.2.2	size0	730
11.105.2.3	version0	730
11.105.2.4	ltracebuffer1	730
11.105.2.5	size1	730
11.105.2.6	version1	730
11.105.2.7	cnt_iobmap_tlb_flush	731
11.106	14_tracebuffer_status_window_t Struct Reference	731
11.106.1	Detailed Description	731

11.10 4 ₄ _vcon_attr_t Struct Reference	731
11.107. Detailed Description	732
11.10 4 ₄ _vcpu_ipc_regs_t Struct Reference	732
11.108. Detailed Description	733
11.10 4 ₄ _vcpu_regs_t Struct Reference	734
11.109. Detailed Description	735
11.109. Field Documentation	736
11.109.2. ldi	736
11.109.2.2si	736
11.109.2.3bp	736
11.109.2.4bx	736
11.109.2.5dx	736
11.109.2.6cx	736
11.109.2.7ax	736
11.11 4 ₄ _vcpu_state_t Struct Reference	737
11.110. Detailed Description	740
11.11 4 ₄ _vhw_descriptor Struct Reference	740
11.111. Detailed Description	742
11.111. Field Documentation	742
11.111.2. lmagic	742
11.111.2.2version	742
11.111.2.3count	742
11.111.2.4descs	743
11.11 4 ₄ _vhw_entry Struct Reference	743
11.112. Detailed Description	743
11.112. Field Documentation	744
11.112.2. ltype	744
11.112.2.2provider_pid	744
11.112.2.3mem_start	744
11.112.2.4mem_size	744
11.112.2.5rq_no	744
11.112.2.6fd	745
11.11 4 ₄ _vm_state Struct Reference	745
11.113. Detailed Description	745
11.11 4 ₄ _vm_svm_vmcb_control_area Struct Reference	745
11.114. Detailed Description	745

11.115 <code>4_vm_svm_vmcb_state_save_area</code> Struct Reference	745
11.115.1Detailed Description	747
11.116 <code>4_vm_svm_vmcb_state_save_area_seg</code> Struct Reference	747
11.116.1Detailed Description	747
11.117 <code>4_vm_svm_vmcb_t</code> Struct Reference	747
11.117.1Detailed Description	749
11.118 <code>4re_aux_t</code> Struct Reference	749
11.118.1Detailed Description	749
11.119 <code>4re_ds_stats_t</code> Struct Reference	749
11.119.1Detailed Description	750
11.120 <code>4re_elf_aux_mword_t</code> Struct Reference	750
11.120.1Detailed Description	750
11.121 <code>4re_elf_aux_t</code> Struct Reference	750
11.121.1Detailed Description	750
11.122 <code>4re_elf_aux_vma_t</code> Struct Reference	750
11.122.1Detailed Description	751
11.123 <code>4re_env_cap_entry_t</code> Struct Reference	751
11.123.1Detailed Description	751
11.123.2Constructor & Destructor Documentation	752
11.123.2.1 <code>4re_env_cap_entry_t</code>	752
11.123.3Field Documentation	752
11.123.3.1 <code>lflags</code>	752
11.123.4 <code>4re_env_t</code> Struct Reference	752
11.123.4.1Detailed Description	754
11.125 <code>4re_event_t</code> Struct Reference	754
11.125.1Detailed Description	755
11.126 <code>4re_video_color_component_t</code> Struct Reference	755
11.126.1Detailed Description	755
11.127 <code>4re_video_goops_info_t</code> Struct Reference	755
11.127.1Detailed Description	757
11.128 <code>4re_video_pixel_info_t</code> Struct Reference	757
11.128.1Detailed Description	758
11.129 <code>4re_video_view_info_t</code> Struct Reference	759
11.129.1Detailed Description	761
11.130 <code>4re_video_view_t</code> Struct Reference	761
11.130.1Detailed Description	761

11.131util_idt_desc_t Struct Reference	762
11.131.1Detailed Description	762
11.132util_idt_header_t Struct Reference	762
11.132.1Detailed Description	763
11.133util_mb_addr_range_t Struct Reference	764
11.133.1Detailed Description	764
11.134util_mb_apm_t Struct Reference	764
11.134.1Detailed Description	764
11.135util_mb_drive_t Struct Reference	765
11.135.1Detailed Description	765
11.135.2Field Documentation	765
11.135.2.1drive_number	765
11.135.2.2drive_mode	765
11.135.2.3drive_cylinders	766
11.136util_mb_info_t Struct Reference	766
11.136.1Detailed Description	767
11.137util_mb_mod_t Struct Reference	767
11.137.1Detailed Description	768
11.137.2Field Documentation	768
11.137.2.1mod_start	768
11.137.2.2mod_end	768
11.138util_mb_vbe_ctrl_t Struct Reference	768
11.138.1Detailed Description	768
11.139util_mb_vbe_mode_t Struct Reference	769
11.139.1Detailed Description	770
11.140xx::List< D, Alloc > Class Template Reference	770
11.140.1Detailed Description	771
11.140.2Member Function Documentation	771
11.140.2.1push_back	771
11.140.2.2push_front	771
11.140.2.3remove	772
11.140.2.4size	772
11.140.2.5operator[]	772
11.140.2.6operator[]	773
11.140.2.7items	773
11.141xx::List_alloc Class Reference	773

11.141.1	Detailed Description	773
11.141.2	Constructor & Destructor Documentation	773
11.141.2.1	<code>IList_alloc</code>	773
11.141.3	Member Function Documentation	774
11.141.3.1	<code>lfree</code>	774
11.141.3.2	<code>alloc</code>	774
11.141.3.3	<code>avail</code>	774
11.142	xx::List_item Class Reference	775
11.142.1	Detailed Description	776
11.142.2	Member Function Documentation	776
11.142.2.1	<code>lget_prev_item</code>	776
11.142.2.2	<code>lget_next_item</code>	776
11.142.2.3	<code>linsert_prev_item</code>	776
11.142.2.4	<code>linsert_next_item</code>	777
11.142.2.5	<code>lremove_me</code>	777
11.142.2.6	<code>lpush_back</code>	777
11.142.2.7	<code>lpush_front</code>	778
11.142.2.8	<code>lremove</code>	778
11.143	4Re::Log Class Reference	779
11.143.1	Detailed Description	782
11.143.2	Member Function Documentation	782
11.143.2.1	<code>lprintn</code>	782
11.143.2.2	<code>lprint</code>	782
11.144	4L4::Factory::Lstr Struct Reference	782
11.144.1	Detailed Description	783
11.145	xx::Lt_functor< Obj > Struct Template Reference	783
11.145.1	Detailed Description	783
11.146	4Re::Mem_alloc Class Reference	783
11.146.1	Detailed Description	786
11.146.2	Member Enumeration Documentation	786
11.146.2.1	<code>lMem_alloc_flags</code>	786
11.146.3	Member Function Documentation	786
11.146.3.1	<code>lalloc</code>	786
11.146.3.2	<code>lfree</code>	787
11.147	4L4::Kip::Mem_desc Class Reference	787
11.147.1	Detailed Description	788

11.147.1	Constructor & Destructor Documentation	788
11.147.2	IMem_desc	788
11.147.3	Member Function Documentation	789
11.147.3.1	first	789
11.147.3.2	count	789
11.147.3.3	count	789
11.147.3.4	start	790
11.147.3.5	end	790
11.147.3.6	size	791
11.147.3.7	type	791
11.147.3.8	sub_type	791
11.147.3.9	~virtual	791
11.147.3.10	~et	792
11.148.1	Meta Class Reference	792
11.148.2	Detailed Description	795
11.148.3	Member Function Documentation	795
11.148.3.1	lnum_interfaces	795
11.148.3.2	Interface	795
11.148.3.3	supports	796
11.149.1	Re::Vfs::Mman Class Reference	797
11.149.2	Detailed Description	799
11.150.1	Thread::Modify_senders Class Reference	799
11.150.2	Detailed Description	799
11.150.3	Member Function Documentation	799
11.150.3.1	ladd	799
11.151.1	Ipc::Msg_ptr< T > Class Template Reference	800
11.151.2	Detailed Description	800
11.151.3	Constructor & Destructor Documentation	800
11.151.3.1	IMsg_ptr	800
11.152.1	Re::Util::Names::Name Class Reference	801
11.152.2	Detailed Description	801
11.153.1	Re::Namespace Class Reference	801
11.153.2	Detailed Description	804
11.153.3	Member Enumeration Documentation	804
11.153.3.1	lRegister_flags	804
11.153.4	Member Function Documentation	804

11.153.3.1query	804
11.153.3.2query	805
11.153.3.3register_obj	805
11.154xx::New_allocator< _Type > Class Template Reference	806
11.154.1Detailed Description	806
11.154.2Factory::Nil Struct Reference	806
11.155.1Detailed Description	806
11.156xx::Avl_set< Item, Compare, Alloc >::Node Class Reference	806
11.156.1Detailed Description	807
11.156.2Member Function Documentation	807
11.156.2.1valid	807
11.157xx::Nothrow Class Reference	807
11.157.1Detailed Description	808
11.1584Re::Vfs::Ops Class Reference	808
11.158.1Detailed Description	809
11.1594::Ipc::Ostream Class Reference	810
11.159.1Detailed Description	813
11.159.2Member Function Documentation	814
11.159.2.1put	814
11.159.2.2put	814
11.159.2.3tag	814
11.159.2.4tag	815
11.159.2.5send	815
11.1604::Out_of_memory Class Reference	816
11.160.1Detailed Description	819
11.161xx::Pair< First, Second > Struct Template Reference	819
11.161.1Detailed Description	820
11.161.2Constructor & Destructor Documentation	820
11.161.2.1Pair	820
11.162xx::Pair_first_compare< Cmp, Typ > Class Template Reference	820
11.162.1Detailed Description	820
11.162.2Constructor & Destructor Documentation	821
11.162.2.1Pair_first_compare	821
11.162.3Member Function Documentation	821
11.162.3.1operator()	821
11.1634Re::Parent Class Reference	821

11.163.1	Detailed Description	823
11.163.2	Member Function Documentation	824
11.163.2.1	lsignal	824
11.164.1	Re::Video::Pixel_info Class Reference	824
11.164.2	Detailed Description	827
11.164.3	Constructor & Destructor Documentation	827
11.164.3.1	lPixel_info	827
11.164.3.2	lPixel_info	827
11.164.3.3	Member Function Documentation	828
11.164.3.3.1	lr	828
11.164.3.3.2	2g	828
11.164.3.3.3	3b	828
11.164.3.3.4	4a	828
11.164.3.3.5	5bytes_per_pixel	828
11.164.3.3.6	6bits_per_pixel	829
11.164.3.3.7	7has_alpha	829
11.164.3.3.8	8r	829
11.164.3.3.9	9g	829
11.164.3.3.10	10	829
11.164.3.3.11	ld	830
11.164.3.3.12	12bytes_per_pixel	830
11.164.3.3.13	13operator==	830
11.164.3.3.14	14lump	830
11.165.1	Re::Util::Ref_cap< T > Struct Template Reference	831
11.165.2	Detailed Description	831
11.166.1	Re::Util::Ref_del_cap< T > Struct Template Reference	831
11.166.2	Detailed Description	831
11.167.1	Re::Vfs::Regular_file Class Reference	832
11.167.2	Detailed Description	834
11.167.2.1	Member Function Documentation	834
11.167.2.2.1	ldata_space	834
11.167.2.2.2	2ready	835
11.167.2.2.3	3writev	835
11.167.2.2.4	4seek64	835
11.167.2.2.5	5truncate64	835
11.167.2.2.6	6sync	835

11.167.2.7 <code>fdatasync</code>	836
11.167.2.8 <code>get_lock</code>	836
11.167.2.9 <code>set_lock</code>	836
11.168.4 <code>Re::Rm</code> Class Reference	836
11.168.1 Detailed Description	840
11.168.2 Member Enumeration Documentation	840
11.168.2.1 <code>IDetach_result</code>	840
11.168.2.2 <code>Region_flags</code>	840
11.168.2.3 <code>Attach_flags</code>	840
11.168.2.4 <code>Detach_flags</code>	841
11.168.3 Member Function Documentation	841
11.168.3.1 <code>reserve_area</code>	841
11.168.3.2 <code>reserve_area</code>	842
11.168.3.3 <code>free_area</code>	843
11.168.3.4 <code>attach</code>	843
11.168.3.5 <code>attach</code>	844
11.168.3.6 <code>detach</code>	844
11.168.3.7 <code>detach</code>	845
11.168.3.8 <code>detach</code>	845
11.168.3.9 <code>find</code>	846
11.169.4 <code>:Runtime_error</code> Class Reference	847
11.169.1 Detailed Description	850
11.170.4 <code>:Factory::S</code> Class Reference	850
11.170.1 Detailed Description	852
11.170.2 Constructor & Destructor Documentation	852
11.170.2.1 <code>IS</code>	852
11.170.3 Member Function Documentation	853
11.170.3.1 <code>operator l4_mshtag_t</code>	853
11.170.3.2 <code>operator<<</code>	853
11.170.3.3 <code>operator<<</code>	853
11.170.3.4 <code>operator<<</code>	853
11.170.3.5 <code>operator<<</code>	854
11.170.3.6 <code>operator<<</code>	854
11.171.4 <code>:Scheduler</code> Class Reference	854
11.171.1 Detailed Description	857
11.171.2 Member Function Documentation	857

11.171.2. <i>linfo</i>	857
11.171.2.2 <i>run_thread</i>	858
11.171.2.3 <i>idle_time</i>	858
11.171.2.4 <i>is_online</i>	859
11.172.4:: <i>Server</i> < LOOP_HOOKS > Class Template Reference	860
11.172.1 <i>Detailed Description</i>	862
11.172.2 <i>Constructor & Destructor Documentation</i>	862
11.172.2.1 <i>IServer</i>	862
11.172.3 <i>Member Function Documentation</i>	862
11.172.3.1 <i>internal_loop</i>	862
11.173.4:: <i>Server_object</i> Class Reference	863
11.173.1 <i>Detailed Description</i>	865
11.173.2 <i>Member Function Documentation</i>	865
11.173.2.1 <i>dispatch</i>	865
11.174.4xx:: <i>Slab</i> < Type, Slab_size, Max_free, Alloc > Class Template Reference	865
11.174.1 <i>Detailed Description</i>	868
11.174.2 <i>Member Function Documentation</i>	868
11.174.2.1 <i>lalloc</i>	868
11.174.2.2 <i>free</i>	868
11.175.5xx:: <i>Slab_static</i> < Type, Slab_size, Max_free, Alloc > Class Template Reference	869
11.175.1 <i>Detailed Description</i>	871
11.175.2 <i>Member Function Documentation</i>	871
11.175.2.1 <i>lalloc</i>	871
11.176.64:: <i>Ipc::Small_buf</i> Class Reference	871
11.176.1 <i>Detailed Description</i>	872
11.177.74:: <i>Smart_cap</i> < T, SMART > Class Template Reference	872
11.177.1 <i>Detailed Description</i>	875
11.177.2 <i>Constructor & Destructor Documentation</i>	875
11.177.2.1 <i>ISmart_cap</i>	875
11.178.84Re:: <i>Smart_cap_auto</i> < Unmap_flags > Class Template Reference	875
11.178.1 <i>Detailed Description</i>	876
11.179.94Re::Util:: <i>Smart_cap_auto</i> < Unmap_flags > Class Template Reference	877
11.179.1 <i>Detailed Description</i>	877
11.180.04Re::Util:: <i>Smart_count_cap</i> < Unmap_flags > Class Template Reference	877
11.180.1 <i>Detailed Description</i>	878
11.181.14Re::Vfs:: <i>Special_file</i> Class Reference	878

11.181.1	Detailed Description	880
11.181.2	Member Function Documentation	880
11.181.2.1	ioctl	880
11.182.1	L4vcpu::State Class Reference	880
11.182.1	Detailed Description	881
11.182.1	Constructor & Destructor Documentation	881
11.182.1.1	IState	881
11.182.2	Member Function Documentation	881
11.182.2.1	ladd	881
11.182.2.2	clear	881
11.182.2.3	set	881
11.183.1	L4Re::Dataspace::Stats Struct Reference	882
11.183.1	Detailed Description	882
11.184.1	L4::String Class Reference	882
11.184.1	Detailed Description	882
11.185.1	xx::List_item::T_iter< T, Poly > Class Template Reference	882
11.185.1	Detailed Description	885
11.185.1	Member Function Documentation	885
11.185.1.1	lremove_me	885
11.186.1	L4::Task Class Reference	885
11.186.1	Detailed Description	888
11.186.1	Member Function Documentation	888
11.186.1.1	lmap	888
11.186.1.2	unmap	889
11.186.1.3	unmap_batch	890
11.186.1.4	delete_obj	891
11.186.1.5	release_cap	892
11.186.1.6	cap_valid	892
11.186.1.7	cap_has_child	893
11.186.1.8	cap_equal	894
11.186.1.9	add_ku_mem	895
11.187.1	L4::Thread Class Reference	895
11.187.1	Detailed Description	898
11.187.1	Member Function Documentation	898
11.187.1.1	lex_regs	898
11.187.1.2	ex_regs	899

11.187.2.3control	900
11.187.2.4switch_to	900
11.187.2.5stats_time	901
11.187.2.6vcpu_resume_start	901
11.187.2.7vcpu_resume_commit	902
11.187.2.8vcpu_control	902
11.187.2.9vcpu_control_ext	903
11.187.2.10register_del_irq	903
11.187.2.11modify_senders	904
11.188.4::Type_info Struct Reference	905
11.188.1Detailed Description	906
11.189.4::Unknown_error Class Reference	906
11.189.1Detailed Description	908
11.190.4::Vcon Class Reference	909
11.190.1Detailed Description	912
11.190.2Member Function Documentation	912
11.190.2.1send	912
11.190.2.2write	913
11.190.2.3read	913
11.190.2.4set_attr	914
11.190.2.5get_attr	915
11.191.4Re::Util::Vcon_svr< SVR > Class Template Reference	916
11.191.1Detailed Description	916
11.191.2Member Function Documentation	916
11.191.2.1dispatch	916
11.192.4vcpu::Vcpu Class Reference	917
11.192.1Detailed Description	921
11.192.2Member Function Documentation	921
11.192.2.1irq_disable_save	921
11.192.2.2state	922
11.192.2.3state	922
11.192.2.4saved_state	922
11.192.2.5saved_state	922
11.192.2.6irq_enable	922
11.192.2.7irq_restore	923
11.192.2.8halt	923

11.192.2.9	task	923
11.192.2.10	page_fault_entry	923
11.192.2.11	irq_entry	923
11.192.2.12		924
11.192.2.13		924
11.192.2.14		924
11.192.2.15		924
11.192.2.16	entry_sp	924
11.192.2.17	entry_ip	925
11.192.2.18	ext_alloc	925
11.192.2.19	last	925
11.192.2.20	host	925
11.193.1	Re::Video::View Class Reference	926
11.193.1.1	Detailed Description	929
11.193.1.2	Member Enumeration Documentation	929
11.193.1.2.1	IFlags	929
11.193.1.2.2	V_flags	929
11.193.1.3	Member Function Documentation	929
11.193.1.3.1	info	929
11.193.1.3.2	set_info	930
11.193.1.3.3	set_viewport	930
11.193.1.3.4	stack	930
11.193.1.3.5	refresh	931
11.194.1	Re::Vm Class Reference	931
11.194.1.1	Detailed Description	934
11.194.1.2	Member Function Documentation	934
11.194.1.2.1	run	934
11.195.1	Bitmap_base::Word< BITS > Class Template Reference	934
11.195.1.1	Detailed Description	935
12	Example Documentation	937
12.1	examples/clntsrv/client.cc	937
12.2	examples/clntsrv/clntsrv.cfg	938
12.3	examples/clntsrv/server.cc	939
12.4	examples/libs/l4re/c++/mem_alloc/ma+rm.cc	940
12.5	examples/libs/l4re/c++/shared_ds/ds_clnt.cc	941
12.6	examples/libs/l4re/c++/shared_ds/ds_srv.cc	943

12.7 examples/libs/l4re/c++/shared_ds/shared_ds.lua	946
12.8 examples/libs/l4re/c/ma+rm.c	946
12.9 examples/libs/l4re/streammap/client.cc	948
12.10 examples/libs/l4re/streammap/server.cc	949
12.11 examples/libs/l4re/streammap/streammap.cfg	950
12.12 examples/libs/libirq/async_isr.c	951
12.13 examples/libs/libirq/loop.c	952
12.14 examples/libs/shmc/prodcons.c	952
12.15 examples/sys/aliens/main.c	955
12.16 examples/sys/ ipc/ ipc.cfg	957
12.17 examples/sys/ ipc/ ipc_example.c	957
12.18 examples/sys/ isr/main.c	959
12.19 examples/sys/ migrate/ thread_migrate.cc	960
12.20 examples/sys/ migrate/ thread_migrate.cfg	962
12.21 examples/sys/ singlestep/main.c	963
12.22 examples/sys/ start-with-exc/main.c	965
12.23 examples/sys/ utcb- ipc/main.c	967
12.24 examples/sys/ ux-vhw/main.c	969
12.25 hello/server/src/main.c	970
12.26 tmpfs/lib/src/fs.cc	970

Chapter 1

Fiasco.OC & L4 Runtime Environment (L4Re)

1.1 Preface

The intention of this document is to provide a birds eye overview about [L4Re](#) and about the environment in which typical applications and servers run. We highlight here the principled functionality of the servers in the environment but do not discuss their specific interfaces. Detailed documentation about these interface is available in the modules section.

The document is meant as a general overview repeating many design concepts of L4-based systems and capability systems in general. We do though assume familiarity with C++ and an idea on the general concepts and terms of [L4](#): threads --- as an abstraction for execution ---, tasks --- holding the capabilities to kernel objects that are accessible by the threads executing in this task ---, and [IPC](#) over [IPC-gates](#) to send messages and to transfer capabilities between tasks.

1.2 General 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.

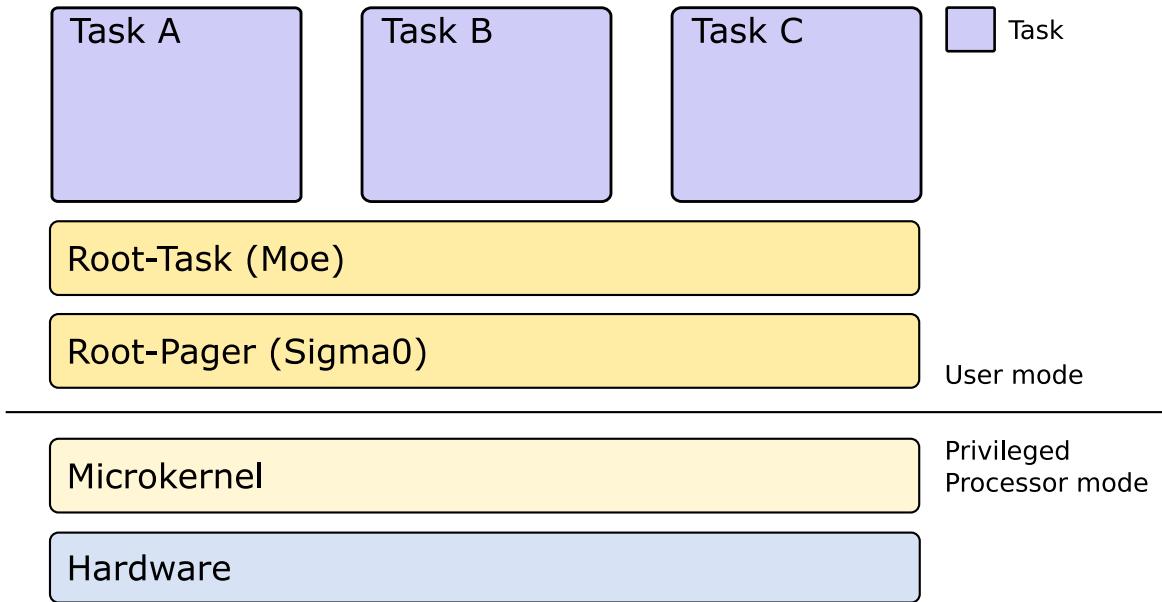


Figure 1.1: Basic Structure of an L4Re based system

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 L4 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 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.

In the following sections, we discuss the basic concepts of our microkernel and its runtime environment in more depth.

1.3 The Fiasco.OC Microkernel

The Fiasco.OC microkernel is the lowest-level piece of software running in an L4-based system. The microkernel is the only program 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 these services and abstractions is provided by the [L4 Runtime Environment](#)).

Fiasco.OC kernel 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*. Fiasco system calls 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.

1.3.1 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.

1.3.2 Kernel Objects

The following list gives a short overview of the kernel objects provided by the Fiasco.OC 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 Fiasco 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.

1.4 L4 Runtime Environment (L4Re)

The [L4 Runtime Environment \(L4Re\)](#) provides a basic set of services and abstractions, which are useful to implement and run user-level applications on top of the Fiasco.OC microkernel.

[L4Re](#) consists of a set of libraries and servers. Libraries as well as server interfaces are completely object oriented. They implement prototype implementations for the classes defined by the [L4Re](#) specification.

A minimal L4Re-based application needs 3 components to be booted beforehand: the Fiasco 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.

1.5 Introduction to L4Re's concepts

This section introduces basic concepts used by [L4Re](#). Understanding of these concepts is a fundamental requirement to understand the inner workings of L4Re's software components and can dramatically help developers in efficiently developing L4Re-based software.

1.6 Memory management - Data Spaces and the Region Map

1.6.1 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 grant the memory in a secure way, often referred to as *memory mapping*.

The described mechanism can be used to construct a memory 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.

1.6.2 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 the some portion are provided and further pages are inserted by the pager as a result of page faults.

1.6.3 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 to attach and detach data spaces to an address space as well as to reserve areas of virtual memory. Since the region-map object possesses all knowledge about virtual memory layout of a task, it also serves as an application's default pager.

1.6.4 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

[Data-Space API](#) and [Region map API](#).

1.7 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 be accessed only 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 flex pages. 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.

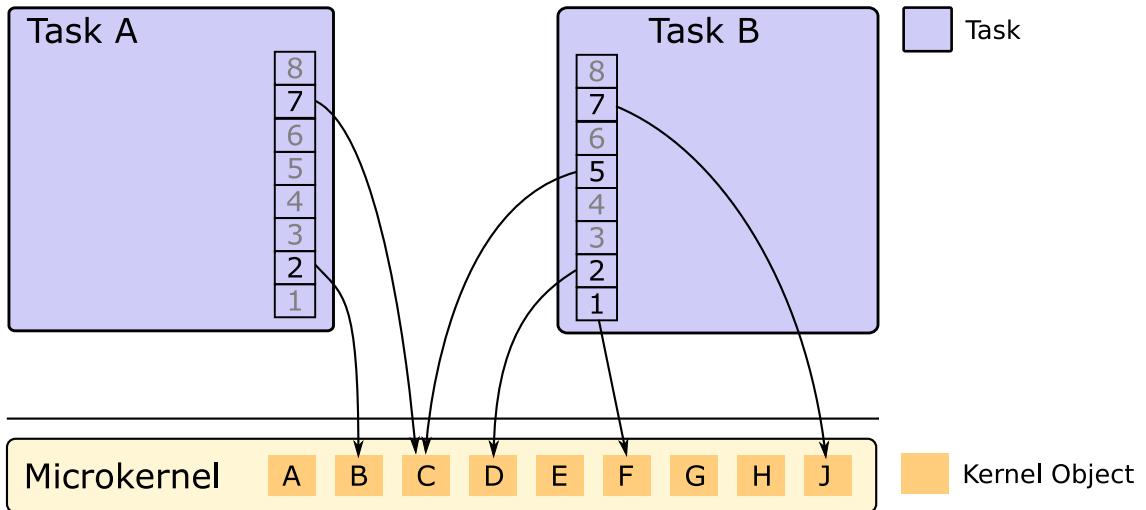


Figure 1.2: Capabilities and Local Naming in L4

So how does an application get access to service? In general all applications are started with an initial set of objects available. This set of objects is predetermined by the creator of a new application process and granted directly to into 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 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.

1.8 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.

1.8.1 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 --a
rg");
```

1.8.2 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 the client and the server. 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:full() } }, "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 (`:full()`) 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 right
                                             ts
loader:start({ caps = { service = svc:full() } }, "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.

1.9 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 Fiasco kernel and connected either to the serial line or to the screen if available.

See also

[Virtual Console](#)

1.10 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 allocation [L4Re](#) environment.

See also

[Memory allocator API](#)

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](#)

1.11 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.

1.11.1 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.

1.11.2 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.

Chapter 2

Getting Started

Here you can find the first steps to boot a very simple setup.

The setup consists of the following components:

- Fiasco.OC --- Microkernel
- Sigma0 --- Root Pager
- Moe --- Root Task
- Ned --- Init Process
- hello --- Hello World Application

The guide assumes that you already compiled the base components and describes how to generate an ISO image, with GRUB 1 or GRUB 2 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 Fiasco.OC 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 `l4/conf/examples`.

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

Or as an alternative use GRUB 2:

```
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-i386 -cdrom images/hello.iso -serial stdio
```

If you press `<ESC>` in the terminal that shows you the serial output you enter the Fiasco.OC kernel debugger... Have fun.

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 3

L4Re Servers

Here you shall find a tight overview over the standard services running on Fiasco.OC and L4Re.

3.1 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 Fiasco.OC you usually need to run Sigma0, nevertheless it is possible to replace Sigma0 by a different implementation.

3.2 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, this 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](#).

3.3 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](#).

3.4 Io, the Platform and Device Resource Manager

Because all peripheral management of Fiasco.OC 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](#).

3.5 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.

3.6 fb-drv, the Low-Level Graphics Driver

The fb-drv server 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-drv*, 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.

3.7 Rtc, the Real-Time Clock Server

Rtc is a simple multiplexer for real-time clock hardware on your platform.

3.8 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.

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

Moe consists of the following subsystems:

- [Name-Space Provider \(L4Re::Namespace\)](#) --- provides instances of name spaces
- [Boot FS](#) --- provides access to the files loaded during platform boot (e.g., linked into the boot image or loaded via GRUB boot loader)
- [Log Subsystem \(L4Re::Log\)](#) --- provides tagged log output for applications
- [l4re_moe_scheduler \(L4::Scheduler\)](#) --- provides simple scheduler objects for scheduling policy enforcement
- [Memory Allocator, Generic Factory \(L4Re::Mem_alloc, L4::Factory\)](#) --- provides allocation of physical RAM as data spaces, as well as allocation of the other [L4Re](#) objects provided by Moe

3.8.1 Memory Allocator, Generic Factory

The generic factory in Moe is responsible for all kinds of dynamic object allocation. The interface is a combination of [L4::Factory](#) and, for traditional reasons, [L4Re::Mem_alloc](#). The generic factory interface allows allocation of the following objects:

- [L4Re::Namespace](#)
- [L4Re::Dataspace](#), RAM allocation
- [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

The memory allocator in Moe is the alternative interface for allocating memory (RAM) in terms of [L4Re::Dataspace](#)-s (

See also

[L4Re::Mem_alloc](#)). The granularity for memory allocation is the machine page size ([L4_PAGESIZE](#)).

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)

3.8.2 Name-Space Provider

Moe provides a name spaces 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.

3.8.3 Boot FS

The Boot FS subsystem provides read only 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.

3.8.4 Log Subsystem

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

3.8.5 Command-Line Options

Moe command-line syntax is:

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

3.8.5.1 --debug=<debug flags>

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

3.8.5.2 --init=<init process>

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

Note

command-line options to the init process are given after the `--` special option.

3.8.5.3 --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=`.

3.8.5.4 --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`.

3.9 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.

3.9.1 Lua Bindings for L4Re

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

3.9.1.1 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.

3.9.1.2 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.

3.9.1.3 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,
  Irq       = -1,
```

```

Sigma0      = -6,
Log         = -13,
Scheduler   = -14,
Factory     = -15,
Ipc_gate   = 0,
}

```

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
}

```

3.9.1.4 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 [L4::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 obejct 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_tag` The string used for tagging log output of this process (defaults to the program name) (see `log_fab`).
- `log_color` The color used for the log tag (defaults to "white").
- `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.

3.10 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 statically.

Each Io 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.

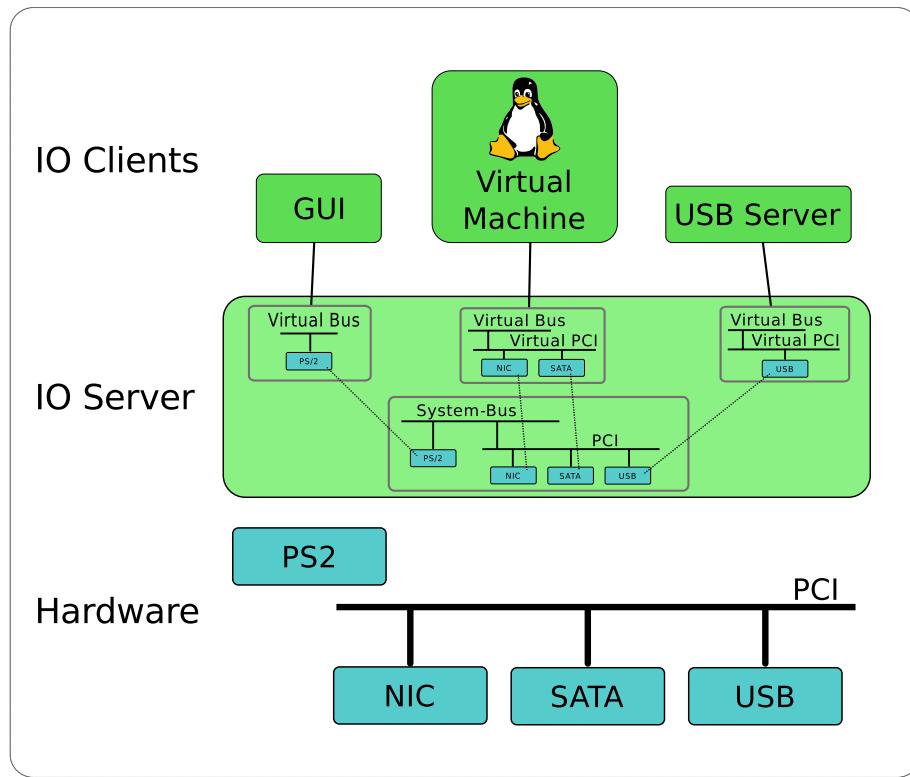


Figure 3.1: IO Service Architecture Overview

The Io 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 Io through the command line.

The platform configuration is stored in the structure called `hw-root`. It lists devices that are available on the platform. For the x86 architecture a basic set of platform devices is defined in the file `x86-legacy.devs`. There are configuration files for various ARM platforms available, as well. If the system contains a PCI bus, it is scanned automatically and the devices found on it are added automatically to the pool of available devices.

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

A very simple configuration for Io could look like this:

```
# Example configuration for io

# Configure 2 platform device to be known to io
hw-root
{
  FOODEVICE => new Device()
  {
    .hid = "FOODEVICE";
    new-res Irq(17);
    new-res Mmio(0x6f000000 .. 0x6f007fff);
  }

  BARDEVICE => new Device()
```

```
{  
    .hid = "BARDEVICE";  
    new-res Irq(19);  
    new-res Irq(20);  
    new-res Mmio(0x6f100000 .. 0x6f100fff);  
}  
}  
  
# Create a virtual bus for a client and give access to FOODEVICE  
client1 => new System_bus()  
{  
    dev => wrap(hw-root.FOODEVICE);  
}  
  
# Create a virtual bus for another client and give it access to  
# BARDEVICE  
client2 => new System_bus()  
{  
    dev => wrap(hw-root.BARDEVICE);  
}
```

Assigning clients PCI devices could look like this:

```
# This is a configuration snippet for PCI device selection  
  
pciclient => new System_bus()  
{  
    pci_storage[] => wrap(hw-root.match("PCI/CC_01"));  
    pci_net[] => wrap(hw-root.match("PCI/CC_02"));  
    pci_mm[] => wrap(hw-root.match("PCI/CC_04"))  
}
```

The CC numbers are PCI class codes. You can also use REV_, VEN_, DEV_ and SUBSYS_ to specify revision, vendor, device and subsystem with a hex number.

Chapter 4

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_getl4cap(pthread_t *t)` to get the capability index of the pthread `t`.

For example:

```
pthread_getl4cap(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);
```


Chapter 5

Module Index

5.1 Modules

Here is a list of all modules:

Base API	115
Basic Macros	122
Cache Consistency	163
Capabilities	267
Error codes	179
Fiasco extensions	125
Fiasco real time scheduling extensions	133
Kernel Debugger	171
Flex pages	142
Integer Types	415
Kernel Interface Page	219
Fiasco-UX Virtual devices	289
Memory descriptors (C version)	222
Kernel Objects	215
Factory	180
IPC-Gate API	119
IRQs	209
Interrupt controller	186
Scheduler	225
Task	230
Thread	238
Thread control	251
vCPU API	287
Virtual Console	281
Virtual Machines	185
VM API for SVM	161
VM API for TZ	292
VM API for VMX	162
Memory operations.	290
Memory related	166
Object Invocation	193
Error Handling	204
Message Items	151

Message Tag	257
Realtime API	209
Timeouts	154
Virtual Registers (UTCBs)	273
ARM Virtual Registers (UTCB)	292
Buffer Registers (BRs)	277
Message Registers (MRs)	277
Exception registers	279
Thread Control Registers (TCRs)	278
amd64 Virtual Registers (UTCB)	294
x86 Virtual Registers (UTCB)	295
Client/Server IPC Framework	46
IO interface	369
IPC Messaging Framework	53
IPC Streams	47
IRQ handling library	375
Interface for asynchronous ISR handlers.	379
Interface for asynchronous ISR handlers with a given IRQ capability.	382
Interface using direct functionality.	375
Interface using direct functionality.	380
L4Re C Interface	112
Capability allocator	85
Dataspace interface	60
Debug interface	62
Event interface	63
Kumem allocator utility	85
L4Re Util C Interface	114
Log interface	66
Memory allocator	69
Namespace interface	73
Region map interface	74
Video API	87
L4Re C++ Interface	56
Auxiliary data	104
C++ Exceptions	43
Console API	93
Data-Space API	94
Debugging API	95
Event API	103
Goos video API	112
Initial Environment	98
L4Re ELF Auxiliary Information	95
L4Re Protocol identifiers	107
L4Re Util C++ Interface	59
Kumem utilties	111
L4Re Capability API	109
Logging interface	104
Memory allocator API	105
Name-space API	106
Parent API	106
Region map API	108
Shared Memory Library	397

Chunks	401
Consumer	406
Producer	404
Signals	409
Consumer	412
Producer	412
Sigma0 API	383
Internal constants	388
Small C++ Template Library	44
Utility Functions	353
Atomic Instructions	306
Bit Manipulation	313
Bitmap graphics and fonts	361
Functions for rendering bitmap data in frame buffers	362
Functions for rendering bitmap fonts to frame buffers	365
CPU related functions	296
Comfortable Command Line Parsing	347
ELF binary format	319
Functions to manipulate the local IDT	298
IA32 Port I/O API	357
Internal functions	313
Kernel Interface Page API	345
Low-Level Thread Functions	351
Machine Restarting Function	350
Priority related functions	349
Random number support	349
Timestamp Counter	299
vCPU Support Library	389
Extended vCPU support	396

Chapter 6

Namespace Index

6.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

cxx (Various kinds of C++ utilities)	419
cxx::Bits (Internal helpers for the cxx package)	421
L4 (L4 low-level kernel interface)	421
L4::Ipc_svr (Helper classes for L4::Server instantiation)	425
L4Re (L4 Runtime Environment)	426
L4Re::Vfs (Virtual file system for interfaces POSIX libc)	427

Chapter 7

Data Structure Index

7.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

L4::Alloc_list	429
L4::Thread::Attr	429
L4Re::Util::Auto_cap< T >	433
L4Re::Util::Auto_del_cap< T >	434
cxx::Auto_ptr< T >	436
cxx::Avl_set< Item, Compare, Alloc >	446
cxx::Avl_set< Pair< Key, Data >, Pair_first_compare< Compare< Key >, Pair< Key, Data > >, Alloc >	446
cxx::Avl_map< Key, Data, Compare, Alloc >	439
cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >	466
cxx::Base_slab< sizeof(Type), Slab_size, Max_free, Alloc >	466
cxx::Slab< Type, Slab_size, Max_free, Alloc >	865
cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >	470
cxx::Base_slab_static< sizeof(Type), Slab_size, Max_free, Alloc >	470
cxx::Slab_static< Type, Slab_size, Max_free, Alloc >	869
L4::Basic_registry	474
cxx::Bitmap_base	485
cxx::Bitmap< BITS >	482
cxx::Bits::Bst< Node, Get_key, Compare >	493
cxx::Avl_tree< Node, Get_key, Compare >	455
cxx::Bits::Bst< _Node, Get_key, Pair_first_compare< Compare< Key >, Pair< Key, Data > > >	493
cxx::Avl_tree< _Node, Get_key, Pair_first_compare< Compare< Key >, Pair< Key, Data > > >	455
cxx::Bits::Bst_node	503
cxx::Avl_tree_node	461
L4::Ipc::Buf_cp_in< T >	506
L4::Ipc::Buf_cp_out< T >	507
L4::Ipc::Buf_in< T >	509
L4Re::Cap_alloc	513
L4Re::Util::Cap_alloc_base	517
L4::Cap_base	519

L4::Cap< T >	510
L4::Smart_cap< T, SMART >	872
cxx::Bitmap_base::Char< BITS >	527
L4Re::Video::Color_component	527
L4::Ipc_svr::Compound_reply	533
L4::Ipc_svr::Default_loop_hooks	559
L4Re::Util::Counting_cap_alloc< COUNTERTYPE >	536
L4Re::Util::Dataspace_svr	548
L4::Ipc_svr::Default_setup_wait	560
L4::Ipc_svr::Default_loop_hooks	559
L4::Ipc_svr::Default_timeout	560
L4::Ipc_svr::Default_loop_hooks	559
cxx::Bits::Direction	561
L4Re::Vfs::Directory	563
L4Re::Vfs::File	623
L4Re::Vfs::Be_file	475
Elf32_Dyn	574
Elf32_Ehdr	574
Elf32_Phdr	576
Elf32_Shdr	577
Elf32_Sym	578
Elf64_Dyn	578
Elf64_Ehdr	579
Elf64_Phdr	581
Elf64_Shdr	581
Elf64_Sym	582
L4Re::Env	583
L4Re::Event_buffer_t< PAYLOAD >::Event	596
L4Re::Event_buffer_t< PAYLOAD >	601
L4Re::Util::Event_buffer_t< PAYLOAD >	605
L4Re::Util::Event_buffer_consumer_t< PAYLOAD >	597
L4Re::Util::Event_t< PAYLOAD >	609
L4::Exception_tracer	612
L4::Base_exception	464
L4::Invalid_capability	663
L4::Runtime_error	847
L4::Bounds_error	490
L4::Com_error	530
L4::Element_already_exists	568
L4::Element_not_found	571
L4::Out_of_memory	816
L4::Unknown_error	906
L4Re::Vfs::File_system	625
L4Re::Vfs::Be_file_system	479
L4Re::Vfs::Fs	629
L4Re::Vfs::Ops	808
L4Re::Vfs::Generic_file	632
L4Re::Vfs::File	623
gfxbitmap_offset	636
L4Re::Util::Video::Goos_svr	642

L4::Ipc_svr::Ignore_errors	654
L4::Ipc_svr::Default_loop_hooks	559
L4Re::Video::Goos::Info	655
L4Re::Video::View::Info	658
L4::IOModifier	666
L4::Ipc::Istream	684
L4::Ipc::Iostream	667
L4Re::Util::Item_alloc_base	693
cxx::List_item::Iter	695
cxx::List_item::T_iter< T, Poly >	882
cxx::List< D, Alloc >::Iter	699
L4::Kobject	701
L4::Kobject_t< Dataspace, L4::Kobject, L4Re::Protocol::Dataspace >	705
L4Re::Dataspace	537
L4::Kobject_t< Debug_obj, L4::Kobject, Protocol::Debug >	705
L4Re::Debug_obj	551
L4::Kobject_t< Debugger, Kobject, L4_PROTO_DEBUGGER >	705
L4::Debugger	554
L4::Kobject_t< Factory, Kobject, L4_PROTO_FACTORY >	705
L4::Factory	614
L4::Kobject_t< Goos, L4::Kobject, L4Re::Protocol::Goos >	705
L4Re::Video::Goos	636
L4::Kobject_t< Ipc_gate, Kobject, L4_PROTO_KOBJECT >	705
L4::Ipc_gate	674
L4::Kobject_t< Irq_eio, Kobject, L4_PROTO_IRQ >	705
L4::Kobject_t< Mem_alloc, L4::Kobject, L4Re::Protocol::Mem_alloc >	705
L4Re::Mem_alloc	783
L4::Kobject_t< Meta, Kobject, L4_PROTO_META >	705
L4::Meta	792
L4::Kobject_t< Namespace, L4::Kobject, L4Re::Protocol::Namespace >	705
L4Re::Namespace	801
L4::Kobject_t< Parent, L4::Kobject, L4Re::Protocol::Parent >	705
L4Re::Parent	821
L4::Kobject_t< Rm, L4::Kobject, L4Re::Protocol::Rm >	705
L4Re::Rm	836
L4::Kobject_t< Task, Kobject, L4_PROTO_TASK >	705
L4::Task	885
L4::Kobject_t< Vm, Task, L4_PROTO_VM >	705
L4::Vm	931
L4::Kobject_t< Thread, Kobject, L4_PROTO_THREAD >	705
L4::Thread	895
L4::Kobject_t< Vm, Kobject, L4_PROTO_VM >	705
L4::Vm	931
L4::Kobject_2t< Derived, Base1, Base2, PROTO >	704
L4::Kobject_2t< Console, Video::Goos, Event >	704
L4Re::Console	534
L4::Kobject_t< Derived, Base, PROTO >	705
L4::Kobject_t< Icu, Irq_eio, L4_PROTO_IRQ >	705
L4::Icu	646
L4::Kobject_t< Event, L4::Icu, L4Re::Protocol::Event >	705

L4Re::Event	593
L4::Kobject_t< Scheduler, Icu, L4_PROTO_SCHEDULER >	705
L4::Scheduler	854
L4::Kobject_t< Vcon, Icu, L4_PROTO_LOG >	705
L4::Vcon	909
L4::Kobject_t< Log, L4::Vcon, L4_PROTO_LOG >	705
L4Re::Log	779
L4::Kobject_t< Irq, Irq_eio, L4_PROTO_IRQ >	705
L4::Irq	676
l4_buf_regs_t	707
l4_exc_regs_t	707
l4_fpage_t	710
l4_icu_info_t	711
L4::Icu::Info	661
l4_kernel_info_mem_desc_t	712
l4_kernel_info_t	712
l4_msg_regs_t	715
l4_mshtag_t	715
l4_rt_preemption_t	717
l4_rt_preemption_val32_t	718
l4_rt_preemption_val_t	719
l4_sched_cpu_set_t	720
l4_sched_param_t	720
l4_snd_fpage_t	722
l4_thread_regs_t	723
l4_timeout_s	724
l4_timeout_t	724
l4_tracebuffer_status_t	726
l4_tracebuffer_status_window_t	731
l4_vcon_attr_t	731
l4_vcpu_ipc_regs_t	732
l4_vcpu_regs_t	734
l4_vcpu_state_t	737
L4vcpu::Vcpu	917
l4_vhw_descriptor	740
l4_vhw_entry	743
l4_vm_state	745
l4_vm_svm_vmcb_control_area	745
l4_vm_svm_vmcb_state_save_area	745
l4_vm_svm_vmcb_state_save_area_seg	747
l4_vm_svm_vmcb_t	747
l4re_aux_t	749
l4re_ds_stats_t	749
l4re_elf_aux_mword_t	750
l4re_elf_aux_t	750
l4re_elf_aux_vma_t	750
l4re_env_cap_entry_t	751
l4re_env_t	752
l4re_event_t	754
l4re_video_color_component_t	755
l4re_video_goops_info_t	755
l4re_video_pixel_info_t	757

l4re_video_view_info_t	759
l4re_video_view_t	761
l4util_idt_desc_t	762
l4util_idt_header_t	762
l4util_mb_addr_range_t	764
l4util_mb_apm_t	764
l4util_mb_drive_t	765
l4util_mb_info_t	766
l4util_mb_mod_t	767
l4util_mb_vbe_ctrl_t	768
l4util_mb_vbe_mode_t	769
cxx::List< D, Alloc >	770
cxx::List_alloc	773
cxx::List_item	775
L4::Factory::Lstr	782
cxx::Lt_functor< Obj >	783
L4::Kip::Mem_desc	787
L4Re::Vfs::Mman	797
L4Re::Vfs::Ops	808
L4::Thread::Modify_senders	799
L4::Ipc::Msg_ptr< T >	800
L4Re::Util::Names::Name	801
cxx::New_allocator< _Type >	806
L4::Factory::Nil	806
cxx::Avl_set< Item, Compare, Alloc >::Node	806
cxx::Nothrow	807
L4::Ipc::Ostream	810
L4::Ipc::Iostream	667
cxx::Pair< First, Second >	819
cxx::Pair_first_compare< Cmp, Typ >	820
L4Re::Video::Pixel_info	824
L4Re::Util::Ref_cap< T >	831
L4Re::Util::Ref_del_cap< T >	831
L4Re::Vfs::Regular_file	832
L4Re::Vfs::File	623
L4::Factory::S	850
L4::Server< LOOP_HOOKS >	860
L4::Server_object	863
L4::Ipc::Small_buf	871
L4Re::Smart_cap_auto< Unmap_flags >	875
L4Re::Util::Smart_cap_auto< Unmap_flags >	877
L4Re::Util::Smart_count_cap< Unmap_flags >	877
L4Re::Vfs::Special_file	878
L4Re::Vfs::File	623
L4vcpu::State	880
L4Re::Dataspace::Stats	882
L4::String	882
L4::Type_info	905
L4Re::Util::Vcon_svr< SVR >	916
L4Re::Video::View	926
cxx::Bitmap_base::Word< BITS >	934

Chapter 8

Data Structure Index

8.1 Data Structures

Here are the data structures with brief descriptions:

L4::Alloc_list (A simple list-based allocator)	429
L4::Thread::Attr (Thread attributes used for control_commit())	429
L4Re::Util::Auto_cap< T > (Automatic capability that implements automatic free and unmap of the capability selector)	433
L4Re::Util::Auto_del_cap< T > (Automatic capability that implements automatic free and unmap+delete of the capability selector)	434
cxx::Auto_ptr< T > (Smart pointer with automatic deletion)	436
cxx::Avl_map< Key, Data, Compare, Alloc > (AVL tree based associative container)	439
cxx::Avl_set< Item, Compare, Alloc > (AVL Tree for simple comapreable items)	446
cxx::Avl_tree< Node, Get_key, Compare > (A generic AVL tree)	455
cxx::Avl_tree_node (Node of an AVL tree)	461
L4::Base_exception (Base class for all exceptions, thrown by the L4Re framework)	464
cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc > (Basic slab allocator)	466
cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc > (Merged slab allocator (allocators for objects of the same size are merged together))	470
L4::Basic_registry (This registry returns the corresponding server object based on the label of an Ipc_gate)	474
L4Re::Vfs::Be_file (Boiler plate class for implementing an open file for L4Re::Vfs)	475
L4Re::Vfs::Be_file_system (Boilerplate class for implementing a L4Re::Vfs::File_system)	479
cxx::Bitmap< BITS > (A static bit map)	482
cxx::Bitmap_base (Basic bitmap abstraction)	485
L4::Bounds_error (Access out of bounds)	490
cxx::Bits::Bst< Node, Get_key, Compare > (Basic binary search tree (BST))	493
cxx::Bits::Bst_node (Basic type of a node in a binary search tree (BST))	503
L4::Ipc::Buf_cp_in< T > (Abstraction for extracting array from an Ipc::Istream)	506
L4::Ipc::Buf_cp_out< T > (Abstraction for inserting an array into an Ipc::Ostream)	507
L4::Ipc::Buf_in< T > (Abstraction to extract an array from an Ipc::Istream)	509
L4::Cap< T > (Capability Selector a la C++)	510
L4Re::Cap_alloc (Capability allocator interface)	513
L4Re::Util::Cap_alloc_base (Capability allocator)	517
L4::Cap_base (Base class for all kinds of capabilities)	519
cxx::Bitmap_base::Char< BITS > (Helper abstraction for a byte contained in the bitmap)	527
L4Re::Video::Color_component (A color component)	527

L4::Com_error (Error conditions during IPC)	530
L4::Ipc_svr::Compound_reply (Mix in for LOOP_HOOKS to always use compound reply and wait)	533
L4Re::Console (Console class)	534
L4Re::Util::Counting_cap_alloc< COUNTERTYPE > (Reference-counting cap allocator)	536
L4Re::Dataspace (This class represents a data space)	537
L4Re::Util::Dataspace_svr (Dataspace server class)	548
L4Re::Debug_obj (Debug interface)	551
L4::Debugger (Debugger interface)	554
L4::Ipc_svr::Default_loop_hooks (Default LOOP_HOOKS)	559
L4::Ipc_svr::Default_setup_wait (Mix in for LOOP_HOOKS for setup_wait no op)	560
L4::Ipc_svr::Default_timeout (Mix in for LOOP_HOOKS to use a 0 send and a infinite receive timeout)	560
cxx::Bits::Direction (The direction to go in a binary search tree)	561
L4Re::Vfs::Directory (Interface for a POSIX file that is a directory)	563
L4::Element_already_exists (Exception for duplicate element insertions)	568
L4::Element_not_found (Exception for a failed lookup (element not found))	571
Elf32_Dyn (ELF32 dynamic entry)	574
Elf32_Ehdr (ELF32 header)	574
Elf32_Phdr (ELF32 program header)	576
Elf32_Shdr (ELF32 section header - figure 1-9, page 1-9)	577
Elf32_Sym (ELF32 symbol table entry)	578
Elf64_Dyn (ELF64 dynamic entry)	578
Elf64_Ehdr (ELF64 header)	579
Elf64_Phdr (ELF64 program header)	581
Elf64_Shdr (ELF64 section header)	581
Elf64_Sym (ELF64 symbol table entry)	582
L4Re::Env (Initial Environment (C++ version))	583
L4Re::Event (Event class)	593
L4Re::Event_buffer_t< PAYLOAD >::Event (Event structure used in buffer)	596
L4Re::Util::Event_buffer_consumer_t< PAYLOAD > (An event buffer consumer)	597
L4Re::Event_buffer_t< PAYLOAD > (Event buffer class)	601
L4Re::Util::Event_buffer_t< PAYLOAD > (Event_buffer utility class)	605
L4Re::Util::Event_t< PAYLOAD > (Convenience wrapper for getting access to an event object)	609
L4::Exception_tracer (Back-trace support for exceptions)	612
L4::Factory (C++ L4 Factory, to create all kinds of kernel objects)	614
L4Re::Vfs::File (The basic interface for an open POSIX file)	623
L4Re::Vfs::File_system (Basic interface for an L4Re::Vfs file system)	625
L4Re::Vfs::Fs (POSIX File-system related functionality)	629
L4Re::Vfs::Generic_file (The common interface for an open POSIX file)	632
gfxbitmap_offset (Offsets in pmap[] and bmap[])	636
L4Re::Video::Goos (A goos)	636
L4Re::Util::Video::Goos_svr (Goos server class)	642
L4::Icu (C++ version of an interrupt controller)	646
L4::Ipc_svr::Ignore_errors (Mix in for LOOP_HOOKS to ignore IPC errors)	654
L4Re::Video::Goos::Info (Information structure of a goos)	655
L4Re::Video::View::Info (Information structure of a view)	658
L4::Icu::Info (Info for an ICU)	661
L4::Invalid_capability (Indicates that an invalid object was invoked)	663
L4::IOModifier (Modifier class for the IO stream)	666
L4::Ipc::Iostream (Input/Output stream for IPC [un]marshalling)	667
L4::Ipc_gate (L4 IPC gate)	674
L4::Irq (C++ version of an L4 IRQ)	676
L4::Ipc::Istream (Input stream for IPC unmarshalling)	684

L4Re::Util::Item_alloc_base (Item allocator)	693
cxx::List_item::Iter (Iterator for a list of ListItem-s)	695
cxx::List< D, Alloc >::Iter (Iterator)	699
L4::Kobject (Base class for all kinds of kernel objects, referred to by capabilities)	701
L4::Kobject_2t< Derived, Base1, Base2, PROTO > (Helper class to create an L4Re interface class that is derived from two base classes)	704
L4::Kobject_t< Derived, Base, PROTO > (Helper class to create an L4Re interface class that is derived from a single base class)	705
l4_buf_regs_t (Encapsulation of the buffer-registers block in the UTCB)	707
l4_exc_regs_t (UTCB structure for exceptions)	707
l4_fpage_t (L4 flexpage type)	710
l4_icu_info_t (Info structure for an ICU)	711
l4_kernel_info_mem_desc_t (Memory descriptor data structure)	712
l4_kernel_info_t (L4 Kernel Interface Page)	712
l4_msg_regs_t (Encapsulation of the message-register block in the UTCB)	715
l4_mshtag_t (Message tag data structure)	715
l4_rt_preemption_t (Struct)	717
l4_rt_preemption_val32_t (Struct)	718
l4_rt_preemption_val_t (Struct)	719
l4_sched_cpu_set_t (CPU sets)	720
l4_sched_param_t (Scheduler parameter set)	720
l4_snd_fpage_t (Send-flex-page types)	722
l4_thread_regs_t (Encapsulation of the thread-control-register block of the UTCB)	723
l4_timeout_s (Basic timeout specification)	724
l4_timeout_t (Timeout pair)	724
l4_tracebuffer_status_t (Trace buffer status)	726
l4_tracebuffer_status_window_t (Trace-buffer status window descriptor)	731
l4_vcon_attr_t (Vcon attribute structure)	731
l4_vcpu_ipc_regs_t (VCPU message registers)	732
l4_vcpu_regs_t (VCPU registers)	734
l4_vcpu_state_t (State of a vCPU)	737
l4_vhw_descriptor (Virtual hardware devices description)	740
l4_vhw_entry (Description of a device)	743
l4_vm_state (State structure for TrustZone VMs)	745
l4_vm_svm_vmcb_control_area (VMCB structure for SVM VMs)	745
l4_vm_svm_vmcb_state_save_area (State save area structure for SVM VMs)	745
l4_vm_svm_vmcb_state_save_area_seg (State save area segment selector struct)	747
l4_vm_svm_vmcb_t (Control structure for SVM VMs)	747
l4re_aux_t (Auxiliary descriptor)	749
l4re_ds_stats_t (Information about the data space)	749
l4re_elf_aux_mword_t (Auxiliary vector element for a single unsigned data word)	750
l4re_elf_aux_t (Generic header for each auxiliary vector element)	750
l4re_elf_aux_vma_t (Auxiliary vector element for a reserved virtual memory area)	750
l4re_env_cap_entry_t (Entry in the L4Re environment array for the named initial objects)	751
l4re_env_t (Initial Environment structure (C version))	752
l4re_event_t (Event structure used in buffer)	754
l4re_video_color_component_t (Color component structure)	755
l4re_video_goops_info_t (Goops information structure)	755
l4re_video_pixel_info_t (Pixel_info structure)	757
l4re_video_view_info_t (View information structure)	759
l4re_video_view_t (C representation of a goops view)	761
l4util_idt_desc_t (IDT entry)	762
l4util_idt_header_t (Header of an IDT table)	762

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)	764
l4util_mb_apm_t (APM BIOS info)	764
l4util_mb_drive_t (Drive Info structure)	765
l4util_mb_info_t	766
l4util_mb_mod_t	767
l4util_mb_vbe_ctrl_t (VBE controller information)	768
l4util_mb_vbe_mode_t (VBE mode information)	769
cxx::List< D, Alloc > (Doubly linked list, with internal allocation)	770
cxx::List_alloc (Standard list-based allocator)	773
cxx::List_item (Basic list item)	775
L4Re::Log (Log interface class)	779
L4::Factory::Lstr (Special type to add a pascal string into the factory create stream)	782
cxx::Lt_functor< Obj > (Generic comparator class that defaults to the less-than operator)	783
L4Re::Mem_alloc (Memory allocator)	783
L4::Kip::Mem_desc (Memory descriptors stored in the kernel interface page)	787
L4::Meta (Meta interface that shall be implemented by each L4Re object and gives access to the dynamic type information for L4Re objects)	792
L4Re::Vfs::Mman (Interface for the POSIX memory management)	797
L4::Thread::Modify_senders (Wrapper class for modifying senders)	799
L4::Ipc::Msg_ptr< T > (Pointer to an element of type T in an Ipc::Istream)	800
L4Re::Util::Names::Name (Name class)	801
L4Re::Namespace (Name-space interface)	801
cxx::New_allocator< _Type > (Standard allocator based on operator new ())	806
L4::Factory::Nil (Special type to add a void argument into the factory create stream)	806
cxx::Avl_set< Item, Compare, Alloc >::Node (A smart pointer to a tree item)	806
cxx::Nothrow (Helper type to distinguish the oeprator new version that does not throw exceptions)	807
L4Re::Vfs::Ops (Interface for the POSIX backends for an application)	808
L4::Ipc::Ostream (Output stream for IPC marshalling)	810
L4::Out_of_memory (Exception signalling insufficient memory)	816
cxx::Pair< First, Second > (Pair of two values)	819
cxx::Pair_first_compare< Cmp, Typ > (Comparison functor for Pair)	820
L4Re::Parent (Parent interface)	821
L4Re::Video::Pixel_info (Pixel information)	824
L4Re::Util::Ref_cap< T > (Automatic capability that implements automatic free and unmap of the capability selector)	831
L4Re::Util::Ref_del_cap< T > (Automatic capability that implements automatic free and unmap+delete of the capability selector)	831
L4Re::Vfs::Regular_file (Interface for a POSIX file that provides regular file semantics)	832
L4Re::Rm (Region map)	836
L4::Runtime_error (Exception for an abstract runtime error)	847
L4::Factory::S (Stream class for the create() argument stream)	850
L4::Scheduler (Scheduler object)	854
L4::Server< LOOP_HOOKS > (Basic server loop for handling client requests)	860
L4::Server_object (Abstract server object to be used with L4::Server and L4::Basic_registry)	863
cxx::Slab< Type, Slab_size, Max_free, Alloc > (Slab allocator for object of type Type)	865
cxx::Slab_static< Type, Slab_size, Max_free, Alloc > (Merged slab allocator (allocators for objects of the same size are merged together))	869
L4::Ipc::Small_buf (A receive item for receiving a single capability)	871
L4::Smart_cap< T, SMART > (Smart capability class)	872
L4Re::Smart_cap_auto< Unmap_flags > (Helper for Auto_cap and Auto_del_cap)	875
L4Re::Util::Smart_cap_auto< Unmap_flags > (Helper for Auto_cap and Auto_del_cap)	877

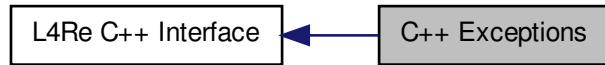
L4Re::Util::Smart_count_cap< Unmap_flags > (Helper for Ref_cap and Ref_del_cap)	877
L4Re::Vfs::Special_file (Interface for a POSIX file that provides special file semantics)	878
L4vcpu::State (C++ implementation of state word in the vCPU area)	880
L4Re::Dataspace::Stats (Information about the data space)	882
L4::String (A null-terminated string container class)	882
cxx::List_item::T_iter< T, Poly > (Iterator for derived classes from ListItem)	882
L4::Task (An L4 Task)	885
L4::Thread (L4 kernel thread)	895
L4::Type_info (Dynamic Type Information for L4Re Interfaces)	905
L4::Unknown_error (Exception for an unknown condition)	906
L4::Vcon (C++ L4 Vcon)	909
L4Re::Util::Vcon_svr< SVR > (Console server template class)	916
L4vcpu::Vcpu (C++ implementation of the vCPU save state area)	917
L4Re::Video::View (View)	926
L4::Vm (Virtual machine)	931
cxx::Bitmap_base::Word< BITS > (Helper abstraction for a word contained in the bitmap)	934

Chapter 9

Module Documentation

9.1 C++ Exceptions

Collaboration diagram for C++ 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.

Files

- file [exceptions](#)
Base exceptions.
- file [std_exc_io](#)
Base exceptions std stream operator.

9.2 Small C++ Template Library

Data Structures

- class [L4::Alloc_list](#)
A simple list-based allocator.
- class [cxx::Auto_ptr< T >](#)
Smart pointer with automatic deletion.
- class [cxx::Avl_map< Key, Data, Compare, Alloc >](#)
AVL tree based associative container.
- class [cxx::Avl_set< Item, Compare, Alloc >](#)
AVL Tree for simple comapreable items.
- class [cxx::Bitmap_base](#)
Basic bitmap abstraction.
- class [cxx::Bitmap< BITS >](#)
A static bit map.
- 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.

Namespaces

- namespace `cxx`
Various kinds of C++ utilities.

Functions

- template<typename T1 >
`T1 cxx::min (T1 a, T1 b)`
Get the minimum of a and b.
- template<typename T1 >
`T1 cxx::max (T1 a, T1 b)`
Get the maximum of a and b.
- `void * operator new (size_t, void *mem, cxx::Nothrow const &) throw ()`
Simple placement new operator.
- `void * operator new (size_t, cxx::Nothrow const &) throw ()`
New operator that does not throw exceptions.

9.2.1 Function Documentation

9.2.1.1 `template<typename T1> T1 cxx::min (T1 a, T1 b) [inline]`

Get the minimum of *a* and *b*.

Parameters

- a* the first value.
- b* the second value.

Examples:

[tmpfs/lib/src/fs.cc](#).

Definition at line 35 of file [minmax](#).

9.2.1.2 `template<typename T1> T1 cxx::max (T1 a, T1 b) [inline]`

Get the maximum of *a* and *b*.

Parameters

- a* the first value.
- b* the second value.

Definition at line 45 of file [minmax](#).

9.2.1.3 `void* operator new (size_t, void * mem, cxx::nothrow const &) throw () [inline]`

Simple placement new operator.

Parameters

- mem* the address of the memory block to place the new object.

Returns

- the address given by *mem*.

Definition at line 39 of file [std_alloc](#).

9.3 Client/Server IPC Framework

Data Structures

- 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](#).

- class [L4::Basic_registry](#)

This registry returns the corresponding server object based on the label of an [Ipc_gate](#).

9.4 IPC Streams

Functions

- [L4::Ipc::Istream & operator>> \(L4::Ipc::Istream &s, bool &v\)](#)
Extract one element of type T from the stream s.
- [L4::Ipc::Istream & operator>> \(L4::Ipc::Istream &s, l4_mshtag_t &v\)](#)
Extract the [L4](#) message tag from the stream s.
- template<typename T >
[L4::Ipc::Istream & operator>> \(L4::Ipc::Istream &s, L4::Ipc::Buf_in< T > const &v\)](#)
Extract an array of T elements from the stream s.
- template<typename T >
[L4::Ipc::Istream & operator>> \(L4::Ipc::Istream &s, L4::Ipc::Msg_ptr< T > const &v\)](#)
Extract an element of type T from the stream s.
- template<typename T >
[L4::Ipc::Istream & operator>> \(L4::Ipc::Istream &s, L4::Ipc::Buf_cp_in< T > const &v\)](#)
Extract an array of T elements from the stream s.
- [L4::Ipc::Ostream & operator<< \(L4::Ipc::Ostream &s, bool v\)](#)
Insert an element to type T into the stream s.
- [L4::Ipc::Ostream & operator<< \(L4::Ipc::Ostream &s, l4_mshtag_t const &v\)](#)
Insert the [L4](#) message tag into the stream s.
- template<typename T >
[L4::Ipc::Ostream & operator<< \(L4::Ipc::Ostream &s, L4::Ipc::Buf_cp_out< T > const &v\)](#)
Insert an array with elements of type T into the stream s.
- [L4::Ipc::Ostream & operator<< \(L4::Ipc::Ostream &s, char const *v\)](#)
Insert a zero terminated character string into the stream s.

9.4.1 Function Documentation

9.4.1.1 [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.

v Output: extracted value.

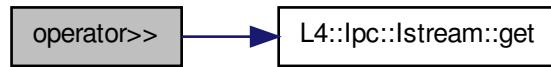
Returns

the stream *s*.

Definition at line 1240 of file `ipc_stream`.

References [L4::Ipc::Istream::get\(\)](#).

Here is the call graph for this function:



9.4.1.2 L4::Ipc::Istream& operator>> (L4::Ipc::Istream & s, l4_mshtag_t & v) [inline]

Extract the [L4](#) message tag from the stream *s*.

Parameters

s The stream to extract from.

v Output: the extracted tag.

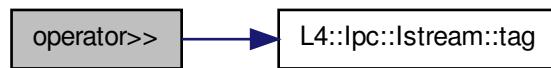
Returns

the stream *s*.

Definition at line 1273 of file `ipc_stream`.

References [L4::Ipc::Istream::tag\(\)](#).

Here is the call graph for this function:



9.4.1.3 `template<typename T > L4::Ipc::Istream& operator>> (L4::Ipc::Istream & s, L4::Ipc::Buf_in< T > const & v) [inline]`

Extract an array of T elements from 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.

See `Ipc::Buf_in`, `Ipc::Buf_cp_in`, and `Ipc::Buf_cp_out`.

Parameters

- s The stream to extract from.
- v Output: pointer to the extracted array (`ipc_buf_in()`).

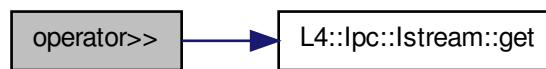
Returns

the stream s .

Definition at line 1295 of file `ipc_stream`.

References [L4::Ipc::Istream::get\(\)](#).

Here is the call graph for this function:



9.4.1.4 `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 .

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`.

Parameters

- s The stream to extract from.
- v Output: pointer to the extracted element.

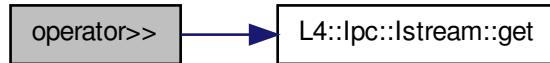
Returns

the stream s .

Definition at line 1320 of file `ipc_stream`.

References [L4::Ipc::Istream::get\(\)](#).

Here is the call graph for this function:



9.4.1.5 `template<typename T > L4::Ipc::Istream& operator>> (L4::Ipc::Istream & s, L4::Ipc::Buf_cp_in< T > const & v) [inline]`

Extract an array of T elements from the stream s .

This operator does a copy out of the data into the given buffer.

See `Ipc::Buf_in`, `Ipc::Buf_cp_in`, and `Ipc::Buf_cp_out`.

Parameters

s The stream to extract from.

v buffer description to copy the array to (`Ipc::Buf_cp_out()`).

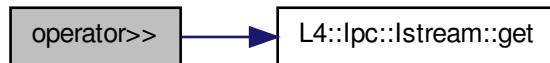
Returns

the stream s .

Definition at line [1341](#) of file `ipc_stream`.

References [L4::Ipc::Istream::get\(\)](#).

Here is the call graph for this function:



9.4.1.6 `L4::Ipc::Ostream& operator<< (L4::Ipc::Ostream & s, bool v) [inline]`

Insert an element to type T into the stream s .

Parameters

s The stream to insert the element *v*.

v The element to insert.

Returns

the stream *s*.

Definition at line 1359 of file ipc_stream.

References [L4::Ipc::Ostream::put\(\)](#).

Here is the call graph for this function:



9.4.1.7 L4::Ipc::Ostream& operator<< (L4::Ipc::Ostream & s, l4_msgtag_t const & v) [inline]

Insert the [L4](#) message tag into the stream *s*.

Note

Only one message tag can be inserted into a stream. Multiple insertions simply overwrite previous insertions.

Parameters

s The stream to insert the tag *v*.

v The [L4](#) message tag to insert.

Returns

the stream *s*.

Definition at line 1393 of file ipc_stream.

References [L4::Ipc::Ostream::tag\(\)](#).

Here is the call graph for this function:



9.4.1.8 `template<typename T > L4::Ipc::Ostream& operator<< (L4::Ipc::Ostream & s, L4::Ipc::Buf_cp_out< T > const & v) [inline]`

Insert an array with elements of type T into the stream s .

Parameters

- s The stream to insert the array v .
- v The array to insert (see `Ipc::Buf_cp_out()`).

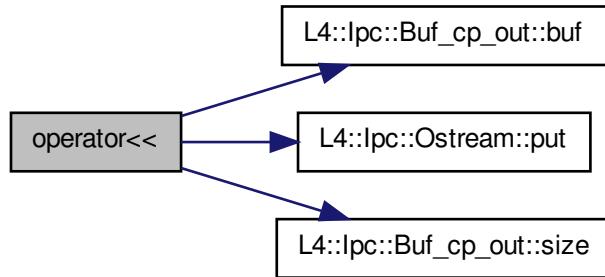
Returns

the stream s .

Definition at line 1409 of file `ipc_stream`.

References `L4::Ipc::Buf_cp_out< T >::buf()`, `L4::Ipc::Ostream::put()`, and `L4::Ipc::Buf_cp_out< T >::size()`.

Here is the call graph for this function:



9.4.1.9 L4::Ipc::Ostream& operator<< (L4::Ipc::Ostream & s, char const * v) [inline]

Insert a zero terminated character string into the stream *s*.

Parameters

- s* The stream to insert the string *v*.
- v* 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 1430 of file `ipc_stream`.

References [L4::Ipc::Ostream::put\(\)](#).

Here is the call graph for this function:



9.5 IPC Messaging Framework

Data Structures

- class [L4::Ipc::Buf_cp_out< T >](#)
Abstraction for inserting an array into an [Ipc::Ostream](#).
- class [L4::Ipc::Buf_cp_in< T >](#)
Abstraction for extracting array from an [Ipc::Istream](#).
- class [L4::Ipc::Msg_ptr< T >](#)
*Pointer to an element of type *T* in an [Ipc::Istream](#).*
- class [L4::Ipc::Buf_in< T >](#)
Abstraction to extract an array from an [Ipc::Istream](#).
- class [L4::Ipc::Istream](#)
Input stream for IPC unmarshalling.
- class [L4::Ipc::Ostream](#)

Output stream for IPC marshalling.

- class [L4::Ipc::Iostream](#)

Input/Output stream for IPC [un]marshalling.

Functions

- template<typename T >
`Buf_cp_out< T > L4::Ipc::buf_cp_out (T *v, unsigned long size)`
Create an instance of [Buf_cp_out](#) for the given values.
- template<typename T >
`Buf_cp_in< T > L4::Ipc::buf_cp_in (T *v, unsigned long &size)`
Create an [Buf_cp_in](#) for the given values.
- template<typename T >
`Msg_ptr< T > L4::Ipc::msg_ptr (T *&p)`
Create an [Msg_ptr](#) to adjust the given pointer.
- template<typename T >
`Buf_in< T > L4::Ipc::buf_in (T *&v, unsigned long &size)`
Create an [Buf_in](#) for the given values.

9.5.1 Function Documentation

9.5.1.1 template<typename T > `Buf_cp_out<T> L4::Ipc::buf_cp_out (T * v, unsigned long size)`

Create an instance of [Buf_cp_out](#) for the given values.

This function makes it more convenient to insert arrays into an [Ipc::Ostream](#) (

See also

[Buf_cp_out.](#))

Parameters

v Pointer to the array that shall be inserted into an [Ipc::Ostream](#).
size Number of elements in the array.

Definition at line 100 of file [ipc_stream](#).

9.5.1.2 template<typename T > `Buf_cp_in<T> L4::Ipc::buf_cp_in (T * v, unsigned long & size)`

Create an [Buf_cp_in](#) for the given values.

This function makes it more convenient to extract arrays from an [Ipc::Istream](#) (

See also

[Buf_cp_in\(\)](#)

Parameters

v Pointer to the array that shall receive the values from the [Ipc::Istream](#).

size Input: the number of elements the array can take at most
Output: the number of elements found in the stream.

See also

[buf_in\(\)](#) and [buf_cp_out\(\)](#).

Definition at line 151 of file [ipc_stream](#).

9.5.1.3 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 193 of file [ipc_stream](#).

References [L4::Ipc::msg_ptr\(\)](#).

Referenced by [L4::Ipc::msg_ptr\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.5.1.4 template<typename T > Buf_in<T> L4::Ipc::buf_in (T *& *v*, unsigned long & *size*)

Create an [Buf_in](#) for the given values.

This function makes it more convenient to extract arrays from an [Ipc::Istream](#) (See [Buf_in](#).)

Parameters

v Output: pointer to the array within the [Ipc::Istream](#).

size Output: the number of elements found in the stream.

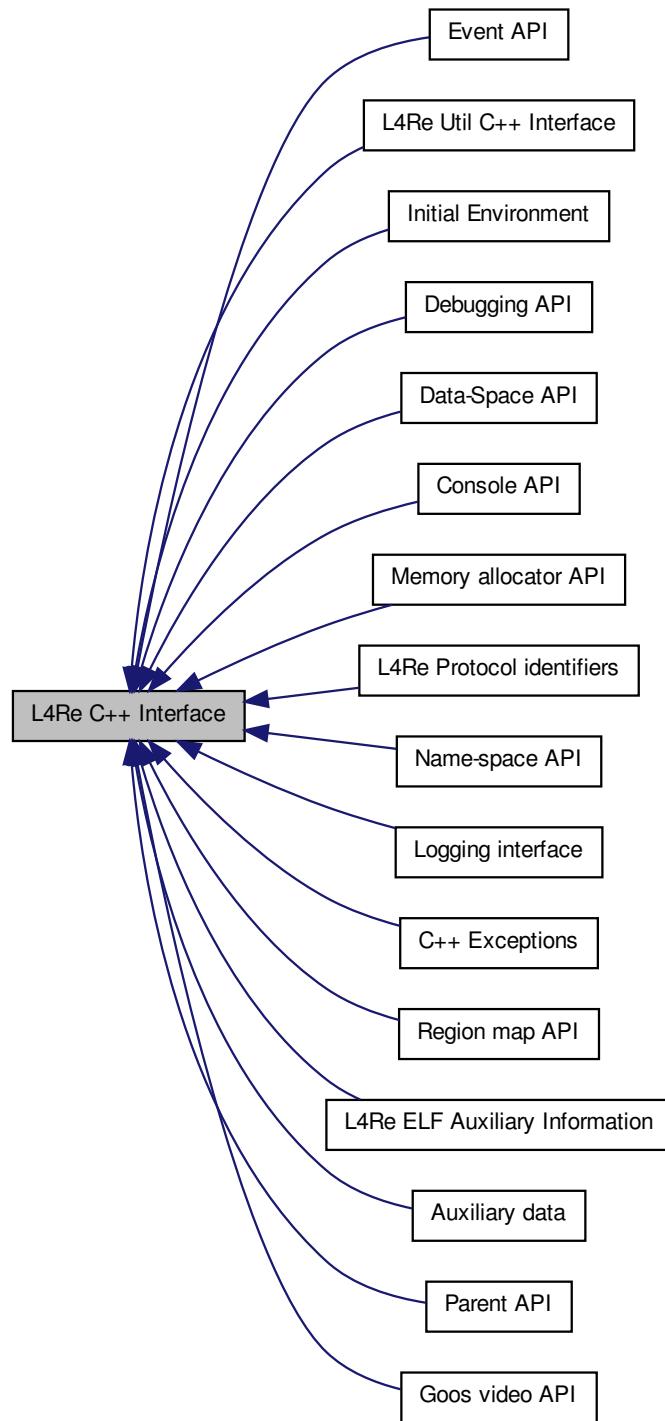
See [buf_cp_in\(\)](#) and [buf_cp_out\(\)](#).

Definition at line 243 of file [ipc_stream](#).

9.6 L4Re C++ Interface

Documentation of the [L4](#) Runtime Environment C++ API.

Collaboration diagram for L4Re C++ Interface:



Modules

- [C++ Exceptions](#)
- [L4Re Util C++ Interface](#)

Documentation of the L4 Runtime Environment utility functionality in C++.

- [Console API](#)

Console interface.

- [Data-Space API](#)

Data-Space API.

- [Debugging API](#)

Debugging Interface.

- [L4Re ELF Auxiliary Information](#)

API for embedding auxiliary information into binary programs.

- [Initial Environment](#)

Environment that is initially provided to an L4 task.

- [Event API](#)

Event interface.

- [Auxiliary data](#)

- [Logging interface](#)

Interface for log output.

- [Memory allocator API](#)

Memory-allocator interface.

- [Name-space API](#)

API for name spaces that store capabilities.

- [Parent API](#)

Parent interface.

- [L4Re Protocol identifiers](#)

Basic protocol identifiers used for L4Re.

- [Region map API](#)

Virtual address-space management.

- [Gos video API](#)

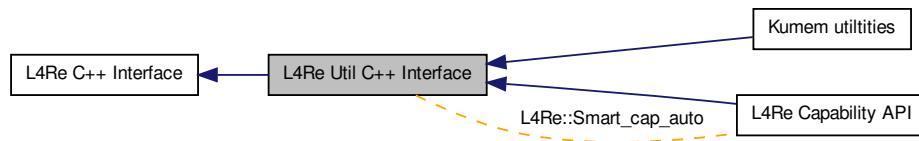
9.6.1 Detailed Description

Documentation of the L4 Runtime Environment C++ API.

9.7 L4Re Util C++ Interface

Documentation of the L4 Runtime Environment utility functionality in C++.

Collaboration diagram for L4Re Util C++ Interface:



Data Structures

- class [L4Re::Smart_cap_auto< Unmap_flags >](#)
Helper for Auto_cap and Auto_del_cap.
- class [L4Re::Util::Cap_alloc_base](#)
Capability allocator.
- class [L4Re::Util::Counting_cap_alloc< COUNTERTYPE >](#)
Reference-counting cap allocator.
- class [L4Re::Util::Dataspace_svr](#)
Dataspace server class.
- 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.

Modules

- [L4Re Capability API](#)
- [Kumem utilities](#)

9.7.1 Detailed Description

Documentation of the [L4](#) Runtime Environment utility functionality in C++.

9.8 Dataspace interface

Dataspace C interface.

Collaboration diagram for Dataspace interface:



Data Structures

- struct [l4re_ds_stats_t](#)
Information about the data space.

Typedefs

- typedef [l4_cap_idx_t](#) [l4re_ds_t](#)
Dataspace type.
- typedef [l4_cap_idx_t](#) [l4re_namespace_t](#)
Dataspace type.

Functions

- long [l4re_ds_clear](#) (const [l4re_ds_t](#) ds, [l4_addr_t](#) offset, unsigned long size) L4_NOTHROW
- long [l4re_ds_allocate](#) (const [l4re_ds_t](#) ds, [l4_addr_t](#) offset, [l4_size_t](#) size) L4_NOTHROW
- int [l4re_ds_copy_in](#) (const [l4re_ds_t](#) ds, [l4_addr_t](#) dst_offs, const [l4re_ds_t](#) src, [l4_addr_t](#) src_offs, unsigned long size) L4_NOTHROW
- long [l4re_ds_size](#) (const [l4re_ds_t](#) ds) L4_NOTHROW
- long [l4re_ds_flags](#) (const [l4re_ds_t](#) ds) L4_NOTHROW
- int [l4re_ds_info](#) (const [l4re_ds_t](#) ds, [l4re_ds_stats_t](#) *stats) L4_NOTHROW
- int [l4re_ds_phys](#) (const [l4re_ds_t](#) ds, [l4_addr_t](#) offset, [l4_addr_t](#) *phys_addr, [l4_size_t](#) *phys_size) L4_NOTHROW

Return physical address.

9.8.1 Detailed Description

Dataspace C interface.

9.8.2 Function Documentation

9.8.2.1 long l4re_ds_clear (*const l4re_ds_t ds, l4_addr_t offset, unsigned long size*)

Returns

0 on success, <0 on errors

See also

[L4Re::Dataspace::clear](#)

9.8.2.2 long l4re_ds_allocate (*const l4re_ds_t ds, l4_addr_t offset, l4_size_t size*)

Returns

0 on success, <0 on errors

See also

[L4Re::Dataspace::allocate](#)

9.8.2.3 int l4re_ds_copy_in (*const l4re_ds_t ds, l4_addr_t dst_offs, const l4re_ds_t src, l4_addr_t src_offs, unsigned long size*)

Returns

0 on success, <0 on errors

See also

[L4Re::Dataspace::copy_in](#)

9.8.2.4 long l4re_ds_size (*const l4re_ds_t ds*)

Returns

size of dataspace, <0 on errors

See also

[L4Re::Dataspace::size](#)

9.8.2.5 long l4re_ds_flags (*const l4re_ds_t ds*)

See also

[L4Re::Dataspace::flags](#)

9.8.2.6 `int l4re_ds_info (const l4re_ds_t ds, l4re_ds_stats_t * stats)`

See also

[L4Re::Dataspace::info](#)

9.8.2.7 `int l4re_ds_phys (const l4re_ds_t ds, l4_addr_t offset, l4_addr_t * phys_addr, l4_size_t * phys_size)`

Return physical address.

Parameters

ds Dataspace

offset Offset in bytes in dataspace

Return values

phys_addr Physical address

phys_size Size of physically contiguous region starting from *phys_addr* (in bytes).

Returns

0 for success, <0 on error

The function returns the physical address of an offset in a dataspace. Use multiple calls of the function to get all physical regions in case of physically non-contiguous dataspaces.

See also

[L4Re::Dataspace::phys](#)

9.9 Debug interface

Collaboration diagram for Debug interface:



Functions

- `void l4re_debug_obj_debug (l4_cap_idx_t srv, unsigned long function) L4_NOTHROW`
Call debug function of L4Re service.

9.9.1 Function Documentation

9.9.1.1 `void l4re_debug_obj_debug (l4_cap_idx_t srv, unsigned long function)`

Call debug function of L4Re service.

Parameters

srv Object to call.

function Function to call.

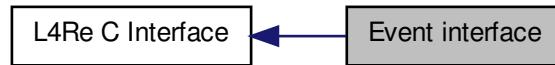
See also

[L4Re::Debug_obj::debug](#)

9.10 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 *axis, l4re_event_absinfo_t *info) L4_NOTHROW`
Get Axis information for a stream.

9.10.1 Detailed Description

Event C interface.

9.10.2 Function Documentation

9.10.2.1 long l4re_event_get_buffer (const l4_cap_idx_t *server*, const l4re_ds_t *ds*)

Get an event signal buffer.

Parameters

server Server to talk to.

ds Buffer to event data.

Returns

0 for success, <0 on error

See also

[L4Re::Event::get_buffer](#)

9.10.2.2 long l4re_event_get_num_streams (const l4_cap_idx_t *server*)

Get number of streams.

Parameters

server Server to talk to.

Returns

0 for success, <0 on error

See also

[L4Re::Event::get_num_streams](#)

9.10.2.3 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

server Server to talk to.

idx Index value.

Return values

info Information buffer.

Returns

0 for success, <0 on error

See also

[L4Re::Event::get_stream_info](#)

9.10.2.4 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

server Server to talk to.

stream_id Stream ID.

Return values

info Information buffer.

Returns

0 for success, <0 on error

See also

[L4Re::Event::get_stream_info_for_id](#)

9.10.2.5 long l4re_event_get_axis_info (const l4_cap_idx_t server, l4_umword_t id, unsigned naxes, unsigned * axis, l4re_event_absinfo_t * info)

Get Axis information for a stream.

Parameters

server Server to talk to.

naxes Number of axes.

Return values

axis Number of axes.

info Information buffer.

Returns

0 for success, <0 on error

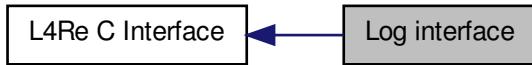
See also

[L4Re::Event::get_axis_info](#)

9.11 Log interface

Log C interface.

Collaboration diagram for Log interface:



Functions

- void [l4re_log_print](#) (char const *string) throw ()
Write a null terminated string to the default log.
- void [l4re_log_printn](#) (char const *string, int len) throw ()
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) throw ()
Write a null terminated string to a log.
- void [l4re_log_printn_srv](#) (const [l4_cap_idx_t](#) logcap, char const *string, int len) throw ()
Write a string of a given length to a log.

9.11.1 Detailed Description

Log C interface.

9.11.2 Function Documentation

9.11.2.1 void [l4re_log_print](#) (char const * *string*) throw () [inline]

Write a null terminated string to the default log.

Parameters

string Text to print, null terminated.

Returns

0 for success, <0 on error

See also

[L4Re::Log::print](#)

Definition at line 99 of file [log.h](#).

References [l4re_log_print_srv\(\)](#), and [l4re_env_t::log](#).

Here is the call graph for this function:



9.11.2.2 void l4re_log_printn (*char const * string*, *int len*) throw () [inline]

Write a string of a given length to the default log.

Parameters

string Text to print, null terminated.

len Length of string in bytes.

Returns

0 for success, <0 on error

See also

[L4Re::Log::printn](#)

Definition at line 105 of file [log.h](#).

References [l4re_log_printn_srv\(\)](#), and [l4re_env_t::log](#).

Here is the call graph for this function:



9.11.2.3 void l4re_log_print_srv (*const l4_cap_idx_t logcap*, *char const * string*) throw ()

Write a null terminated string to a log.

Parameters

logcap Log capability (service).
string Text to print, null terminated.

Returns

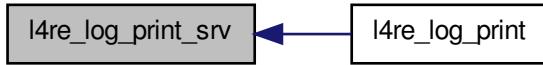
0 for success, <0 on error

See also

[L4Re::Log::print](#)

Referenced by [l4re_log_print\(\)](#).

Here is the caller graph for this function:



9.11.2.4 void l4re_log_printn_srv (const l4_cap_idx_t *logcap*, char const * *string*, int *len*) throw ()

Write a string of a given length to a log.

Parameters

logcap Log capability (service).
string Text to print, null terminated.
len Length of string in bytes.

Returns

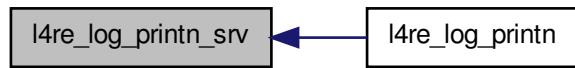
0 for success, <0 on error

See also

[L4Re::Log::printn](#)

Referenced by [l4re_log_printn\(\)](#).

Here is the caller graph for this function:



9.12 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` (unsigned long size, `l4re_ds_t` const mem, unsigned long flags) throw ()
Allocate memory.
- long `l4re_ma_free` (`l4re_ds_t` const mem) throw ()
Free memory.
- long `l4re_ma_alloc_srv` (`l4_cap_idx_t` srv, unsigned long size, `l4re_ds_t` const mem, unsigned long flags) throw ()
Allocate memory.
- long `l4re_ma_free_srv` (`l4_cap_idx_t` srv, `l4re_ds_t` const mem) throw ()
Free memory.

9.12.1 Detailed Description

Memory allocator C interface.

9.12.2 Enumeration Type Documentation

9.12.2.1 enum l4re_ma_flags

Flags for requesting memory at the memory allocator.

See also

[L4Re::Mem_alloc::Mem_alloc_flags](#)

Definition at line 42 of file [mem_alloc.h](#).

9.12.3 Function Documentation

9.12.3.1 long l4re_ma_alloc (unsigned long *size*, l4re_ds_t const *mem*, unsigned long *flags*) throw () [inline]

Allocate memory.

Parameters

size Size to be requested in bytes (granularity is (super)pages and the size is rounded up to this granularity).

mem Capability slot to put the requested dataspace in

flags Flags, see [l4re_ma_flags](#)

Returns

0 on success, <0 on 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 123 of file [mem_alloc.h](#).

References [l4re_ma_alloc_srv\(\)](#), and [l4re_env_t::mem_alloc](#).

Here is the call graph for this function:



9.12.3.2 long l4re_ma_free (l4re_ds_t const mem) throw () [inline]

Free memory.

Parameters

mem Dataspace to free.

Returns

0 on success, <0 on error

See also

[L4Re::Mem_alloc::free](#)

Note

This function is using the [L4Re::Env::env\(\)->mem_alloc\(\)](#) service.

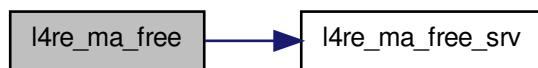
Examples:

[examples/libs/l4re/c/ma+rm.c](#)

Definition at line [130](#) of file [mem_alloc.h](#).

References [l4re_ma_free_srv\(\)](#), and [l4re_env_t::mem_alloc](#).

Here is the call graph for this function:



9.12.3.3 long l4re_ma_alloc_srv (*l4_cap_idx_t* *srv*, *unsigned long* *size*, *l4re_ds_t const* *mem*, *unsigned long* *flags*) throw ()

Allocate memory.

Parameters

srv Memory allocator service.

size Size to be requested.

mem Capability slot to put the requested dataspace in

flags Flags, see [l4re_ma_flags](#)

Returns

0 on success, <0 on error

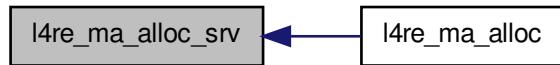
See also

[L4Re::Mem_alloc::alloc](#)

The memory allocator returns a dataspace.

Referenced by [l4re_ma_alloc\(\)](#).

Here is the caller graph for this function:



9.12.3.4 long l4re_ma_free_srv (*l4_cap_idx_t* *srv*, *l4re_ds_t const* *mem*) throw ()

Free memory.

Parameters

srv Memory allocator service.

mem Dataspace to free.

Returns

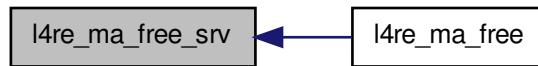
0 on success, <0 on error

See also

[L4Re::Mem_alloc::free](#)

Referenced by [l4re_ma_free\(\)](#).

Here is the caller graph for this function:



9.13 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) throw ()
- long [l4re_ns_register_obj_srv](#) (l4re_namespace_t srv, char const *name, [l4_cap_idx_t](#) const obj, unsigned flags) throw ()

9.13.1 Detailed Description

Namespace C interface.

9.13.2 Enumeration Type Documentation

9.13.2.1 enum l4re_ns_register_flags

Namespace register flags.

See also

[L4Re::Namespace::Register_flags](#)

Definition at line 39 of file [namespace.h](#).

9.13.3 Function Documentation

9.13.3.1 long l4re_ns_query_to_srv (l4re_namespace_t *srv*, char const * *name*, l4_cap_idx_t const *cap*, int *timeout*) throw ()

Returns

0 on success, <0 on error

See also

[L4Re::Namespace::query](#)

9.13.3.2 long l4re_ns_register_obj_srv (l4re_namespace_t *srv*, char const * *name*, l4_cap_idx_t const *obj*, unsigned *flags*) throw ()

Returns

0 on success, <0 on error

See also

[L4Re::Namespace::register_obj](#)

9.14 Region map interface

Region map C interface.

Collaboration diagram for Region map interface:



Enumerations

- enum `l4re_rm_flags_t` {

`L4RE_RM_READ_ONLY` = 0x01, `L4RE_RM_NO_ALIAS` = 0x02, `L4RE_RM_PAGER` = 0x04,

`L4RE_RM_RESERVED` = 0x08,

`L4RE_RM_REGION_FLAGS` = 0x0f, `L4RE_RM_OVERMAP` = 0x10, `L4RE_RM_SEARCH_ADDR` = 0x20, `L4RE_RM_IN_AREA` = 0x40,

`L4RE_RM_EAGER_MAP` = 0x80, `L4RE_RM_ATTACH_FLAGS` = 0xf0 }

Flags for region operations.

Functions

- int `l4re_rm_reserve_area` (`l4_addr_t` *start, unsigned long size, unsigned flags, unsigned char align) throw ()

Detach and unmap in current task.
- int `l4re_rm_free_area` (`l4_addr_t` addr) throw ()

Detach, unmap and return affected dataspace in current task.
- int `l4re_rm_attach` (void **start, unsigned long size, unsigned long flags, `l4re_ds_t` const mem, `l4_addr_t` offs, unsigned char align) throw ()

Detach and unmap in specified task.
- int `l4re_rm_detach` (void *addr) throw ()

Detach and unmap in current task.
- int `l4re_rm_detach_ds` (void *addr, `l4re_ds_t` *ds) throw ()

Detach, unmap and return affected dataspace in current task.
- int `l4re_rm_detach_unmap` (`l4_addr_t` addr, `l4_cap_idx_t` task) throw ()

Detach and unmap in specified task.
- int `l4re_rm_detach_ds_unmap` (void *addr, `l4re_ds_t` *ds, `l4_cap_idx_t` task) throw ()

Detach and unmap in specified task.
- int `l4re_rm_find` (`l4_addr_t` *addr, unsigned long *size, `l4_addr_t` *offset, unsigned *flags, `l4re_ds_t` *m) throw ()

Dump region map internal data structures.
- void `l4re_rm_show_lists` (void) throw ()

Dump region map internal data structures.
- int `l4re_rm_reserve_area_srv` (`l4_cap_idx_t` rm, `l4_addr_t` *start, unsigned long size, unsigned flags, unsigned char align) throw ()

Detach, unmap and return affected dataspace in current task.
- int `l4re_rm_free_area_srv` (`l4_cap_idx_t` rm, `l4_addr_t` addr) throw ()

Detach, unmap and return affected dataspace in current task.
- int `l4re_rm_attach_srv` (`l4_cap_idx_t` rm, void **start, unsigned long size, unsigned long flags, `l4re_ds_t` const mem, `l4_addr_t` offs, unsigned char align) throw ()

Detach and unmap in specified task.
- int `l4re_rm_detach_srv` (`l4_cap_idx_t` rm, `l4_addr_t` addr, `l4re_ds_t` *ds, `l4_cap_idx_t` task) throw ()

Detach and unmap in specified task.
- int `l4re_rm_find_srv` (`l4_cap_idx_t` rm, `l4_addr_t` *addr, unsigned long *size, `l4_addr_t` *offset, unsigned *flags, `l4re_ds_t` *m) throw ()

Dump region map internal data structures.
- void `l4re_rm_show_lists_srv` (`l4_cap_idx_t` rm) throw ()

Dump region map internal data structures.

9.14.1 Detailed Description

Region map C interface.

9.14.2 Enumeration Type Documentation

9.14.2.1 enum l4re_rm_flags_t

Flags for region operations.

Enumerator:

- L4RE_RM_READ_ONLY* Region is read-only.
- L4RE_RM_NO_ALIAS* The region contains exclusive memory that is not mapped anywhere else.
- L4RE_RM_PAGER* Region has a pager.
- L4RE_RM_RESERVED* Region is reserved (blocked).
- L4RE_RM_REGION_FLAGS* Mask of all region flags.
- L4RE_RM_OVERMAP* Unmap memory already mapped in the region.
- L4RE_RM_SEARCH_ADDR* Search for a suitable address range.
- L4RE_RM_IN_AREA* Search only in area, or map into area.
- L4RE_RM_EAGER_MAP* Eagerly map the attached data space in.
- L4RE_RM_ATTACH_FLAGS* Mask of all attach flags.

Definition at line 40 of file [rm.h](#).

9.14.3 Function Documentation

9.14.3.1 int l4re_rm_reserve_area (l4_addr_t * start, unsigned long size, unsigned flags, unsigned char align) throw() [inline]

Returns

0 on success, <0 on error

See also

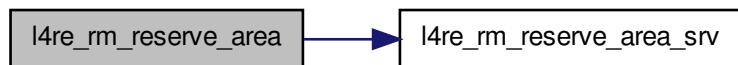
[L4Re::Rm::reserve_area](#)

This function is using the L4::Env::env()->rm() service.

Definition at line 229 of file [rm.h](#).

References [l4re_rm_reserve_area_srv\(\)](#), and [l4re_env_t::rm](#).

Here is the call graph for this function:



9.14.3.2 int l4re_rm_free_area (l4_addr_t *addr*) throw () [inline]**Returns**

0 on success, <0 on error

See also

[L4Re::Rm::free_area](#)

This function is using the L4::Env::env()->rm() service.

Definition at line 237 of file [rm.h](#).

References [l4re_rm_free_area_srv\(\)](#), and [l4re_env_t::rm](#).

Here is the call graph for this function:

**9.14.3.3 int l4re_rm_attach (void ** *start*, unsigned long *size*, unsigned long *flags*, l4re_ds_t *mem*, l4_addr_t *offs*, unsigned char *align*) throw () [inline]****Parameters**

start Virtual start address

size Size of the data space to attach (in bytes)

flags Flags, see [Attach_flags](#) and [Region_flags](#)

mem Data space

offs Offset into the data space to use

align Alignment of the virtual region, log2-size, default: a page ([L4_PAGESHIFT](#)), Only meaningful if the [Search_addr](#) flag is used.

Return values

start Start of region if [Search_addr](#) was used.

Returns

0 on success, <0 on error

- [-L4_ENOENT](#)
- [-L4_EPERM](#)
- [-L4_EINVAL](#)
- [-L4_EADDRNOTAVAIL](#)

- 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 [find](#)).

Returns

0 on success, <0 on error

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 243 of file [rm.h](#).

References [l4re_rm_attach_srv\(\)](#), and [l4re_env_t::rm](#).

Here is the call graph for this function:



9.14.3.4 int l4re_rm_detach (void * *addr*) throw () [inline]

Detach and unmap in current task.

Parameters

addr Address of the region to detach.

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 253 of file [rm.h](#).

References [L4_BASE_TASK_CAP](#), [l4re_rm_detach_srv\(\)](#), and [l4re_env_t::rm](#).

Here is the call graph for this function:



9.14.3.5 int l4re_rm_detach_ds (void * *addr*, l4re_ds_t * *ds*) throw () [inline]

Detach, unmap and return affected dataspace in current task.

Parameters

addr Address of the region to detach.

Return values

ds Returns dataspace that is affected.

Returns

0 on success, <0 on error

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 266 of file [rm.h](#).

References [L4_BASE_TASK_CAP](#), [l4re_rm_detach_srv\(\)](#), and [l4re_env_t::rm](#).

Here is the call graph for this function:



9.14.3.6 `int l4re_rm_detach_unmap(l4_addr_t addr, l4_cap_idx_t task) throw() [inline]`

Detach and unmap in specified task.

Parameters

addr Address of the region to detach.

task 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 260 of file [rm.h](#).

References [l4re_rm_detach_srv\(\)](#), and [l4re_env_t::rm](#).

Here is the call graph for this function:



9.14.3.7 `int l4re_rm_detach_ds_unmap(void * addr, l4re_ds_t * ds, l4_cap_idx_t task) throw() [inline]`

Detach and unmap in specified task.

Parameters

addr Address of the region to detach.

Return values

ds Returns dataspace that is affected.

Parameters

task Task to unmap pages from, specify L4_INVALID_CAP to not unmap

Returns

0 on success, <0 on error

Also

See also

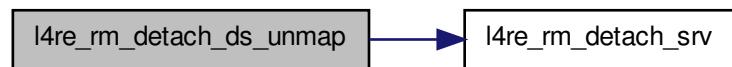
[L4Re::Rm::detach](#)

This function is using the L4::Env::env()->rm() service.

Definition at line [273](#) of file [rm.h](#).

References [l4re_rm_detach_srv\(\)](#), and [l4re_env_t::rm](#).

Here is the call graph for this function:



9.14.3.8 int l4re_rm_find (l4_addr_t * *addr*, unsigned long * *size*, l4_addr_t * *offset*, unsigned * *flags*, l4re_ds_t * *m*) throw () [inline]

Returns

0 on success, <0 on error

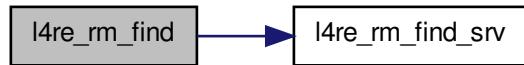
See also

[L4Re::Rm::find](#)

Definition at line [280](#) of file [rm.h](#).

References [l4re_rm_find_srv\(\)](#), and [l4re_env_t::rm](#).

Here is the call graph for this function:



9.14.3.9 void l4re_rm_show_lists (void) throw () [inline]

Dump region map internal data structures.

This function is using the L4::Env::env()->rm() service.

Definition at line 287 of file [rm.h](#).

References [l4re_rm_show_lists_srv\(\)](#), and [l4re_env_t::rm](#).

Here is the call graph for this function:



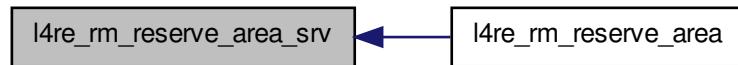
9.14.3.10 int l4re_rm_reserve_area_srv (l4_cap_idx_t rm, l4_addr_t * start, unsigned long size, unsigned flags, unsigned char align) throw ()

See also

[L4Re::Rm::reserve_area](#)

Referenced by [l4re_rm_reserve_area\(\)](#).

Here is the caller graph for this function:



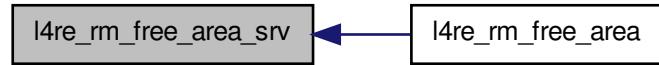
9.14.3.11 int l4re_rm_free_area_srv (l4_cap_idx_t *rm*, l4_addr_t *addr*) throw ()

See also

[L4Re::Rm::free_area](#)

Referenced by [l4re_rm_free_area\(\)](#).

Here is the caller graph for this function:



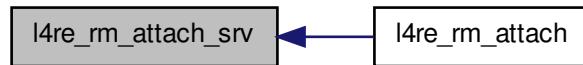
9.14.3.12 int l4re_rm_attach_srv (l4_cap_idx_t *rm*, void ** *start*, unsigned long *size*, unsigned long *flags*, l4re_ds_t const *mem*, l4_addr_t *offs*, unsigned char *align*) throw ()

See also

[L4Re::Rm::attach](#)

Referenced by [l4re_rm_attach\(\)](#).

Here is the caller graph for this function:



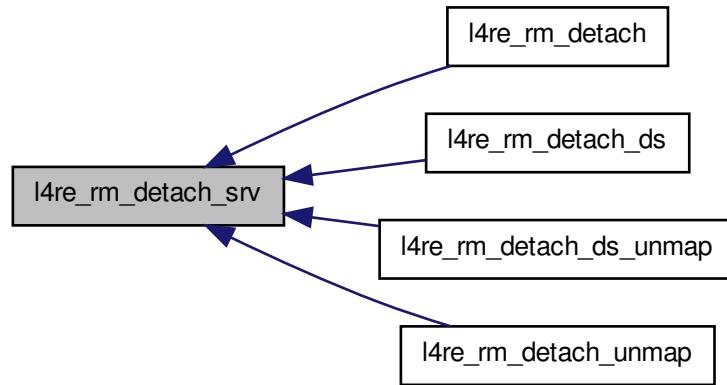
9.14.3.13 int l4re_rm_detach_srv (l4_cap_idx_t rm, l4_addr_t addr, l4re_ds_t * ds, l4_cap_idx_t task) throw ()

See also

[L4Re::Rm::detach](#)

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:



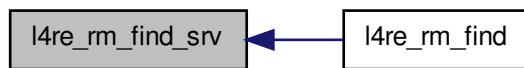
9.14.3.14 int l4re_rm_find_srv (l4_cap_idx_t rm, l4_addr_t * addr, unsigned long * size, l4_addr_t * offset, unsigned * flags, l4re_ds_t * m) throw ()

See also

[L4Re::Rm::find](#)

Referenced by [l4re_rm_find\(\)](#).

Here is the caller graph for this function:



9.15 Capability allocator

Capability allocator C interface.

Collaboration diagram for Capability allocator:



Functions

- `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.

9.15.1 Detailed Description

Capability allocator C interface.

9.15.2 Function Documentation

9.15.2.1 long l4re_util_cap_last (void)

Return last capability index the allocator can return.

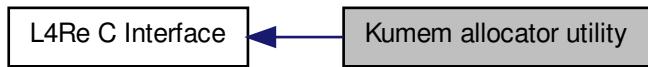
Returns

last/biggest capability index the allocator can return

9.16 Kumem allocator utility

Kumem allocator utility C interface.

Collaboration diagram for Kumem allocator utility:



Functions

- `int l4re_util_kumem_alloc (l4_addr_t *mem, unsigned pages_order, l4_cap_idx_t task, l4_cap_idx_t regmgr) L4_NO_THROW`

Get free capability index at capability allocator.

9.16.1 Detailed Description

Kumem allocator utility C interface.

9.16.2 Function Documentation

9.16.2.1 `int l4re_util_kumem_alloc (l4_addr_t * mem, unsigned pages_order, l4_cap_idx_t task, l4_cap_idx_t regmgr)`

Get free capability index at capability allocator.

Allocate state area.

Return values

`mem` Pointer to memory that has been allocated.

`pages_order` Size to allocate, in log2 pages.

Parameters

`task` Task to use for allocation.

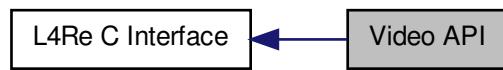
`regmgr` Region manager to use for allocation.

Returns

0 for success, error code otherwise

9.17 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_color_component_t`
Color component structure.
- typedef struct `l4re_video_pixel_info_t` `l4re_pixel_info_t`
Pixel_info structure.
- typedef struct `l4re_video_view_info_t` `l4re_view_info_t`
View information structure.
- typedef struct `l4re_video_view_t` `l4re_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_above` = 0x01000, `F_l4re_video_view_flags_mask` = 0xff000 }

Flags of information on a view.

Functions

- 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.

- 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.

9.17.1 Typedef Documentation

9.17.1.1 `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.

9.17.2 Enumeration Type Documentation

9.17.2.1 `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 39 of file `goos.h`.

9.17.2.2 `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 33 of file [view.h](#).

9.17.3 Function Documentation

9.17.3.1 int l4re_video_goops_info (l4re_video_goops_t *goos*, l4re_video_goops_info_t * *ginfo*)

Get information on a goos.

Parameters

goos Goos object

Return values

ginfo Pointer to goos information structure.

Returns

0 for success, <0 on error

- [-L4_ENODEV](#)
- IPC errors

9.17.3.2 int l4re_video_goops_refresh (l4re_video_goops_t *goos*, int *x*, int *y*, int *w*, int *h*)

Flush a rectangle of pixels of the goos screen.

Parameters

goos the target object of the operation.

x the x-coordinate of the upper left corner of the rectangle

y the y-coordinate of the upper left corner of the rectangle

w the width of the rectangle to be flushed

h the height of the rectangle

9.17.3.3 int l4re_video_goops_create_buffer (l4re_video_goops_t *goos*, unsigned long *size*, l4_cap_idx_t *buffer*)

Create a new buffer (memory buffer) for pixel data.

Parameters

goos the target object for the operation.

size the size in bytes for the pixel buffer.

buffer 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

9.17.3.4 int l4re_video_goops_delete_buffer (l4re_video_goops_t *goos*, unsigned *idx*)

Delete a pixel buffer.

Parameters

goos the target goos object.

idx the buffer index of the buffer to delete (the return value of [l4re_video_goops_create_buffer\(\)](#))

9.17.3.5 int l4re_video_goops_get_static_buffer (l4re_video_goops_t *goos*, unsigned *idx*, l4_cap_idx_t *buffer*)

Get the data-space capability of the static pixel buffer.

Parameters

goos the target goos object.

buffer 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.

9.17.3.6 int l4re_video_goops_create_view (l4re_video_goops_t *goos*, l4re_video_view_t * *view*)

Create a new view (.

See also

[l4re_video_view_t](#)

Parameters

goos the goos session to use.

Return values

view the structure will be initialized for the new view.

9.17.3.7 `int l4re_video_goops_delete_view(l4re_video_goops_t goos, l4re_video_view_t * view)`

Delete a view.

Parameters

goos the goos session to use.

view the view to delete, the given data-structure is invalid afterwards.

9.17.3.8 `int l4re_video_goops_get_view(l4re_video_goops_t goos, unsigned idx, l4re_video_view_t * view)`

Get a view for the given index.

Parameters

goos the target goos session.

idx the index of the view to retrieve.

Return values

view the structure will be 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_goops_create_view\(\)](#).

9.17.3.9 `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

view the target view of the operation.

x x-coordinate of the upper left corner

y y-coordinate of the upper left corner

w the width of the rectangle

h the height of the rectangle

9.17.3.10 `int l4re_video_view_get_info(l4re_video_view_t * view, l4re_video_view_info_t * info)`

Retrieve information about the given *view*.

Parameters

view the target view for the operation.

Return values

info a buffer receiving the information about the view.

9.17.3.11 `int l4re_video_view_set_info (l4re_video_view_t * view, l4re_video_view_info_t * info)`

Set properties of the view.

Parameters

view the target view of the operation.

info 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.

9.17.3.12 `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

view the target view of the operation.

x the x-coordinate of the upper left corner on the screen.

y the y-coordinate of the upper left corner on the screen.

w the width of the view.

h the height of the view.

bofs 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.

9.17.3.13 `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

view the target view for the operation.

pivot the neighbor view, relative to which *view* shall be stacked. a NULL value allows top (*behind* = 1) and bottom (*behind* = 0) placement of the view.

behind describes the placement of the view relative to the *pivot* view.

9.18 Console API

[Console interface](#).

Collaboration diagram for Console API:



Data Structures

- class [L4Re::Console](#)

Console class.

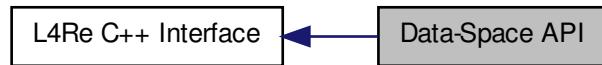
9.18.1 Detailed Description

[Console interface.](#)

9.19 Data-Space API

Data-Space API.

Collaboration diagram for Data-Space API:



Data Structures

- class [L4Re::Dataspace](#)

This class represents a data space.

- struct [L4Re::Dataspace::Stats](#)

Information about the data space.

9.19.1 Detailed Description

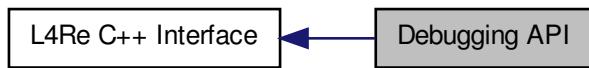
Data-Space API. Data spaces are a central abstraction provided by [L4Re](#). A data space is an abstraction for any thing that is available via usual memory access instructions. A data space can be a file, as well as the memory-mapped registers of a device, or anonymous memory, such as a heap.

The data space interface defines a set of methods that allow any kind of data space to be attached (mapped) to the virtual address space of an [L4](#) task and then be accessed via memory-access instructions. The [region-map interface](#) ([L4Re::Rm](#)) can be used to attach a data space to a virtual address space of a task paged by a certain instance of a region map ([L4Re::Rm](#)).

9.20 Debugging API

Debugging Interface.

Collaboration diagram for Debugging API:



Data Structures

- class [L4Re::Debug_obj](#)

Debug interface.

9.20.1 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.

9.21 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.

Defines

- #define [L4RE_ELF_AUX_ELEM](#) const __attribute__((used, section(".ro14re_elf_aux")))
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
 }

9.21.1 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.

9.21.2 Define Documentation

9.21.2.1 #define L4RE_ELF_AUX_ELEM const __attribute__((used, section(".rol4re_elf_aux")))

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 52 of file [elf_aux.h](#).

9.21.2.2 #define L4RE_ELF_AUX_ELEM_T(*type*, *id*, *tag*, *val...*)

Value:

```
static const __attribute__((used, section(".rol4re_elf_aux"))) \
type id = {tag, sizeof(type), val}
```

Define an auxiliary vector element.

Parameters

type is the data type for the element (e.g., [l4re_elf_aux_vma_t](#))

id is the identifier (variable name) for the declaration (the variable is defined with `static` storage class)

tag is the tag value for the element e.g., [L4RE_ELF_AUX_T_VMA](#)

val 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, 0
x4000 );
```

Definition at line 67 of file [elf_aux.h](#).

9.21.3 Enumeration Type Documentation

9.21.3.1 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.

Definition at line 71 of file [elf_aux.h](#).

9.22 Initial Environment

Environment that is initially provided to an [L4](#) task.

Collaboration diagram for Initial Environment:



Data Structures

- class [L4Re::Env](#)
Initial Environment (C++ version).
- struct [l4re_env_cap_entry_t](#)
Entry in the [L4Re](#) environment array for the named initial objects.
- struct [l4re_env_t](#)
Initial Environment structure (C version).

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 structure (C version).

Functions

- `l4re_env_t * l4re_env (void) throw ()`
Get L4Re initial environment (C version).
- `l4_kernel_info_t * l4re_kip (void) throw ()`
Get Kernel Info Page.
- `l4_cap_idx_t l4re_env_get_cap (char const *name) throw ()`
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) throw ()`
Get the capability selector for the object named name.
- `l4re_env_cap_entry_t const * l4re_env_get_cap_1 (char const *name, unsigned l, l4re_env_t const *e) throw ()`
Get the full l4re_env_cap_entry_t for the object named name.

9.22.1 Detailed Description

Environment that is initially 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).

The initial set of capabilities is:

- C[parent:L4Re::Parent] --- parent object
- C[mem_alloc:L4Re::Mem_alloc] --- initial memory allocator
- C[log:L4Re::Log] --- logging facility
- C[main_thread:L4::Thread] --- first application thread
- C[rm:L4::Rm] --- region manager
- C[factory:L4::Factory] --- factory to create kernel objects
- C[task:L4::Task] --- the task itself

Additional information is:

- First free entry in capability table
- The UTCB area (as flex page)
- First free UTCB (address in the UTCB area)

See also

[L4Re::Env](#), [l4re_env_t](#) for more information.

9.22.2 Typedef Documentation**9.22.2.1 `typedef struct l4re_env_t l4re_env_t`**

Initial Environment structure (C version).

See also

[Initial environment](#)

9.22.3 Function Documentation**9.22.3.1 `l4re_env_t * l4re_env (void) throw () [inline]`**

Get [L4Re](#) initial environment (C version).

Returns

Pointer to [L4Re](#) initial environment (C version).

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 185 of file [env.h](#).

Referenced by [l4re_env_get_cap\(\)](#).

Here is the caller graph for this function:

**9.22.3.2 `l4_kernel_info_t * l4re_kip (void) throw () [inline]`**

Get Kernel Info Page.

Returns

Pointer to Kernel Info Page (KIP) structure.

Examples:

[examples/sys/aliens/main.c](#), and [examples/sys/ux-vhw/main.c](#).

Definition at line 189 of file [env.h](#).

9.22.3.3 l4_cap_idx_t l4re_env_get_cap (char const * name) throw () [inline]

Get the capability selector for the object named *name*.

Parameters

name 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\(\)](#)).

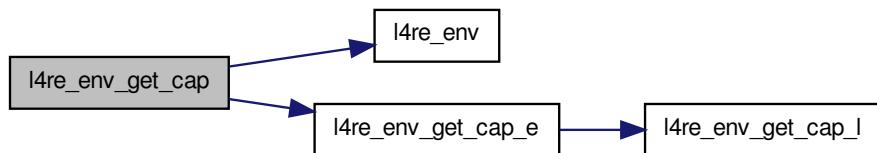
Examples:

[examples/sys/isr/main.c](#).

Definition at line 227 of file [env.h](#).

References [l4re_env\(\)](#), and [l4re_env_get_cap_e\(\)](#).

Here is the call graph for this function:

**9.22.3.4 l4_cap_idx_t l4re_env_get_cap_e (char const * name, l4re_env_t const * e) throw () [inline]**

Get the capability selector for the object named *name*.

Parameters

name is the name of the object to lookup in the initial objects.

e 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 214 of file [env.h](#).

References [l4re_env_cap_entry_t::cap](#), 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:



9.22.3.5 [l4re_env_cap_entry_t const * l4re_env_get_cap_l\(char const * name, unsigned l, l4re_env_t const * e \) throw\(\) \[inline\]](#)

Get the full [l4re_env_cap_entry_t](#) for the object named *name*.

Parameters

name is the name of the object to lookup in the initial objects.

l is the length of the name string, thus *name* might not be zero terminated.

e is the environment structure to use for the operation.

Returns

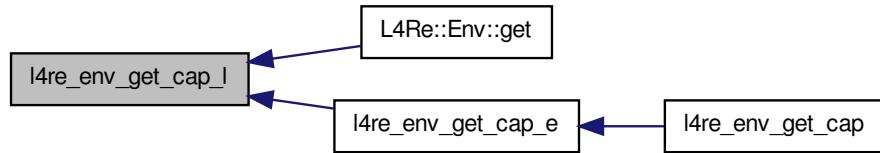
A pointer to an [l4re_env_cap_entry_t](#) if the object exists or NULL if not.

Definition at line 196 of file [env.h](#).

References [l4re_env_cap_entry_t::flags](#), 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:



9.23 Event API

[Event](#) interface.

Collaboration diagram for Event API:



Data Structures

- class [L4Re::Event](#)

Event class.

- class [L4Re::Event_buffer_t< PAYLOAD >](#)

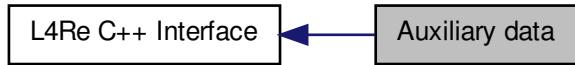
Event buffer class.

9.23.1 Detailed Description

[Event](#) interface.

9.24 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.

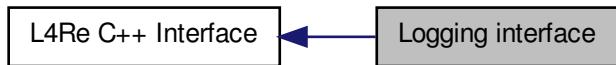
Enumerations

- enum [l4re_aux_ldr_flags_t](#)
Flags for program loading.

9.25 Logging interface

Interface for log output.

Collaboration diagram for Logging interface:



Data Structures

- class [L4Re::Log](#)
Log interface class.

9.25.1 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.

9.26 Memory allocator API

Memory-allocator interface.

Collaboration diagram for Memory allocator API:



Data Structures

- class [L4Re::Mem_alloc](#)
Memory allocator.

9.26.1 Detailed Description

Memory-allocator interface. The memory-allocator API is the basic API to allocate memory from the [L4Re](#) subsystem. The memory is allocated in terms of data spaces (see [L4Re::Dataspace](#)). The provided data spaces have at least the property that data written to such a data space is available as long as the data space is not freed or the data is not overwritten. In particular, the memory backing a data space from an allocator need not be allocated instantly, but may be allocated lazily on demand.

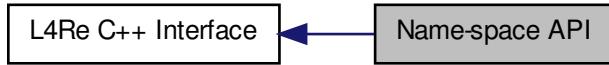
A memory allocator can provide data spaces with additional properties, such as physically contiguous memory, pre-allocated memory, or pinned memory. To request memory with an additional property the [L4Re::Mem_alloc::alloc\(\)](#) method provides a flags parameter. If the concrete implementation of a memory allocator does not support or allow allocation of memory with a certain property, the allocation may be refused.

The main interface is defined by the class [L4Re::Mem_alloc](#).

9.27 Name-space API

API for name spaces that store capabilities.

Collaboration diagram for Name-space API:



Data Structures

- class [L4Re::Namespace](#)
Name-space interface.

9.27.1 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 determine the policy which capabilities (which objects) are accessible to a client of a name space.

9.28 Parent API

[Parent](#) interface.

Collaboration diagram for Parent API:



Data Structures

- class [L4Re::Parent](#)

Parent interface.

9.28.1 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.

See also

[L4Re::Parent](#) for information about the concrete interface.

9.29 L4Re Protocol identifiers

Basic protocol identifiers used for [L4Re](#).

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::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::Protocol::Protocols` {

`L4Re::Protocol::Default = 0, L4Re::Protocol::Dataspace, L4Re::Protocol::Namespace,`
`L4Re::Protocol::Parent,`
`L4Re::Protocol::Goos, L4Re::Protocol::Mem_alloc, L4Re::Protocol::Rm, L4Re::Protocol::Event,`
`L4Re::Protocol::Debug = ~0x7ffffUL }`

Protocols
These protocol IDs are used to distinguish requests for the different `L4Re` interfaces.
- enum `L4Re::Rm_::Opcodes`
Region-map communication-protocol opcodes.
- enum `L4Re::Video::Goos_::Opcodes`
Frame buffer communication-protocol opcodes.

9.29.1 Detailed Description

Basic protocol identifiers used for `L4Re`.

9.29.2 Enumeration Type Documentation

9.29.2.1 enum `L4Re::Protocol::Protocols`

Protocols

These protocol IDs are used to distinguish requests for the different `L4Re` interfaces.

The interfaces use different protocol IDs to enable objects that realize a set of those interfaces at once.

Enumerator:

- `Default`** Default protocol, used in message tag.
- `Dataspace`** ID for data space objects.
- `Namespace`** ID for name space objects.
- `Parent`** ID for parent objects.
- `Goos`** ID for goos objects.
- `Mem_alloc`** ID for memory allocator objects.
- `Rm`** ID for region map objects.
- `Event`** ID for event channel objects.
- `Debug`** ID for debug objects.

Definition at line 44 of file `protocols`.

9.30 Region map API

Virtual address-space management.

Collaboration diagram for Region map API:



Data Structures

- class [L4Re::Rm](#)
Region map.

9.30.1 Detailed Description

Virtual address-space management. The central purpose of the region-map API is to provide means to manage the virtual memory address space of an [L4](#) task. 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, that allows to attach a data-space object to a region of the virtual address space.

There are two basic concepts provided by a region-map abstraction:

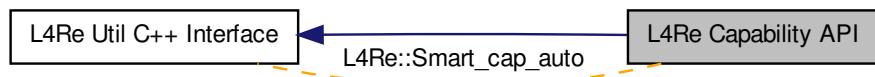
- Regions provide a means to create a view to a data space (or parts of a data space).
- Areas provide a means to reserve areas in a virtual memory address space for special purposes. A reserved area is skipped when searching for an available range of virtual memory, or may be explicitly used to search only within that area.

See also

[Data-Space API.](#) , [L4Re::Dataspace](#), [L4Re::Rm](#)

9.31 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 Auto_cap and Auto_del_cap.
- class [L4Re::Util::Smart_cap_auto< Unmap_flags >](#)
Helper for Auto_cap and Auto_del_cap.
- class [L4Re::Util::Smart_count_cap< Unmap_flags >](#)
Helper for Ref_cap and Ref_del_cap.
- struct [L4Re::Util::Auto_cap< T >](#)
Automatic capability that implements automatic free and unmap of the capability selector.
- struct [L4Re::Util::Auto_del_cap< T >](#)
Automatic capability that implements automatic free and unmap+delete of the capability selector.
- 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

- virtual [L4Re::Cap_alloc::~Cap_alloc \(\)=0](#)
Destructor.

Variables

- [_Cap_alloc](#) & [L4Re::Util::cap_alloc](#)
Capability allocator.

9.31.1 Variable Documentation

9.31.1.1 [_Cap_alloc& L4Re::Util::cap_alloc](#)

Capability allocator.

This is the instance of the capability allocator that is used by usual applications. The actual implementation of the allocator depends on the configuration of the system.

Per default we use a reference count capability allocator, that keeps a reference counter for each managed capability selector.

Note

This capability allocator is not thread-safe.

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/ds_srv.cc](#), and [examples/libs/l4re/streammap/client.cc](#).

Referenced by [L4Re::Util::Smart_count_cap< Unmap_flags >::copy\(\)](#), and [L4Re::Util::Smart_cap_auto< Unmap_flags >::free\(\)](#).

9.32 Kumem utilties

Collaboration diagram for Kumem utilties:



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()) throw ()`
Allocate state area.

9.32.1 Function Documentation

9.32.1.1 `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()) throw ()`

Allocate state area.

Return values

mem Pointer to memory that has been allocated.

pages_order Size to allocate, in log2 pages.

Parameters

task Task to use for allocation.

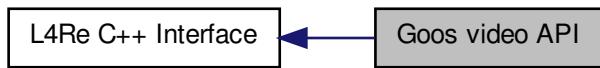
rm Region manager to use for allocation.

Returns

0 for success, error code otherwise

9.33 Goos video API

Collaboration diagram for Goos video API:



Data Structures

- class [L4Re::Video::Color_component](#)

A color component.

- class [L4Re::Video::Pixel_info](#)

Pixel information.

- class [L4Re::Video::Goos](#)

A goos.

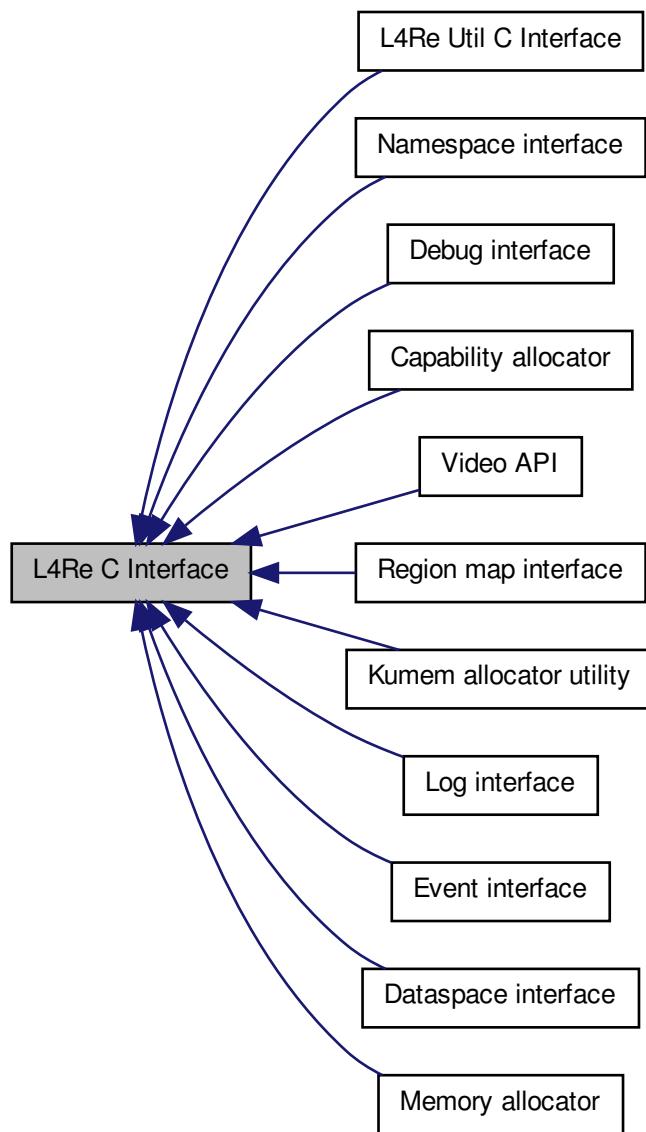
- class [L4Re::Video::View](#)

View.

9.34 L4Re C Interface

Documentation for the [L4Re](#) C Interface.

Collaboration diagram for L4Re C Interface:



Modules

- [Dataspace interface](#)

Dataspace C interface.

- [Debug interface](#)
- [Event interface](#)

Event C interface.

- [Log interface](#)

Log C interface.

- [Memory allocator](#)

Memory allocator C interface.

- [Namespace interface](#)

Namespace C interface.

- [Region map interface](#)

Region map C interface.

- [Capability allocator](#)

Capability allocator C interface.

- [Kumem allocator utility](#)

Kumem allocator utility C interface.

- [Video API](#)

- [L4Re Util C Interface](#)

Documentation of the [L4](#) Runtime Environment utility functionality in C.

9.34.1 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.

9.35 L4Re Util C Interface

Documentation of the [L4](#) Runtime Environment utility functionality in C.

Collaboration diagram for L4Re Util C Interface:



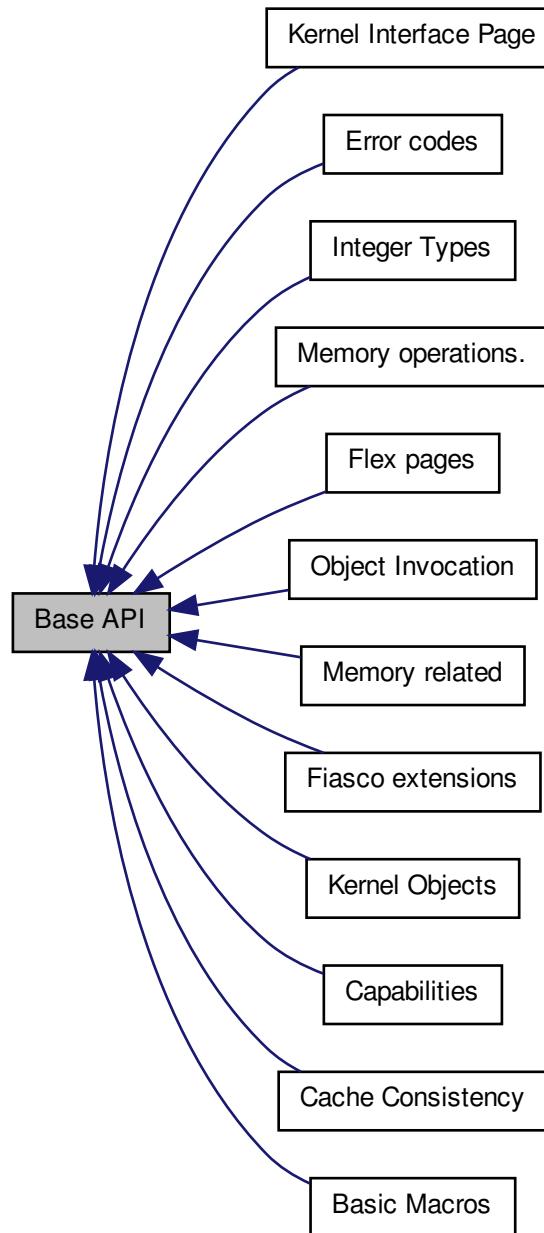
Documentation of the [L4](#) Runtime Environment utility functionality in C. The interface functions closely align with the C++ functions and add no further functionalities.

For new programs it is advised to use the C++ interface.

9.36 Base API

Interfaces for all kinds of base functionality.

Collaboration diagram for Base API:



Modules

- **Basic Macros**

L4 standard macros for header files, function definitions, and public APIs etc.

- **Fiasco extensions**

Kernel debugger extensions of the Fiasco L4 implementation.

- **Flex pages**

Flex-page related API.

- **Cache Consistency**

Various functions for cache consistency.

- **Memory related**

Memory related constants, data types and functions.

- **Error codes**

Common error codes.

- **Object Invocation**

API for L4 object invocation.

- **Kernel Objects**

API of kernel objects.

- **Kernel Interface Page**

Kernel Interface Page.

- **Capabilities**

Functions and definitions related to capabilities.

- **Memory operations.**

Operations for memory access.

- **Integer Types**

`#include<l4/sys/l4int.h>`

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](#)

Common factory 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](#)

Interrupt functionality.

- file [irq.h](#)

Interrupt functionality.

- file [kip](#)

L4::Kip class, memory descriptors.

- file [kip.h](#)

Kernel Info Page access functions.

- file [memdesc.h](#)

Memory description functions.

- file [meta](#)

Meta interface for getting dynamic type information about objects behind capabilities.

- file [types.h](#)

Common L4 ABI Data Types.

- file [vhw.h](#)

Descriptors for virtual hardware (under UX).

- file [consts.h](#)

Common L4 constants, arm version.

- file [types.h](#)

L4 kernel API type definitions.

- file [consts.h](#)

Common L4 constants, amd64 version.

- file [ipc.h](#)

L4 IPC System Calls, x86.

- file `types.h`

L4 kernel API type definitions.

- file `consts.h`

Common L4 constants, x86 version.

9.36.1 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 elapses without the destination becoming ready to receive.

9.37 IPC-Gate API

Secure communication object.

Collaboration diagram for IPC-Gate API:



Data Structures

- class `L4::Ipc_gate`

L4 IPC gate.

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_bind_thread (l4_cap_idx_t gate, l4_cap_idx_t thread, l4_umword_t label)`

Bind the IPC-gate to the thread.

- `l4_mshtag_t l4_ipc_gate_get_infos (l4_cap_idx_t gate, l4_umword_t *label)`

Get information on the IPC-gate.

9.37.1 Detailed Description

Secure communication object. IPC-Gate objects provide a means to establish secure communication channels to L4 Threads (Thread). An IPC-Gate object can be created using a Factory (`l4_factory_create_gate()`) and get assigned a specific L4 thread and a *label* as protected payload. The *label* has the size of one machine word and can only be seen by the Task running the thread that is assigned of the IPC-gate. The *label* is received as part of the IPC message. The *label* can thus be used to securely identify the IPC-gate that was used to send a message.

An IPC-gate is usually used to represent an user-level object and may be the address of the data structure for the object in the server task.

With client privileges an IPC-gate does not provide any direct API and thus an IPC-gate kernel object cannot be modified by invocations. Each invocation of an IPC-gate kernel object is translated into an IPC message to the assigned thread.

See also

[Object Invocation](#)

9.37.2 Enumeration Type Documentation

9.37.2.1 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 75 of file [ipc_gate.h](#).

9.37.3 Function Documentation

9.37.3.1 `l4_mshtag_t l4_ipc_gate_bind_thread (l4_cap_idx_t gate, l4_cap_idx_t thread, l4_umword_t label) [inline]`

Bind the IPC-gate to the thread.

Parameters

t Thread to bind the IPC-gate to

label Label to use

utc UTCB to use.

Returns

System call return tag.

Definition at line 117 of file [ipc_gate.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.37.3.2 `l4_mshtag_t l4_ipc_gate_get_infos (l4_cap_idx_t gate, l4_umword_t * label) [inline]`

Get information on the IPC-gate.

Return values

label Label of the gate.

Parameters

utcb UTCb to use.

Returns

System call return tag.

Definition at line 124 of file [ipc_gate.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.38 Basic Macros

[L4](#) standard macros for header files, function definitions, and public APIs etc.

Collaboration diagram for Basic Macros:



Defines

- `#define L4_DECLARE_CONSTRUCTOR(func, prio)`
L4 Inline function attribute.
- `#define __END_DECLS`
End section with C types and functions.
- `#define EXTERN_C_BEGIN`
Start section with C types and functions.
- `#define EXTERN_C_END`
End section with C types and functions.
- `#define EXTERN_C`
Mark 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_NORETURN`
Noreturn function attribute.
- `#define L4_NOINSTRUMENT`
No instrumentation function attribute.
- `#define EXPECT_TRUE(x)`
Expression is likely to execute.

- `#define EXPECT_FALSE(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 __attribute__((regparm(0)))`
Define calling convention.

Functions

- `void l4_barrier (void)`
Memory barrier.
- `void l4_mb (void)`
Memory barrier.
- `void l4_wmb (void)`
Write memory barrier.

9.38.1 Detailed Description

L4 standard macros for header files, function definitions, and public APIs etc. `#include <l4/sys/compiler.h>`

9.38.2 Define Documentation

9.38.2.1 `#define L4_DECLARE_CONSTRUCTOR(func, prio)`

L4 Inline function attribute.

Handcoded version of `__attribute__((constructor(xx)))`.

Parameters

func function declaration (prototype)
prio the prio must be 65535 - *gcc_prio*

Definition at line 84 of file [compiler.h](#).

9.38.2.2 #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 202 of file [compiler.h](#).

9.38.2.3 #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 230 of file [compiler.h](#).

9.38.2.4 #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

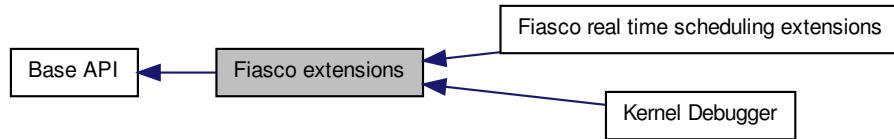
```

Definition at line 227 of file [compiler.h](#).

9.39 Fiasco extensions

Kernel debugger extensions of the Fiasco [L4](#) implementation.

Collaboration diagram for Fiasco extensions:



Data Structures

- struct [l4_tracebuffer_status_t](#)
Trace buffer status.
- struct [l4_tracebuffer_status_window_t](#)
Trace-buffer status window descriptor.

Modules

- [Fiasco real time scheduling extensions](#)
Real time scheduling extension for the Fiasco [L4](#) implementation.
- [Kernel Debugger](#)
Kernel debugger related functionality.

Files

- file [segment.h](#)
l4f specific fs/gs manipulation

- file `segment.h`
l4f specific segment manipulation

Defines

- `#define LOG_EVENT_CONTEXT_SWITCH 0`
Event: context switch.
- `#define LOG_EVENT_IPC_SHORTCUT 1`
Event: IPC shortcut.
- `#define LOG_EVENT_IRQ_RAISED 2`
Event: IRQ occurred.
- `#define LOG_EVENT_TIMER_IRQ 3`
Event: Timer IRQ occurred.
- `#define LOG_EVENT_THREAD_EX_REGS 4`
Event: thread_ex_regs.
- `#define LOG_EVENT_MAX_EVENTS 16`
Maximum number of events.
- `#define LOG_EVENT_CONTEXT_SWITCH 0`
Event: context switch.
- `#define LOG_EVENT_IPC_SHORTCUT 1`
Event: IPC shortcut.
- `#define LOG_EVENT_IRQ_RAISED 2`
Event: IRQ occurred.
- `#define LOG_EVENT_TIMER_IRQ 3`
Event: Timer IRQ occurred.
- `#define LOG_EVENT_THREAD_EX_REGS 4`
Event: thread_ex_regs.
- `#define LOG_EVENT_MAX_EVENTS 16`
Maximum number of events.

Enumerations

- `enum`
Log event types.

Functions

- `l4_tracebuffer_status_t * fiasco_tbuf_get_status (void)`
Return trace buffer status.
- `l4_addr_t fiasco_tbuf_get_status_phys (void)`
Return the physical address of the trace buffer status struct.
- `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, unsigned v1, unsigned v2, unsigned 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.
- `void fiasco_profile_start (void) throw ()`
Start profiling.
- `void fiasco_profile_stop_and_dump (void) throw ()`
Stop profiling and dump result to console.
- `void fiasco_profile_stop (void) throw ()`
Stop profiling.
- `void fiasco_watchdog_enable (void) throw ()`
Enable Fiasco watchdog.
- `void fiasco_watchdog_disable (void) throw ()`
Disable Fiasco watchdog.
- `void fiasco_watchdog_takeover (void) throw ()`
Disable automatic resetting of watchdog.
- `void fiasco_watchdog_giveback (void) throw ()`
Reenable automatic resetting of watchdog.
- `void fiasco_watchdog_touch (void) throw ()`
Reset watchdog from user land.
- `long fiasco_ldt_set (l4_cap_idx_t task, void *ldt, unsigned int size, 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.

9.39.1 Detailed Description

Kernel debugger extensions of the Fiasco [L4](#) implementation.

9.39.2 Function Documentation

9.39.2.1 `l4_tracebuffer_status_t * fiasco_tbuf_get_status (void) [inline]`

Return trace buffer status.

Return trace-buffer status.

Return tracebuffer status.

Returns

Pointer to trace buffer status struct.
 Pointer to tracebuffer status struct.
 Pointer to trace-buffer status struct.

Definition at line 171 of file [ktrace.h](#).

9.39.2.2 `l4_addr_t fiasco_tbuf_get_status_phys (void) [inline]`

Return the physical address of the trace buffer status struct.

Return the physical address of the trace-buffer status struct.

Return the physical address of the tracebuffer status struct.

Returns

physical address of status struct.

Definition at line 178 of file [ktrace.h](#).

References [enter_kdebug](#).

9.39.2.3 `l4_umword_t fiasco_tbuf_log (const char * text) [inline]`

Create new trace buffer entry with describing <text>.

Create new trace-buffer entry with describing <text>.

Create new tracebuffer entry with describing <text>.

Parameters

text Logging text

Returns

Pointer to trace buffer entry

Parameters

text Logging text

Returns

Pointer to tracebuffer entry

Parameters

text Logging text

Returns

Pointer to trace-buffer entry

Definition at line 185 of file [ktrace.h](#).

9.39.2.4 `l4_umword_t fiasco_tbuf_log_3val (const char * text, unsigned v1, unsigned v2, unsigned v3) [inline]`

Create new trace buffer entry with describing <text> and three additional values.

Create new trace-buffer entry with describing <text> and three additional values.

Create new tracebuffer entry with describing <text> and three additional values.

Parameters

text Logging text

v1 first value

v2 second value

v3 third value

Returns

Pointer to trace buffer entry

Parameters

text Logging text

v1 first value

v2 second value

v3 third value

Returns

Pointer to tracebuffer entry

Parameters

text Logging text

v1 first value

v2 second value

v3 third value

Returns

Pointer to trace-buffer entry

Definition at line 191 of file [ktrace.h](#).

9.39.2.5 l4_umword_t fiasco_tbuf_log_binary (const unsigned char * *data*) [inline]

Create new trace buffer entry with binary data.

Create new trace-buffer entry with binary data.

Create new tracebuffer entry with binary data.

Parameters

data binary data

Returns

Pointer to trace buffer entry

Parameters

data binary data

Returns

Pointer to tracebuffer entry

Parameters

data binary data

Returns

Pointer to trace-buffer entry

Definition at line 221 of file [ktrace.h](#).

9.39.2.6 void fiasco_tbuf_clear (void) [inline]

Clear trace buffer.

Clear trace-buffer.

Clear tracebuffer.

Definition at line 197 of file [ktrace.h](#).

9.39.2.7 void fiasco_tbuf_dump (void) [inline]

Dump trace buffer to kernel console.

Dump trace-buffer to kernel console.

Dump tracebuffer to kernel console.

Definition at line 203 of file [ktrace.h](#).

9.39.2.8 void fiasco_watchdog_takeover (void) throw () [inline]

Disable automatic resetting of watchdog.

User is responsible to call `fiasco_watchdog_touch` from time to time to ensure that the watchdog does not trigger.

Definition at line 407 of file [kdebug.h](#).

9.39.2.9 void fiasco_watchdog_touch (void) throw () [inline]

Reset watchdog from user land.

This function **must** be called from time to time to prevent the watchdog from triggering if the watchdog is activated and if `fiasco_watchdog_takeover` was performed.

Definition at line 419 of file [kdebug.h](#).

9.39.2.10 long fiasco_ldt_set (l4_cap_idx_t task, void * ldt, unsigned int size, unsigned int entry_number_start, l4_utcb_t * utcb) [inline]

Set LDT segments descriptors.

Parameters

task Task to set the segment for.

ldt Pointer to LDT hardware descriptors.

num_desc Number of descriptors.

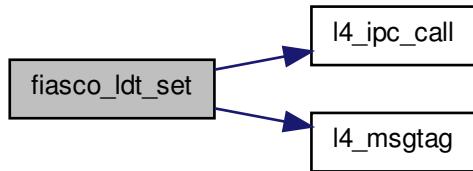
entry_number_start Entry number to start.

utcb UTCB of the caller.

Definition at line 105 of file [segment.h](#).

References `l4_ipc_call()`, `L4_IPC_NEVER`, `l4_mshtag()`, `L4_PROTO_TASK`, and `l4_msg_regs_t::mr`.

Here is the call graph for this function:



9.39.2.11 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 3 consecutive entries, starting at the value returned by [fiasco_gdt_get_entry_offset\(\)](#)

Parameters

thread Thread to set the GDT entry for.
desc Pointer to GDT descriptors.
size Size of the descriptors in bytes (multiple of 8).
entry_number_start Entry number to start (valid values: 0-2).
utcb UTCB of the caller.

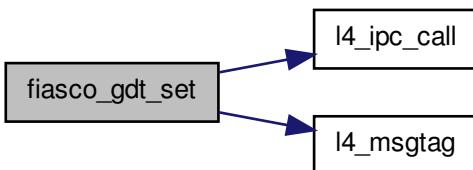
Returns

System call error

Definition at line 43 of file [segment.h](#).

References [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), [l4_mshtag\(\)](#), [L4_PROTO_THREAD](#), and [l4_msg_regs_t::mr](#).

Here is the call graph for this function:



9.39.2.12 `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

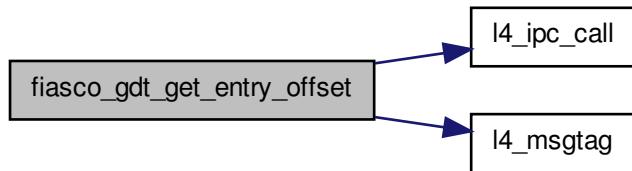
thread Thread to get info from.

utcb UTCB of the caller.

Definition at line 118 of file [segment.h](#).

References [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), [l4_mshtag\(\)](#), [L4_PROTO_THREAD](#), and [l4_msg_regs_t::mr](#).

Here is the call graph for this function:



9.40 Fiasco real time scheduling extensions

Real time scheduling extension for the Fiasco [L4](#) implementation.

Collaboration diagram for Fiasco real time scheduling extensions:



Functions

- int `l4_rt_add_time slice (l4_threadid_t dest, int prio, int time)`
Add a time slice for periodic execution.
- int `l4_rt_change_time slice (l4_threadid_t dest, int id, int prio, int time)`

Change a time slice for periodic execution.

- int `l4_rt_begin_strictly_periodic` (`l4_threadid_t dest, l4_kernel_clock_t clock`)
Start strictly periodic execution.
- int `l4_rt_begin_minimal_periodic` (`l4_threadid_t dest, l4_kernel_clock_t clock`)
Start periodic execution with minimal inter-release times.
- int `l4_rt_end_periodic` (`l4_threadid_t dest`)
Stop periodic execution.
- int `l4_rt_remove` (`l4_threadid_t dest`)
Remove all reservation scheduling contexts
This function removes all the scheduling contexts that were set up so far for the given thread.
- void `l4_rt_set_period` (`l4_threadid_t dest, l4_kernel_clock_t clock`)
Set the length of the period
This function sets the length of the period for periodic execution.
- int `l4_rt_next_reservation` (`unsigned id, l4_kernel_clock_t *clock`)
activate the next time slice (scheduling context)
- int `l4_rt_next_period` (`void`)
Wait for the next period, skipping all unused time slices.
- `l4_threadid_t l4_preemption_id` (`l4_threadid_t id`)
Return the preemption id of a thread.
- `l4_threadid_t l4_next_period_id` (`l4_threadid_t id`)
Return thread-id that flags waiting for the next period.
- int `l4_rt_generic` (`l4_threadid_t dest, l4_sched_param_t param, l4_kernel_clock_t clock`)
Generic real-time setup function.

9.40.1 Detailed Description

Real time scheduling extension for the Fiasco [L4](#) implementation.

9.40.2 Function Documentation

9.40.2.1 int `l4_rt_add_time_slice` (`l4_threadid_t dest, int prio, int time`) [\[inline\]](#)

Add a time slice for periodic execution.

Parameters

- `dest` thread to add the time slice to
- `prio` priority of the time slice
- `time` length of the time slice in microseconds

Return values**0** OK**-1** Error, one of:

- dest does not exist
- insufficient MCP (old or new prio>MCP),
- dest running in periodic mode or transitioning to
- time quantum 0 or infinite

9.40.2.2 int l4_rt_change_time_slice (l4_threadid_t dest, int id, int prio, int time) [inline]

Change a time slice for periodic execution.

Parameters*dest* thread whose timing parameters are to change*id* number of the time-slice to change (rt start at 1)*prio* new priority of the time slice*time* new length of the time slice in microseconds, 0: don't change.**Return values****0** OK**-1** Error, one of:

- dest does not exist
- insufficient MCP (old or new prio>MCP),
- time slice does not exist

This function modifies the priority and optionally the length of an existing time slice of a thread. When calling this function while the time slice is active, the effect may be delayed till the next period.

This function can be called as soon as the denoted time slice was added with [l4_rt_add_time_slice\(\)](#). Thus, the thread may have started periodic execution already, but it needs not.

9.40.2.3 int l4_rt_begin_strictly_periodic (l4_threadid_t dest, l4_kernel_clock_t clock) [inline]

Start strictly periodic execution.

Parameters*dest* thread that starts periodic execution*clock* absolute time to start.**Return values****0** OK**-1** Error, one of:

- dest does not exist
- insufficient MCP (old or new prio>MCP),

- dest running in periodic mode or transitioning to

Call this function to start the periodic execution after setting up the time slices using `l4_rt_add_time slice()` and `l4_rt_set_period()`.

By the time specified in clock thread dest must wait for the next period, e.g. by using `l4_rt_next_period()` or some other IPC with the L4_RT_NEXT_PERIOD flag enabled. Otherwise the transition to periodic mode fails.

Definition at line 81 of file `rt_sched-impl.h`.

References `l4_rt_generic()`.

Here is the call graph for this function:



9.40.2.4 `int l4_rt_begin_minimal_periodic (l4_threadid_t dest, l4_kernel_clock_t clock) [inline]`

Start periodic execution with minimal inter-release times.

Parameters

dest thread that starts periodic execution

clock absolute time to start.

Return values

0 OK

-1 Error, one of:

- dest does not exist
- insufficient MCP (old or new prio>MCP),
- dest running in periodic mode or transitioning to

Call this function to start the periodic execution after setting up the time slices using `l4_rt_add_time slice()` and `l4_rt_set_period()`.

By the time specified in clock thread dest must wait for the next period, e.g. by using `l4_rt_next_period()` or some other IPC with the L4_RT_NEXT_PERIOD flag enabled. Otherwise the transition to periodic mode fails.

Definition at line 91 of file `rt_sched-impl.h`.

References `l4_rt_generic()`.

Here is the call graph for this function:



9.40.2.5 int l4_rt_end_periodic (l4_threadid_t dest) [inline]

Stop periodic execution.

Parameters

dest thread that stops periodic execution

Return values

0 OK

-1 Error, one of:

- dest does not exist
- insufficient MCP (old or new prio>MCP),
- dest not running in periodic mode and not transitioning to

This function aborts the periodic execution of thread dest. Thread dest returns to conventional scheduling then.

Definition at line 101 of file [rt_sched-impl.h](#).

References [l4_rt_generic\(\)](#).

Here is the call graph for this function:



9.40.2.6 int l4_rt_remove (l4_threadid_t dest) [inline]

Remove all reservation scheduling contexts

This function removes all the scheduling contexts that were set up so far for the given thread.

Parameters

dest thread the scheduling contexts should be removed from

Return values

0 OK

-1 Error, one of:

- dest does not exist
- insufficient MCP
- dest running in periodic mode or transitioning to

Definition at line 110 of file [rt_sched-impl.h](#).

References [l4_rt_generic\(\)](#).

Here is the call graph for this function:



9.40.2.7 void l4_rt_set_period (l4_threadid_t dest, l4_kernel_clock_t clock) [inline]

Set the length of the period

This function sets the length of the period for periodic execution.

Parameters

dest destination thread

clock period length in microseconds. Will be rounded up by the kernel according to the timer granularity.

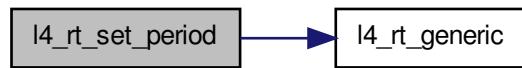
Returns

This function always succeeds.

Definition at line 119 of file [rt_sched-impl.h](#).

References [l4_rt_generic\(\)](#).

Here is the call graph for this function:



9.40.2.8 `int l4_rt_next_reservation (unsigned id, l4_kernel_clock_t * clock) [inline]`

activate the next time slice (scheduling context)

Parameters

id The ID of the time slice we think we are on (current time slice)

clock pointer to a `l4_kernel_clock_t` variable

Return values

0 OK, **clock* contains the remaining time of the time slice

-1 Error, *id* did not match current time slice

Definition at line 129 of file [rt_sched-impl.h](#).

9.40.2.9 `int l4_rt_next_period (void) [inline]`

Wait for the next period, skipping all unused time slices.

Return values

0 OK

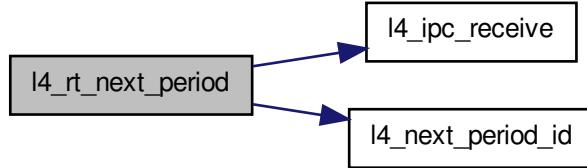
!0 IPC Error.

< 0 receive and send timeout

Definition at line 155 of file [rt_sched-impl.h](#).

References [L4_IPC_BOTH_TIMEOUT_0](#), [l4_ipc_receive\(\)](#), and [l4_next_period_id\(\)](#).

Here is the call graph for this function:



9.40.2.10 `l4_threadid_t l4_preemption_id (l4_threadid_t id) [inline]`

Return the preemption id of a thread.

Parameters

id thread

Returns

thread-id of the (virtual) preemption IPC sender

Definition at line 303 of file [rt_sched-proto.h](#).

9.40.2.11 `l4_threadid_t l4_next_period_id (l4_threadid_t id) [inline]`

Return thread-id that flags waiting for the next period.

Parameters

id original thread-id

Returns

modified id, to be used in an IPC, waiting for the next period.

Definition at line 310 of file [rt_sched-proto.h](#).

Referenced by [l4_rt_next_period\(\)](#).

Here is the caller graph for this function:



9.40.2.12 `int l4_rt_generic(l4_threadid_t dest, l4_sched_param_t param, l4_kernel_clock_t clock) [inline]`

Generic real-time setup function.

Parameters

dest destination thread

param scheduling parameter

clock clock parameter

Return values

0 OK

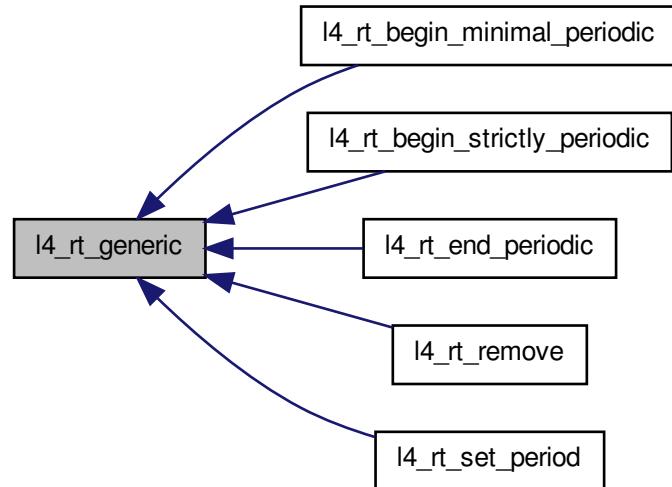
-1 Error.

This function is not meant to be used directly, it is merely used by others.

Definition at line 30 of file `rt_sched-impl.h`.

Referenced by `l4_rt_begin_minimal_periodic()`, `l4_rt_begin_strictly_periodic()`, `l4_rt_end_periodic()`, `l4_rt_remove()`, and `l4_rt_set_period()`.

Here is the caller graph for this function:



9.41 Flex pages

Flex-page related API.

Collaboration diagram for Flex pages:



Data Structures

- union `l4_fpage_t`
`L4` flexpage type.
- struct `l4_snd_fpage_t`
`Send-flex-page types.`

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 flexpage structure.

- enum { `L4_WHOLE_ADDRESS_SPACE` = 63 }

Constants for flexpages.

- enum `L4_fpage_rights` { `L4_FPAGE_RO` = 4, `L4_FPAGE_RW` = 6 }

Flex-page rights.

- enum `L4_cap_fpage_rights` { `L4_CAP_FPAGE_R` = 0x4, `L4_CAP_FPAGE_RO` = 0x4, `L4_CAP_FPAGE_RW` = 0x5 }

Cap-flex-page rights.

- enum `L4_fpage_type`

Flex-page type.

- enum `L4_fpage_control`

Flex-page map control flags.

- enum `L4_obj_fpage_ctl`

Flex-page map control for capabilities (snd_base).

- enum `l4_fpage_cacheability_opt_t` { `L4_FPAGE_CACHE_OPT` = 0x1, `L4_FPAGE_CACHEABLE` = 0x3, `L4_FPAGE_BUFFERABLE` = 0x5, `L4_FPAGE_UNCACHEABLE` = 0x1 }

Flex-page cacheability option.

- enum { `L4_WHOLE_IOADDRESS_SPACE` = 16, `L4_IOPORT_MAX` = (1L << `L4_WHOLE_IOADDRESS_SPACE`) }

Special constants for IO flex pages.

Functions

- `l4_fpage_t l4_fpage` (unsigned long address, unsigned int size, unsigned char rights) `L4_NOTHROW`
Create a memory flex page.
- `l4_fpage_t l4_fpage_all` (void) `L4_NOTHROW`
Get a flex page, describing all address spaces at once.
- `l4_fpage_t l4_fpage_invalid` (void) `L4_NOTHROW`
Get an invalid flex page.
- `l4_fpage_t l4_iofpage` (unsigned long port, unsigned int size) `L4_NOTHROW`

Create an IO-port flex page.

- `l4_fpage_t l4_obj_fpage (l4_cap_idx_t obj, unsigned int order, unsigned char rights) L4_NOTHROW`

Create a kernel-object flex page.

- `int l4_is_fpage_writable (l4_fpage_t fp) L4_NOTHROW`

Test if the flex page is writable.

- `unsigned l4_fpage_rights (l4_fpage_t f) L4_NOTHROW`

Return rights from a flex page.

- `unsigned l4_fpage_type (l4_fpage_t f) L4_NOTHROW`

Return type from a flex page.

- `unsigned l4_fpage_size (l4_fpage_t f) L4_NOTHROW`

Return size from a flex page.

- `unsigned long l4_fpage_page (l4_fpage_t f) L4_NOTHROW`

Return page from a flex page.

- `l4_fpage_t l4_fpage_set_rights (l4_fpage_t src, unsigned char new_rights) L4_NOTHROW`

Set new right in a flex page.

- `int l4_fpage_contains (l4_fpage_t fpage, l4_addr_t addr, unsigned size) 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 L4_DEFAULT_PARAM(0))`

Determine maximum flex page size of a region.

9.41.1 Detailed Description

Flex-page related API. A flex page is a page with a variable size, that can describe memory, IO-Ports (IA32 only), and sets of kernel objects.

A flex page 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 flex page also carries type and access right information for the described region. The type information selects the address space in which the flex page 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 flex page.

9.41.2 Enumeration Type Documentation

9.41.2.1 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.

Definition at line 55 of file [__l4_fpage.h](#).

9.41.2.2 anonymous enum

Constants for flexpages.

Enumerator:

- L4_WHOLE_ADDRESS_SPACE* Whole address space size.

Definition at line 86 of file [__l4_fpage.h](#).

9.41.2.3 enum L4_fpage_rights

Flex-page rights.

Enumerator:

- L4_FPAGE_RO* Read-only flex page.
- L4_FPAGE_RW* Read-write flex page.

Definition at line 104 of file [__l4_fpage.h](#).

9.41.2.4 enum L4_cap_fpage_rights

Cap-flex-page rights.

Enumerator:

- L4_CAP_FPAGE_R* Read-only cap.
- L4_CAP_FPAGE_RO* Read-only cap.
- L4_CAP_FPAGE_RW* Read-write cap.

Definition at line 117 of file [__l4_fpage.h](#).

9.41.2.5 enum l4_fpage_cacheability_opt_t

Flex-page cacheability option.

Enumerator:

L4_FPAGE_CACHE_OPT Enable the cacheability option in a send flex page.

L4_FPAGE_CACHEABLE Cacheability option to enable caches for the mapping.

L4_FPAGE_BUFFERABLE Cacheability option to enable buffered writes for the mapping.

L4_FPAGE_UNCACHEABLE Cacheability option to disable caching for the mapping.

Definition at line 164 of file [__l4_fpage.h](#).

9.41.2.6 anonymous enum

Special constants for IO flex pages.

Enumerator:

L4_WHOLE_IOPADDRESS_SPACE Whole I/O address space size.

L4_IOPORT_MAX Maximum I/O port address.

Definition at line 183 of file [__l4_fpage.h](#).

9.41.3 Function Documentation

9.41.3.1 l4_fpage_t l4_fpage (*unsigned long address*, *unsigned int size*, *unsigned char rights*) [inline]

Create a memory flex page.

Parameters

address Flex-page start address

size Flex-page size (log2), *L4_WHOLE_ADDRESS_SPACE* to specify the whole address space (with *address* 0)

rights Access rights, see [l4_fpage_rights](#)

Returns

Memory flex page

Definition at line 453 of file [__l4_fpage.h](#).

9.41.3.2 l4_fpage_t l4_fpage_all (*void*) [inline]

Get a flex page, describing all address spaces at once.

Returns

Special *all-spaces* flex page.

Definition at line 471 of file [__l4_fpage.h](#).

References [L4_WHOLE_ADDRESS_SPACE](#).

9.41.3.3 l4_fpage_t l4_fpage_invalid (void) [inline]

Get an invalid flex page.

Returns

Special *invalid* flex page.

Definition at line 477 of file [__l4_fpage.h](#).

9.41.3.4 l4_fpage_t l4_iofpage (unsigned long port, unsigned int size) [inline]

Create an IO-port flex page.

Parameters

port I/O-flex-page port base

size I/O-flex-page size, [L4_WHOLE_IOADDRESS_SPACE](#) to specify the whole I/O address space (with *port* 0)

Returns

I/O flex page

Definition at line 459 of file [__l4_fpage.h](#).

References [L4_FPAGE_ADDR_SHIFT](#), and [L4_FPAGE_RW](#).

9.41.3.5 l4_fpage_t l4_obj_fpage (l4_cap_idx_t obj, unsigned int order, unsigned char rights) [inline]

Create a kernel-object flex page.

Parameters

obj Base capability selector.

order Log2 size (number of capabilities).

rights Access rights

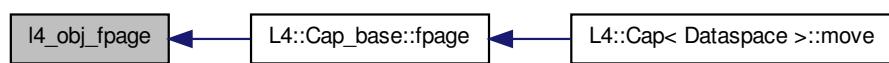
Returns

Flex page for a set of kernel objects.

Definition at line 465 of file [__l4_fpage.h](#).

Referenced by [L4::Cap_base::fpage\(\)](#).

Here is the caller graph for this function:



9.41.3.6 `int l4_is_fpage_writable (l4_fpage_t fp) [inline]`

Test if the flex page is writable.

Parameters

fp Flex page.

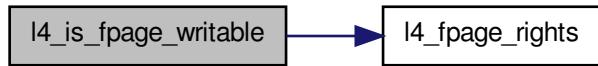
Returns

$\neq 0$ if flex page is writable, 0 if not

Definition at line 484 of file [__l4_fpage.h](#).

References [l4_fpage_rights\(\)](#).

Here is the call graph for this function:



9.41.3.7 `unsigned l4_fpage_rights (l4_fpage_t f) [inline]`

Return rights from a flex page.

Parameters

f Flex page

Returns

Size part of the given flex page.

Definition at line 403 of file [__l4_fpage.h](#).

Referenced by [l4_is_fpage_writable\(\)](#).

Here is the caller graph for this function:



9.41.3.8 `unsigned l4_fpage_type (l4_fpage_t f) [inline]`

Return type from a flex page.

Parameters

f Flex page

Returns

Type part of the given flex page.

Definition at line 409 of file [__l4_fpage.h](#).

9.41.3.9 `unsigned l4_fpage_size (l4_fpage_t f) [inline]`

Return size from a flex page.

Parameters

f Flex page

Returns

Size part of the given flex page.

Definition at line 415 of file [__l4_fpage.h](#).

9.41.3.10 `unsigned long l4_fpage_page (l4_fpage_t f) [inline]`

Return page from a flex page.

Parameters

f Flex page

Returns

Page part of the given flex page.

Definition at line 421 of file [__l4_fpage.h](#).

Referenced by [l4_fpage_contains\(\)](#).

Here is the caller graph for this function:



9.41.3.11 `l4_fpage_t l4_fpage_set_rights (l4_fpage_t src, unsigned char new_rights) [inline]`

Set new right in a flex page.

Parameters

src Flex page

new_rights New rights

Returns

Modified flex page with new rights.

Definition at line 444 of file [__l4_fpage.h](#).

References [l4_fpage_t::raw](#).

9.41.3.12 `int l4_fpage_contains (l4_fpage_t fpage, l4_addr_t addr, unsigned size) [inline]`

Test whether a given range is completely within an fpage.

Parameters

fpage Flex page

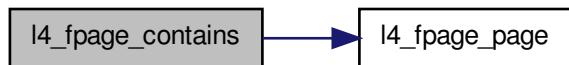
addr Address

size Size of range in log2.

Definition at line 503 of file [__l4_fpage.h](#).

References [l4_fpage_page\(\)](#).

Here is the call graph for this function:



9.41.3.13 `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 L4_DEFAULT_PARAM0) [inline]`

Determine maximum flex page size of a region.

Parameters

order Order value to start with (e.g. for memory L4_LOG2_PAGESIZE would be used)

- addr** Address to be covered by the flex page.
- min_addr** Start of region / minimal address (including).
- max_addr** End of region / maximal address (excluding).
- hotspot** (Optional) hot spot.

Returns

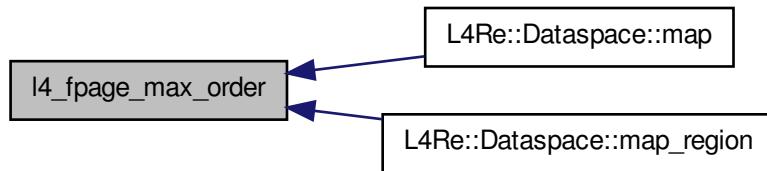
Maximum order (log2-size) possible.

Note

The start address of the flex-page can be determined with l4_trunc_size(addr, returnvalue)

Referenced by [L4Re::Dataspace::map\(\)](#), and [L4Re::Dataspace::map_region\(\)](#).

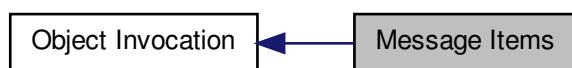
Here is the caller graph for this function:



9.42 Message Items

Message item related functions.

Collaboration diagram for Message Items:



Enumerations

- 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_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 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 for the object space.

9.42.1 Detailed Description

Message item related functions. Message items are typed items that can be transferred via IPC operations. Message items are also used to specify receive windows for typed items to be received. Message items are placed in the message registers (MRs) of the UTCB of the sending thread. Receive items are placed in the buffer registers (BRs) of the UTCB of the receiving thread.

Message items are usually two-word data structures. The first word denotes the type of the message item (for example a memory flex-page, io flex-page or object flex-page) and the second word contains information depending on the type. There is actually one exception that is a small (one word) receive buffer item for a single capability.

9.42.2 Enumeration Type Documentation

9.42.2.1 enum `l4_msg_item_consts_t`

Constants for message items.

Enumerator:

`L4_ITEM_MAP` Identify a message item as *map item*.

`L4_ITEM_CONT` Denote that the following item shall be put into the same receive item as this one.

`L4_MAP_ITEM_GRANT` Flag as *grant* instead of *map* operation.

`L4_MAP_ITEM_MAP` Flag as usual *map* operation.

`L4_RCV_ITEM_SINGLE_CAP` Mark the receive buffer to be a small receive item that describes a buffer for a single capability.

`L4_RCV_ITEM_LOCAL_ID` The receiver requests to receive a local ID instead of a mapping whenever possible.

Definition at line 193 of file `consts.h`.

9.42.3 Function Documentation

9.42.3.1 `l4_umword_t l4_map_control(l4_umword_t spot, unsigned char cache, unsigned grant) [inline]`

Create the first word for a map item for the memory space.

Parameters

spot Hot spot address, used to determine what is actually mapped when send and receive flex page have differing sizes.

cache Cacheability hints for memory flex pages. See [Cacheability options](#)

grant Indicates if it is a map or a grant item.

Returns

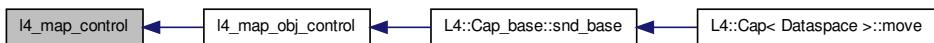
The value to be used as first word in a map item for memory.

Definition at line 490 of file [__l4_fpage.h](#).

References [L4_ITEM_MAP](#).

Referenced by [l4_map_obj_control\(\)](#).

Here is the caller graph for this function:



9.42.3.2 `l4_umword_t l4_map_obj_control(l4_umword_t spot, unsigned grant) [inline]`

Create the first word for a map item for the object space.

Parameters

spot Hot spot address, used to determine what is actually mapped when send and receive flex pages have different size.

grant Indicates if it is a map item or a grant item.

Returns

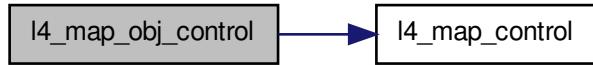
The value to be used as first word in a map item for kernel objects or IO-ports.

Definition at line 497 of file [__l4_fpage.h](#).

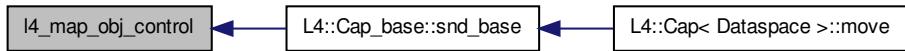
References [l4_map_control\(\)](#).

Referenced by [L4::Cap_base::snd_base\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.43 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.

Defines

- #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, init
- `#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

Typedefs

- `typedef struct l4_timeout_s l4_timeout_s`
Basic timeout specification.
- `typedef union l4_timeout_t l4_timeout_t`
Timeout pair.

Enumerations

- `enum l4_timeout_abs_validity`
Intervals of validity for absolute timeouts
Times are actually 2^x values (e.g.

Functions

- `l4_timeout_s l4_timeout_rel` (unsigned man, unsigned exp) L4_NOTHROW
Get relative timeout consisting of mantissa and exponent.
- `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_timeout_t l4_timeout` (`l4_timeout_s` snd, `l4_timeout_s` rcv) L4_NOTHROW
Combine send and receive timeout in a timeout.

- void `l4_snd_timeout (l4_timeout_s snd, l4_timeout_t *to) L4_NOTHROW`
Set send timeout in given to timeout.
- void `l4_rcv_timeout (l4_timeout_s rcv, l4_timeout_t *to) L4_NOTHROW`
Set receive timeout in given to timeout.
- `l4_kernel_clock_t l4_timeout_rel_get (l4_timeout_s to) L4_NOTHROW`
Get clock value of out timeout.
- unsigned `l4_timeout_is_absolute (l4_timeout_s to) L4_NOTHROW`
Return whether the given timeout is absolute or not.
- `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) throw ()`
Set an absolute timeout.

9.43.1 Detailed Description

All kinds of timeouts and time related functions.

9.43.2 Define Documentation

9.43.2.1 `#define L4_IPC_TIMEOUT_0 ((l4_timeout_s){0x0400})`

Timeout constants.

0 timeout

Definition at line [77](#) of file `__timeout.h`.

9.43.3 Typedef Documentation

9.43.3.1 `typedef struct l4_timeout_s l4_timeout_s`

Basic timeout specification.

Basically a floating point number with 10 bits mantissa and 5 bits exponent ($t = m \cdot 2^e$).

The timeout can also specify an absolute point in time (bit 16 == 1).

9.43.3.2 `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.

9.43.4 Enumeration Type Documentation

9.43.4.1 enum l4_timeout_abs_validity

Intervals of validity for absolute timeouts

Times are actually 2^x values (e.g.

2ms -> 2048 μ s)

Definition at line 92 of file [__timeout.h](#).

9.43.5 Function Documentation

9.43.5.1 l4_timeout_s l4_timeout_rel (*unsigned man*, *unsigned exp*) [inline]

Get relative timeout consisting of mantissa and exponent.

Parameters

man Mantissa of timeout

exp Exponent of timeout

Returns

timeout value

Definition at line 245 of file [__timeout.h](#).

9.43.5.2 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

snd_man Mantissa of send timeout.

snd_exp Exponent of send timeout.

rcv_man Mantissa of receive timeout.

rcv_exp Exponent of receive timeout.

Definition at line 210 of file [__timeout.h](#).

References [l4_timeout_t::p](#), [l4_timeout_t::rcv](#), [l4_timeout_t::snd](#), and [l4_timeout_s::t](#).

9.43.5.3 l4_timeout_t l4_timeout (*l4_timeout_s snd*, *l4_timeout_s rcv*) [inline]

Combine send and receive timeout in a timeout.

Parameters

snd Send timeout

rcv Receive timeout

Returns

[L4](#) timeout

Definition at line 221 of file [__timeout.h](#).

References [l4_timeout_t::p](#), [l4_timeout_t::rev](#), and [l4_timeout_t::snd](#).

9.43.5.4 void l4_snd_timeout (l4_timeout_s snd, l4_timeout_t * to) [inline]

Set send timeout in given to timeout.

Parameters

snd Send timeout

Return values

to [L4](#) timeout

Definition at line 231 of file [__timeout.h](#).

9.43.5.5 void l4_rcv_timeout (l4_timeout_s rcv, l4_timeout_t * to) [inline]

Set receive timeout in given to timeout.

Parameters

rcv Receive timeout

Return values

to [L4](#) timeout

Definition at line 238 of file [__timeout.h](#).

9.43.5.6 l4_kernel_clock_t l4_timeout_rel_get (l4_timeout_s to) [inline]

Get clock value of out timeout.

Parameters

to [L4](#) timeout

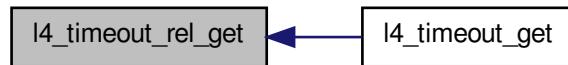
Returns

Clock value

Definition at line 252 of file [__timeout.h](#).

Referenced by [l4_timeout_get\(\)](#).

Here is the caller graph for this function:



9.43.5.7 `unsigned l4_timeout_is_absolute(l4_timeout_s to) [inline]`

Return whether the given timeout is absolute or not.

Parameters

to L4 timeout

Returns

$\neq 0$ if absolute, 0 if relative

Definition at line 261 of file `__timeout.h`.

Referenced by `l4_timeout_get()`.

Here is the caller graph for this function:



9.43.5.8 `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

cur Clock value

to L4 timeout

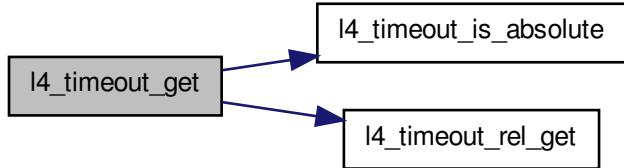
Returns

Clock sum

Definition at line 268 of file [__timeout.h](#).

References [l4_timeout_is_absolute\(\)](#), and [l4_timeout_rel_get\(\)](#).

Here is the call graph for this function:



9.43.5.9 l4_timeout_s l4_timeout_abs (l4_kernel_clock_t *pint*, int *br*) throw () [inline]

Set an absolute timeout.

Parameters

pint Point in time in clocks

br 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 362 of file [utcb.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.44 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.

9.44.1 Detailed Description

Virtual machine API for SVM.

9.45 VM API for VMX

Virtual machine API for VMX.

Collaboration diagram for VM API for VMX:



Enumerations

- enum
Additional VMCS fields.

Functions

- `unsigned l4_vm_vmx_field_len (unsigned field)`
Return length in bytes of a VMCS field.
- `void * l4_vm_vmx_field_ptr (void *vmcs, unsigned field)`
Get pointer into VMCS.

9.45.1 Detailed Description

Virtual machine API for VMX.

9.45.2 Function Documentation

9.45.2.1 `unsigned l4_vm_vmx_field_len (unsigned field) [inline]`

Return length in bytes of a VMCS field.

Parameters

field Field number.

Returns

Width of field in bytes.

Definition at line 71 of file `__vm-vmx.h`.

Referenced by `l4_vm_vmx_field_ptr()`.

Here is the caller graph for this function:



9.45.2.2 `void * l4_vm_vmx_field_ptr (void * vmcs, unsigned field) [inline]`

Get pointer into VMCS.

Parameters

`vmcs` Pointer to VMCS buffer.

`field` Field number.

Pointer to field in the VMCS.

Definition at line 79 of file `__vm-vmx.h`.

References `l4_vm_vmx_field_len()`.

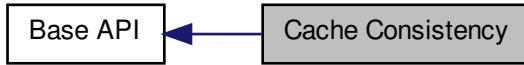
Here is the call graph for this function:



9.46 Cache Consistency

Various functions for cache consistency.

Collaboration diagram for Cache Consistency:



Functions

- void [l4_cache_clean_data](#) (unsigned long *start*, unsigned long *end*) throw ()
Cache clean a range in D-cache.
- void [l4_cache_flush_data](#) (unsigned long *start*, unsigned long *end*) throw ()
Cache flush a range.
- void [l4_cache_inv_data](#) (unsigned long *start*, unsigned long *end*) throw ()
Cache invalidate a range.
- void [l4_cache_coherent](#) (unsigned long *start*, unsigned long *end*) throw ()
Make memory coherent between I-cache and D-cache.
- void [l4_cache_dma_coherent](#) (unsigned long *start*, unsigned long *end*) throw ()
Make memory coherent for use with external memory.
- void [l4_cache_dma_coherent_full](#) (void) throw ()
Make memory coherent for use with external memory.

9.46.1 Detailed Description

Various functions for cache consistency. #include <l4/sys/cache.h>

9.46.2 Function Documentation

9.46.2.1 void [l4_cache_clean_data](#) (unsigned long *start*, unsigned long *end*) throw () `[inline]`

Cache clean a range in D-cache.

Parameters

start Start of range (inclusive)

end End of range (exclusive)

Examples:

[examples/libs/l4re/c++/shared_ds/ds_clnt.cc](#).

Definition at line 84 of file [cache.h](#).

9.46.2.2 void l4_cache_flush_data (unsigned long *start*, unsigned long *end*) throw () [inline]

Cache flush a range.

Parameters

start Start of range (inclusive)

end End of range (exclusive)

Definition at line 91 of file [cache.h](#).

9.46.2.3 void l4_cache_inv_data (unsigned long *start*, unsigned long *end*) throw () [inline]

Cache invalidate a range.

Parameters

start Start of range (inclusive)

end End of range (exclusive)

Definition at line 98 of file [cache.h](#).

9.46.2.4 void l4_cache_coherent (unsigned long *start*, unsigned long *end*) throw () [inline]

Make memory coherent between I-cache and D-cache.

Parameters

start Start of range (inclusive)

end End of range (exclusive)

Definition at line 105 of file [cache.h](#).

9.46.2.5 void l4_cache_dma_coherent (unsigned long *start*, unsigned long *end*) throw () [inline]

Make memory coherent for use with external memory.

Parameters

start Start of range (inclusive)

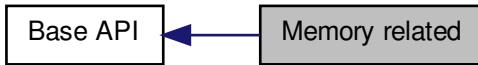
end End of range (exclusive)

Definition at line 112 of file [cache.h](#).

9.47 Memory related

Memory related constants, data types and functions.

Collaboration diagram for Memory related:



Defines

- `#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 20`
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) throw ()`
Round an address down to the next lower page boundary.
- `l4_addr_t l4_trunc_size (l4_addr_t address, unsigned char bits) throw ()`
Round an address down to the next lower flex page with size bits.
- `l4_addr_t l4_round_page (l4_addr_t address) throw ()`
Round address up to the next page.
- `l4_addr_t l4_round_size (l4_addr_t address, unsigned char bits) throw ()`
Round address up to the next flex page with bits size.

9.47.1 Detailed Description

Memory related constants, data types and functions.

9.47.2 Define Documentation

9.47.2.1 `#define L4_PAGEMASK`

Mask for the page number.

Note

The most significant bits are set.

Definition at line 285 of file `consts.h`.

9.47.2.2 `#define L4_LOG2_PAGESIZE`

Number of bits used for page offset.

Size of page in log2.

Definition at line 294 of file `consts.h`.

Referenced by `L4Re::Dataspace::map()`.

9.47.2.3 `#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 303 of file `consts.h`.

9.47.2.4 `#define L4_SUPERPAGEMASK`

Mask for the number of a large page.

Note

The most significant bits are set.

Definition at line 312 of file `consts.h`.

9.47.2.5 `#define L4_LOG2_SUPERPAGESIZE`

Number of bits used as offset for a large page.

Size of large page in log2

Definition at line 320 of file `consts.h`.

9.47.3 Enumeration Type Documentation

9.47.3.1 `enum l4_addr_consts_t`

Address related constants.

Enumerator:

`L4_INVALID_ADDR` Invalid address.

Definition at line 368 of file `consts.h`.

9.47.4 Function Documentation

9.47.4.1 `l4_addr_t l4_trunc_page(l4_addr_t address) throw() [inline]`

Round an address down to the next lower page boundary.

Parameters

address The address to round.

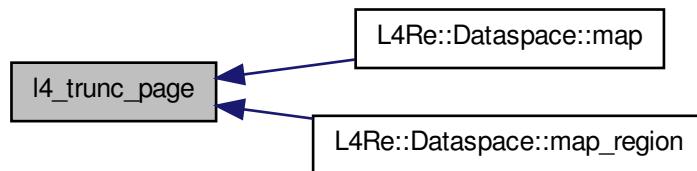
Examples:

[examples/libs/l4re/c++/mem_alloc/ma+rm.cc](#), and [examples/libs/l4re/c/ma+rm.c](#).

Definition at line 329 of file `consts.h`.

Referenced by `L4Re::Dataspace::map()`, and `L4Re::Dataspace::map_region()`.

Here is the caller graph for this function:



9.47.4.2 l4_addr_t l4_trunc_size (l4_addr_t *address*, unsigned char *bits*) throw () [inline]

Round an address down to the next lower flex page with size *bits*.

Parameters

address The address to round.

bits The size of the flex page (log2).

Definition at line 340 of file `consts.h`.

Referenced by `L4Re::Dataspace::map_region()`.

Here is the caller graph for this function:



9.47.4.3 `l4_addr_t l4_round_page (l4_addr_t address) throw () [inline]`

Round address up to the next page.

Parameters

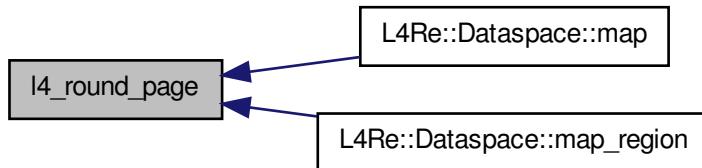
address The address to round up.

Definition at line 350 of file [consts.h](#).

References [L4_PAGESIZE](#).

Referenced by [L4Re::Dataspace::map\(\)](#), and [L4Re::Dataspace::map_region\(\)](#).

Here is the caller graph for this function:



9.47.4.4 `l4_addr_t l4_round_size (l4_addr_t address, unsigned char bits) throw () [inline]`

Round address up to the next flex page with *bits* size.

Parameters

address The address to round up to the next flex page.

bits The size of the flex page (log2).

Definition at line 361 of file [consts.h](#).

Referenced by [L4Re::Dataspace::map_region\(\)](#).

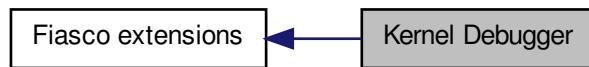
Here is the caller graph for this function:



9.48 Kernel Debugger

Kernel debugger related functionality.

Collaboration diagram for Kernel Debugger:



Data Structures

- class [L4::Debugger](#)
Debugger interface.

Defines

- `#define enter_kdebug(text)`
Enter L4 kernel debugger.
- `#define asm_enter_kdebug(text)`
Enter L4 kernel debugger (plain assembler version).
- `#define kd_display(text)`
Show message with L4 kernel debugger, but do not enter debugger.
- `#define ko(c)`
Output character with L4 kernel debugger.
- `#define enter_kdebug(text)`
Enter L4 kernel debugger.
- `#define asm_enter_kdebug(text)`
Enter L4 kernel debugger (plain assembler version).
- `#define kd_display(text)`
Show message with L4 kernel debugger, but do not enter debugger.
- `#define ko(c)`
Output character with L4 kernel debugger.

Functions

- `l4_msgtag_t l4_debugger_set_object_name (l4_cap_idx_t cap, const char *name) throw ()`
The string name of kernel object.
- `unsigned long l4_debugger_global_id (l4_cap_idx_t cap) throw ()`
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) throw ()`
Get the globally unique ID of the object behind the kobject pointer.
- `void uchar (char c) throw ()`
Print character.
- `void outstring (const char *text) throw ()`
Print character string.
- `void outnstring (char const *text, unsigned len) throw ()`
Print character string.
- `void outhex32 (int number) throw ()`
Print 32 bit number (hexadecimal).
- `void outhex20 (int number) throw ()`
Print 20 bit number (hexadecimal).
- `void outhex16 (int number) throw ()`
Print 16 bit number (hexadecimal).
- `void outhex12 (int number) throw ()`
Print 12 bit number (hexadecimal).
- `void outhex8 (int number) throw ()`
Print 8 bit number (hexadecimal).
- `void outdec (int number) throw ()`
Print number (decimal).
- `char l4kd_inchar (void) throw ()`
Read character from console, non blocking.

9.48.1 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 <l4/sys/debugger.h>
```

9.48.2 Define Documentation

9.48.2.1 #define enter_kdebug(*text*)

Value:

```
asm(\n    "int $3 \n\t\"\n    "jmp 1f \n\t\"\n    ".ascii \" text \"\n    \"1: \n\t\"\n)
```

Enter [L4](#) kernel debugger.

Parameters

text Text to be shown at kernel debugger prompt

Examples:

[examples/sys/singlestep/main.c](#).

Definition at line [41](#) of file [kdebug.h](#).

Referenced by [fiasco_tbuf_get_status_phys\(\)](#).

9.48.2.2 #define asm_enter_kdebug(*text*)

Value:

```
"int $3 \n\t\"\n    "jmp 1f \n\t\"\n    ".ascii \" text \"\n    \"1: \n\t\"
```

Enter [L4](#) kernel debugger (plain assembler version).

Parameters

text Text to be shown at kernel debugger prompt

Definition at line [63](#) of file [kdebug.h](#).

9.48.2.3 #define kd_display(*text*)

Value:

```
asm(\n    "int $3 \n\t\"\n    "nop \n\t\"\n    "jmp 1f \n\t\"\n    ".ascii \" text \"\n    \"1: \n\t\"\n)
```

Show message with [L4](#) kernel debugger, but do not enter debugger.

Parameters

text Text to be shown

Definition at line [76](#) of file [kdebug.h](#).

9.48.2.4 #define ko(*c*)

Value:

```
asm( \
    "int $3 \n\t" \
    "cmpb %0,%al \n\t" \
    : /* No output */ \
    : "N" (c) \
)
```

Output character with [L4](#) kernel debugger.

Parameters

c Character to be shown

Definition at line [92](#) of file [kdebug.h](#).

9.48.2.5 #define enter_kdebug(*text*)

Value:

```
asm(\
    "int $3 \n\t" \
    "jmp 1f \n\t" \
    ".ascii \" text \"\n\t" \
    "1: \n\t" \
)
```

Enter [L4](#) kernel debugger.

Parameters

text Text to be shown at kernel debugger prompt

Definition at line [41](#) of file [kdebug.h](#).

9.48.2.6 #define asm_enter_kdebug(*text*)

Value:

```
"int $3 \n\t" \
"jmp 1f \n\t" \
".ascii \" text \"\n\t" \
"1: \n\t"
```

Enter [L4](#) kernel debugger (plain assembler version).

Parameters

text Text to be shown at kernel debugger prompt

Definition at line [63](#) of file [kdebug.h](#).

9.48.2.7 #define kd_display(*text*)

Value:

```
asm(\n    "int    $3    \n\t\"\n    \"nop    \n\t\"\n    \"jmp    1f    \n\t\"\n    \".ascii \\\" text  \"\\\"\\n\\t\"\n    \"1:    \n\t\"\n)
```

Show message with [L4](#) kernel debugger, but do not enter debugger.

Parameters

text Text to be shown

Definition at line [76](#) of file [kdebug.h](#).

9.48.2.8 #define ko(*c*)

Value:

```
asm(\n    "int    $3    \n\t\"    \\\n    \"cmpb  %0,%al \n\t\"    \\\n    : /* No output */\n    : \"N\"  (c)    \\\n)
```

Output character with [L4](#) kernel debugger.

Parameters

c Character to be shown

Definition at line [92](#) of file [kdebug.h](#).

9.48.3 Function Documentation

9.48.3.1 l4_mshtag_t l4_debugger_set_object_name (l4_cap_idx_t *cap*, const char * *name*) throw () [inline]

The string name of kernel object.

Parameters*cap* Capability*name* Name

This is a debugging facility, the call might be invalid.

Examples:

[examples/sys/aliens/main.c](#).

9.48.3.2 `unsigned long l4_debugger_global_id (l4_cap_idx_t cap) throw () [inline]`

Get the globally unique ID of the object behind a capability.

Parameters*cap* Capability**Returns**

~0UL on non-valid capability, ID otherwise

This is a debugging facility, the call might be invalid.

Definition at line [297](#) of file [debugger.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:

**9.48.3.3 `unsigned long l4_debugger_kobj_to_id (l4_cap_idx_t cap, l4_addr_t kobjp) throw () [inline]`**

Get the globally unique ID of the object behind the kobject pointer.

Parameters*cap* Capability*kobjp* Kobject pointer**Returns**

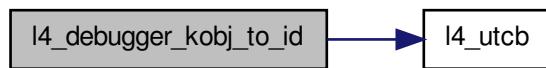
~0UL on non-valid capability or invalid kobject pointer, ID otherwise

This is a debugging facility, the call might be invalid.

Definition at line 303 of file [debugger.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.48.3.4 void outchar (char *c*) throw () [inline]

Print character.

Parameters

c Character

9.48.3.5 void outstring (const char * *text*) throw () [inline]

Print character string.

Parameters

text Character string

text String

Examples:

[examples/sys/aliens/main.c](#).

9.48.3.6 void outnstring (char const * *text*, unsigned *len*) throw () [inline]

Print character string.

Parameters

text Character string

len Number of characters

text String

len Number of characters

Examples:

[examples/sys/aliens/main.c](#).

9.48.3.7 void outhex32 (int *number*) throw () [inline]

Print 32 bit number (hexadecimal).

Parameters

number 32 bit number

9.48.3.8 void outhex20 (int *number*) throw () [inline]

Print 20 bit number (hexadecimal).

Parameters

number 20 bit number

9.48.3.9 void outhex16 (int *number*) throw () [inline]

Print 16 bit number (hexadecimal).

Parameters

number 16 bit number

9.48.3.10 void outhex12 (int *number*) throw () [inline]

Print 12 bit number (hexadecimal).

Parameters

number 12 bit number

9.48.3.11 void outhex8 (int *number*) throw () [inline]

Print 8 bit number (hexadecimal).

Parameters

number 8 bit number

9.48.3.12 void outdec (int *number*) throw () [inline]

Print number (decimal).

Parameters

number Number

9.48.3.13 `char l4kd_inchar (void) throw () [inline]`

Read character from console, non blocking.

Returns

Input character, -1 if no character to read

9.49 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_EAGAIN` = 11, `L4_ENOMEM` = 12, `L4_EACCESS` = 13, `L4_EBUSY` = 16,

`L4_EEXIST` = 17, `L4_ENODEV` = 19, `L4_EINVAL` = 22, `L4_ERANGE` = 34,

`L4_ENAMETOOLONG` = 36, `L4_ENOSYS` = 38, `L4_EBADPROTO` = 39, `L4_EADDRNOTAVAIL` = 99,

`L4_ERRNOMAX` = 100, `L4_ENOREPLY` = 1000, `L4_EIPC_LO` = 2000, `L4_EIPC_HI` = 2000 + 0x1f
 }

L4 error codes.

9.49.1 Detailed Description

Common error codes. #include <14/sys/err.h>

9.49.2 Enumeration Type Documentation

9.49.2.1 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_EAGAIN Try again.
L4_ENOMEM No memory.
L4_EACCESS Permission denied.
L4_EBUSY Object currently busy, try later.
L4_EEXIST Already exists.
L4_ENODEV No such thing.
L4_EINVAL Invalid argument.
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_EIPC_LO Communication error-range low.
L4_EIPC_HI Communication error-range high.

Definition at line 41 of file [err.h](#).

9.50 Factory

A factory is used to create all kinds of kernel objects.

Collaboration diagram for Factory:



Data Structures

- class [L4::Factory](#)
C++ [L4 Factory](#), to create all kinds of kernel objects.

Functions

- `l4_mshtag_t l4_factory_create_task (l4_cap_idx_t factory, l4_cap_idx_t target_cap, l4_fpage_t const utcb_area) throw ()`
Create a new task.
- `l4_mshtag_t l4_factory_create_thread (l4_cap_idx_t factory, l4_cap_idx_t target_cap) throw ()`
Create a new thread.
- `l4_mshtag_t l4_factory_create_factory (l4_cap_idx_t factory, l4_cap_idx_t target_cap, unsigned long limit) throw ()`
Create a new factory.
- `l4_mshtag_t l4_factory_create_gate (l4_cap_idx_t factory, l4_cap_idx_t target_cap, l4_cap_idx_t thread_cap, l4_umword_t label) throw ()`
Create a new IPC gate.
- `l4_mshtag_t l4_factory_create_irq (l4_cap_idx_t factory, l4_cap_idx_t target_cap) throw ()`
Create a new IRQ.
- `l4_mshtag_t l4_factory_create_vm (l4_cap_idx_t factory, l4_cap_idx_t target_cap) throw ()`
Create a new virtual machine.

9.50.1 Detailed Description

A factory is used to create all kinds of kernel objects. #include <14/sys/factory.h>

A factory provides the means to create all kinds of kernel objects. 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.

9.50.2 Function Documentation

9.50.2.1 `l4_mshtag_t l4_factory_create_task (l4_cap_idx_t factory, l4_cap_idx_t target_cap, l4_fpage_t const utcb_area) throw () [inline]`

Create a new task.

Parameters

`factory` Capability selector for factory to use for creation.

`target_cap` Capability selector for the root capability of the new task.

`utcb_area` Flexpage that describes the area for the UTCBs of the new task

Note

The size of the UTCB area specifies indirectly the maximum number of UTCBs available for this task and cannot be changed afterwards.

Returns

Syscall return tag

See also

[Task](#)

Definition at line 306 of file [factory.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.50.2.2 `l4_msntag_t l4_factory_create_thread (l4_cap_idx_t factory, l4_cap_idx_t target_cap)` throw () [inline]

Create a new thread.

Parameters

factory Capability selector for factory to use for creation.

target_cap Capability selector for the root capability of the new thread.

Returns

Syscall return tag

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 313 of file [factory.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.50.2.3 `l4_mshtag_t l4_factory_create_factory (l4_cap_idx_t factory, l4_cap_idx_t target_cap, unsigned long limit) throw () [inline]`

Create a new factory.

Parameters

factory Capability selector for factory to use for creation.

target_cap Capability selector for the root capability of the new factory.

limit Limit for the new factory in bytes

Note

The limit of the new factory is subtracted from the available amount of the factory used for creation.

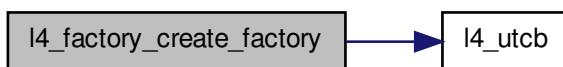
Returns

Syscall return tag

Definition at line 320 of file [factory.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.50.2.4 `l4_mshtag_t l4_factory_create_gate (l4_cap_idx_t factory, l4_cap_idx_t target_cap, l4_cap_idx_t thread_cap, l4_umword_t label) throw () [inline]`

Create a new IPC gate.

Parameters

- factory* Capability selector for factory to use for creation.
- target_cap* Capability selector for the root capability of the new IPC gate.
- thread_cap* Thread to bind the gate to
- label* Label of the gate

Returns

Syscall return tag

See also

[IPC-Gate API](#)

Definition at line 328 of file [factory.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.50.2.5 `l4_msgtag_t l4_factory_create_irq (l4_cap_idx_t factory, l4_cap_idx_t target_cap)` throw () [inline]

Create a new IRQ.

Parameters

- factory* Capability selector for factory to use for creation.
- target_cap* Capability selector for the root capability of the new IRQ.

Returns

Syscall return tag

See also

[IRQs](#)

Examples:

[examples/sys/isr/main.c](#)

Definition at line 336 of file [factory.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.50.2.6 `l4_mshtag_t l4_factory_create_vm (l4_cap_idx_t factory, l4_cap_idx_t target_cap)`
`throw () [inline]`

Create a new virtual machine.

Parameters

factory Capability selector for factory to use for creation.

target_cap Capability selector for the root capability of the new VM.

Returns

Syscall return tag

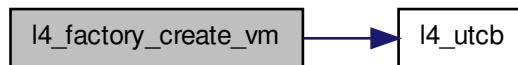
See also

[Virtual Machines](#)

Definition at line 343 of file [factory.h](#).

References [l4_utcb\(\)](#).

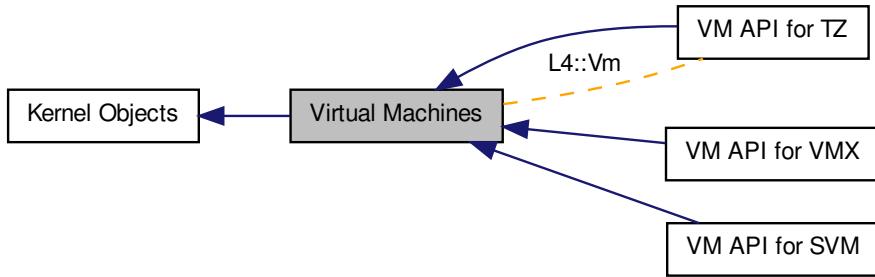
Here is the call graph for this function:



9.51 Virtual Machines

Virtual Machine API.

Collaboration diagram for Virtual Machines:



Data Structures

- class [L4::Vm](#)
Virtual machine.

Modules

- [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.

9.51.1 Detailed Description

Virtual Machine API.

9.52 Interrupt controller

The ICU class.

Collaboration diagram for Interrupt controller:



Data Structures

- struct [l4_icu_info_t](#)
Info structure for an ICU.
- class [L4::Icu](#)
C++ version of an interrupt controller.
- class [L4::Icu::Info](#)
Info 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_mshtag_t](#) [l4_icu_bind](#) ([l4_cap_idx_t](#) icu, unsigned irqnum, [l4_cap_idx_t](#) irq) throw ()
Bind an interrupt vector of an interrupt controller to an interrupt object.
- [l4_mshtag_t](#) [l4_icu_unbind](#) ([l4_cap_idx_t](#) icu, unsigned irqnum, [l4_cap_idx_t](#) irq) throw ()
Remove binding of an interrupt vector from the interrupt controller object.
- [l4_mshtag_t](#) [l4_icu_set_mode](#) ([l4_cap_idx_t](#) icu, unsigned irqnum, [l4_umword_t](#) mode) throw ()
Set mode of interrupt.
- [l4_mshtag_t](#) [l4_icu_info](#) ([l4_cap_idx_t](#) icu, [l4_icu_info_t](#) *info) throw ()

Get info about capabilites of ICU.

- [l4_msntag_t l4_icu_msi_info \(l4_cap_idx_t icu, unsigned irqnum, l4_umword_t *msg\) throw \(\)](#)
Get MSI info about IRQ.
- [l4_msntag_t l4_icu_unmask \(l4_cap_idx_t icu, unsigned irqnum, l4_umword_t *label, l4_timeout_t to\) throw \(\)](#)
Unmask an IRQ vector.
- [l4_msntag_t l4_icu_mask \(l4_cap_idx_t icu, unsigned irqnum, l4_umword_t *label, l4_timeout_t to\) throw \(\)](#)
Mask an IRQ vector.

9.52.1 Detailed Description

The ICU class. #include <[l4/sys/icu.h](#)>

9.52.2 Typedef Documentation

9.52.2.1 [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\(\)](#).

9.52.3 Enumeration Type Documentation

9.52.3.1 [enum L4_icu_flags](#)

Flags for IRQ numbers used for the ICU.

Enumerator:

[L4_ICU_FLAG_MSI](#) 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 44 of file [icu.h](#).

9.52.4 Function Documentation

9.52.4.1 [l4_msntag_t l4_icu_bind \(l4_cap_idx_t icu, unsigned irqnum, l4_cap_idx_t irq \) throw \(\) \[inline\]](#)

Bind an interrupt vector of an interrupt controller to an interrupt object.

Parameters

icu ICU to use.

irqnum IRQ vector at the ICU.

irq IRQ capability to bind the IRQ to.

Returns

Syscall return tag

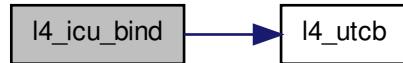
Examples:

[examples/sys/isr/main.c](#).

Definition at line [418](#) of file [icu.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.52.4.2 `l4_mshtag_t l4_icu_unbind (l4_cap_idx_t icu, unsigned irqnum, l4_cap_idx_t irq) throw () [inline]`

Remove binding of an interrupt vector from the interrupt controller object.

Parameters

icu ICU to use.

irqnum IRQ vector at the ICU.

irq IRQ object to remove from the ICU.

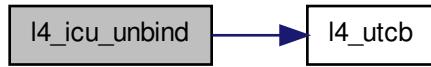
Returns

Syscall return tag

Definition at line [422](#) of file [icu.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.52.4.3 l4_msgtag_t l4_icu_set_mode (l4_cap_idx_t *icu*, unsigned *irqnum*, l4_umword_t *mode*) throw () [inline]

Set mode of interrupt.

Parameters

icu ICU to use.

irqnum IRQ vector at the ICU.

mode Mode, see L4_irq_flow_type.

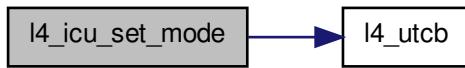
Returns

Syscall return tag

Definition at line 444 of file [icu.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.52.4.4 l4_msgtag_t l4_icu_info (l4_cap_idx_t *icu*, l4_icu_info_t * *info*) throw () [inline]

Get info about capabilites of ICU.

Parameters

icu ICU to use.

info Pointer to an info structure to be filled with information.

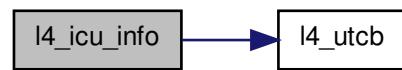
Returns

Syscall return tag

Definition at line 426 of file [icu.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.52.4.5 l4_msntag_t l4_icu_msi_info (l4_cap_idx_t *icu*, unsigned *irqnum*, l4_umword_t * *msg*) throw () [inline]

Get MSI info about IRQ.

Parameters

icu ICU to use.

irqnum IRQ vector at the ICU.

msg Pointer to a word to receive the message that must be used for the PCI devices MSI message.

Returns

Syscall return tag

Definition at line 430 of file [icu.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.52.4.6 `l4_msntag_t l4_icu_unmask(l4_cap_idx_t icu, unsigned irqnum, l4_umword_t * label, l4_timeout_t to) throw() [inline]`

Unmask an IRQ vector.

Parameters

icu ICU to use.
irqnum IRQ vector at the ICU.
label If non-NULL the function also waits for the next message.
to Timeout for message to ICU, if unsure use L4_IPC_NEVER.

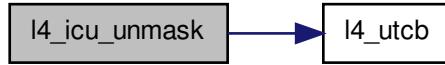
Returns

Syscall return tag

Definition at line 434 of file [icu.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.52.4.7 `l4_msntag_t l4_icu_mask(l4_cap_idx_t icu, unsigned irqnum, l4_umword_t * label, l4_timeout_t to) throw() [inline]`

Mask an IRQ vector.

Parameters

icu ICU to use.
irqnum IRQ vector at the ICU.
label If non-NULL the function also waits for the next message.
to Timeout for message to ICU, if unsure use L4_IPC_NEVER.

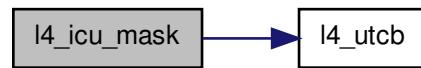
Returns

Syscall return tag

Definition at line 439 of file [icu.h](#).

References [l4_utcb\(\)](#).

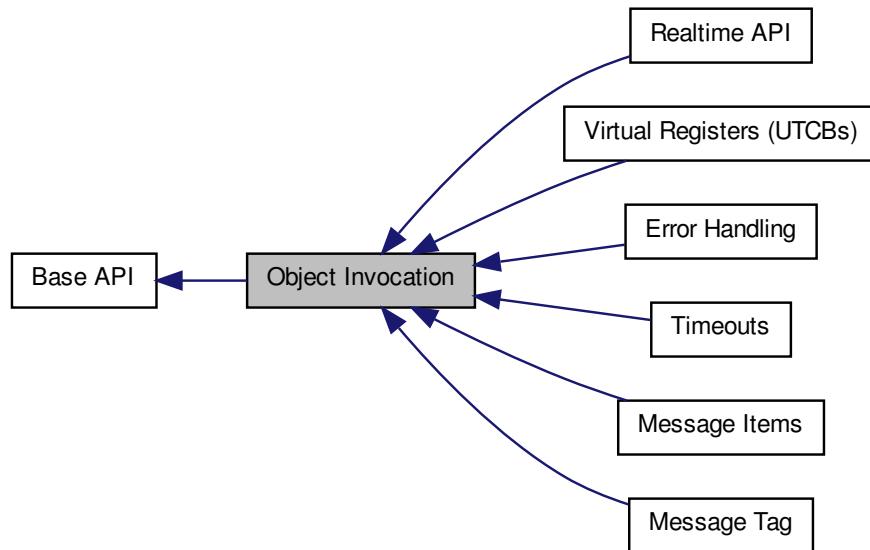
Here is the call graph for this function:



9.53 Object Invocation

API for L4 object invocation.

Collaboration diagram for Object Invocation:



Modules

- **Message Items**

Message item related functions.

- **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) throw ()

*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) throw ()

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) throw ()

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) throw ()

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) throw ()

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) throw ()

Send a message and do an open wait.

- `l4_mshtag_t l4_ipc (l4_cap_idx_t dest, l4_utcb_t *utcb, l4_umword_t flags, l4_umword_t slabel, l4_mshtag_t tag, l4_umword_t *rlabel, l4_timeout_t timeout) throw ()`

Generic L4 object invocation.

- `l4_mshtag_t l4_ipc_sleep (l4_timeout_t timeout) throw ()`

Sleep for an amount of time.

- `int l4_sndfpage_add (l4_fpage_t const snd_fpage, unsigned long snd_base, l4_mshtag_t *tag) throw ()`

Add a flex-page to be sent to the UTCB.

9.53.1 Detailed Description

API for L4 object invocation. `#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](#)

9.53.2 Enumeration Type Documentation

9.53.2.1 enum l4_syscall_flags_t

Capability selector flags.

These flags determine the concrete operation when a kernel object is invoked.

Enumerator:

`L4_SYSF_NONE` Default flags (call to a kernel object). Using this value as flags in the capability selector for an invocation indicates a call (send and wait for a reply).

`L4_SYSF_SEND` Send-phase flag. Setting this flag in a capability selector induces a send phase, this means a message is send to the object denoted by the capability. For receive phase see [L4_SYSF_RECV](#).

`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 *open wait*. *Open wait* means that the invoking thread shall wait for a message from any possible sender and *not* 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 [45](#) of file [consts.h](#).

9.53.3 Function Documentation

9.53.3.1 `l4_mshtag_t l4_ipc_send (l4_cap_idx_t dest, l4_utcb_t * utcb, l4_mshtag_t tag, l4_timeout_t timeout) throw () [inline]`

Send a message to an object (do **not** wait for a reply).

Parameters

dest Capability selector for the destination object.

utcb UTCB of the caller.

tag Descriptor for the message to be sent.

timeout Timeout pair (see [l4_timeout_t](#)) only send part is relevant.

Returns

result tag

A message is sent to the destination object. There is no receive phase included. The invoker continues working after sending the message.

Attention

This is a special-purpose message transfer, objects usually support only invocation via [l4_ipc_call\(\)](#).

Examples:

[examples/sys/start-with-exc/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

Definition at line [135](#) of file [ipc.h](#).

References [L4_SYSF_SEND](#), and [l4_mshtag_t::raw](#).

Referenced by [L4::Ipc::Ostream::send\(\)](#).

Here is the caller graph for this function:



9.53.3.2 `l4_mshtag_t l4_ipc_wait (l4_utcb_t * utcb, l4_umword_t * label, l4_timeout_t timeout) throw () [inline]`

Wait for an incoming message from any possible sender.

Parameters

utcb UTCB of the caller.

Return values

label Label assigned to the source object (IPC gate or IRQ).

Parameters

timeout 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\(\)](#).

Examples:

[examples/sys/ ipc/ ipc_example.c](#)

Definition at line [168](#) of file `ipc.h`.

References `L4_INVALID_CAP`, `L4_SYSF_WAIT`, and `l4_mshtag_t::raw`.

Referenced by [L4::Ipc::Istream::wait\(\)](#).

Here is the caller graph for this function:



9.53.3.3 `l4_mshtag_t l4_ipc_receive (l4_cap_idx_t object, l4_utcb_t * utcb, l4_timeout_t timeout) throw () [inline]`

Wait for a message from a specific source.

Parameters

object Object to receive a message from.

timeout Timeout pair (see [l4_timeout_t](#), only the receive part matters).

utcb UTCB of the caller.

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.

Examples:

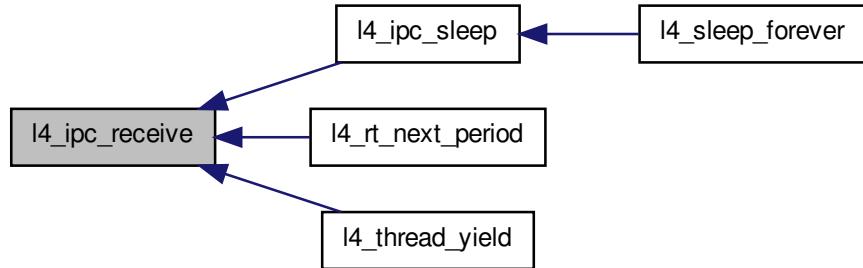
[examples/sys/aliens/main.c](#), [examples/sys/singlestep/main.c](#), [examples/sys/start-with-exc/main.c](#), and [examples/sys/utcb-ipc/main.c](#).

Definition at line [204](#) of file [ipc.h](#).

References [L4_SYSF_RECV](#), and [l4_mshtag_t::raw](#).

Referenced by [l4_ipc_sleep\(\)](#), [l4_rt_next_period\(\)](#), and [l4_thread_yield\(\)](#).

Here is the caller graph for this function:



9.53.3.4 `l4_mshtag_t l4_ipc_call (l4_cap_idx_t object, l4_utcb_t * utcb, l4_mshtag_t tag, l4_timeout_t timeout) throw () [inline]`

Object call (usual invocation).

Parameters

object Capability selector for the object to call.

utcb UTCB of the caller.

tag Message tag to describe the message to be sent.

timeout 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.

Examples:

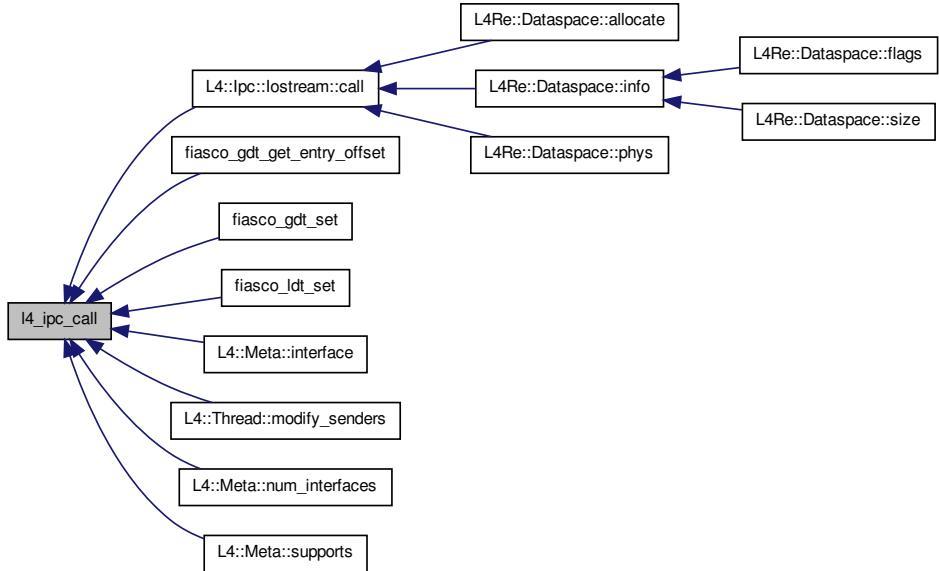
[examples/sys/aliens/main.c](#), [examples/sys/ ipc/ ipc_example.c](#), and [examples/sys/ singlestep/main.c](#).

Definition at line 34 of file [ipc.h](#).

References [L4_SYSF_CALL](#), and [l4_mshtag_t::raw](#).

Referenced by [L4::Ipc::Iostream::call\(\)](#), [fiasco_gdt_get_entry_offset\(\)](#), [fiasco_gdt_set\(\)](#), [fiasco_ldt_set\(\)](#), [L4::Meta::interface\(\)](#), [L4::Thread::modify_senders\(\)](#), [L4::Meta::num_interfaces\(\)](#), and [L4::Meta::supports\(\)](#).

Here is the caller graph for this function:



9.53.3.5 `l4_mshtag_t l4_ipc_reply_and_wait (l4_utcb_t * utcb, l4_mshtag_t tag, l4_umword_t * label, l4_timeout_t timeout) throw () [inline]`

Reply and wait operation (uses the *reply* capability).

Parameters

tag Describes the message to be sent as reply.
utcb UTCB of the caller.

Return values

label Label assigned to the source object of the received message.

Parameters

timeout 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.

Examples:

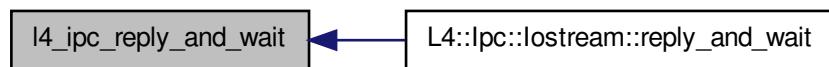
[examples/sys/ ipc/ ipc_example.c.](#)

Definition at line [65](#) of file [ipc.h](#).

References [L4_INVALID_CAP](#), [L4_SYSF_REPLY_AND_WAIT](#), and [l4_mshtag_t::raw](#).

Referenced by [L4::Ipc::Iostream::reply_and_wait\(\)](#).

Here is the caller graph for this function:



9.53.3.6 l4_mshtag_t l4_ipc_send_and_wait (l4_cap_idx_t dest, l4_utcb_t * utcb, l4_mshtag_t tag, l4_umword_t * label, l4_timeout_t timeout) throw() [inline]

Send a message and do an open wait.

Parameters

dest Object to send a message to.

utcb UTCB of the caller.

tag Describes the message that shall be sent.

Return values

label Label assigned to the source object of the receive phase.

Parameters

timeout 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.

Definition at line [100](#) of file [ipc.h](#).

References [L4_SYSF_SEND_AND_WAIT](#), and [l4_mshtag_t::raw](#).

9.53.3.7 `l4_mshtag_t l4_ipc (l4_cap_idx_t dest, l4_utcb_t * utcb, l4_umword_t flags, l4_umword_t slabel, l4_mshtag_t tag, l4_umword_t * rlabel, l4_timeout_t timeout) throw () [inline]`

Generic L4 object invocation.

Parameters

dest Destination object.

utcb UTCB of the caller.

flags Invocation flags (see [l4_syscall_flags_t](#)).

slabel Send label if applicable (may be seen by the receiver).

tag Sending message tag.

Return values

rlabel Receiving label.

Parameters

timeout Timeout pair (see [l4_timeout_t](#)).

Returns

return tag

Definition at line 240 of file [ipc.h](#).

References [l4_mshtag_t::raw](#).

9.53.3.8 `l4_mshtag_t l4_ipc_sleep (l4_timeout_t timeout) throw () [inline]`

Sleep for an amount of time.

Parameters

timeout 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\(\)](#).

Definition at line 28 of file [ipc-impl.h](#).

References [L4_INVALID_CAP](#), and [l4_ipc_receive\(\)](#).

Referenced by [l4_sleep_forever\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.53.3.9 `int l4_sndfpage_add (l4_fpage_t const snd_fpage, unsigned long snd_base, l4_mshtag_t * tag) throw () [inline]`

Add a flex-page to be sent to the UTCB.

Parameters

`snd_fpage` Flex-page.

`snd_base` Send base.

`tag` Tag to be modified.

Return values

`tag` Modified tag, the number of items will be increased, all other values in the tag will be retained.

Returns

0 on success, negative error code otherwise

Definition at line 486 of file [ipc.h](#).

References [l4_utcb\(\)](#).

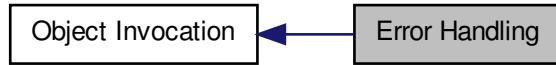
Here is the call graph for this function:



9.54 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_ENOTEXISTENT` = 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_mshtag_t tag, l4_utcb_t *utcb) throw ()`

Get the error code for an object invocation.

- long `l4_error (l4_msntag_t tag) throw ()`
Return error code of a system call return message tag.
- int `l4_ipc_is_snd_error (l4_utcb_t *utcb) throw ()`
Returns whether an error occurred in send phase of an invocation.
- int `l4_ipc_is_rcv_error (l4_utcb_t *utcb) throw ()`
Returns whether an error occurred in receive phase of an invocation.
- int `l4_ipc_error_code (l4_utcb_t *utcb) throw ()`
Get the error condition of the last invocation from the TCR.

9.54.1 Detailed Description

Error handling for L4 object invocation. #include <l4/sys/ ipc.h>

9.54.2 Enumeration Type Documentation

9.54.2.1 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 SECANCELED`** 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`** Cut receive message, due to message buffer is too small.
- `L4_IPC SEMSGCUT`** Cut send message, due to message buffer is too small,

Definition at line 75 of file `ipc.h`.

9.54.3 Function Documentation

9.54.3.1 `l4_umword_t l4_ipc_error (l4_mshtag_t tag, l4_utcb_t * utcb) throw () [inline]`

Get the error code for an object invocation.

Parameters

tag Return value of the invocation.

utcb UTCB that was used for the invocation.

Returns

0 if no error condition is set, error code otherwise (see [l4_ipc_tcr_error_t](#)).

Examples:

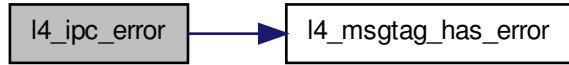
[examples/sys/ ipc/ ipc_example.c](#), [examples/sys/ isr/ main.c](#), and [examples/sys/ start-with-exc/ main.c](#).

Definition at line 430 of file [ipc.h](#).

References [l4_thread_regs_t::error](#), and [l4_mshtag_has_error\(\)](#).

Referenced by [L4Re::Util::Event_buffer_consumer_t< PAYLOAD >::process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.54.3.2 `long l4_error (l4_mshtag_t tag) throw () [inline]`

Return error code of a system call return message tag.

Parameters

tag System call return message type

Returns

0 for no error, error number in case of error

Examples:

[examples/clntsrv/client.cc](#), [examples/libs/l4re/streammap/client.cc](#), [examples/sys/ist/main.c](#), and [examples/sys/migrate/thread_migrate.cc](#).

Definition at line 447 of file [ipc.h](#).

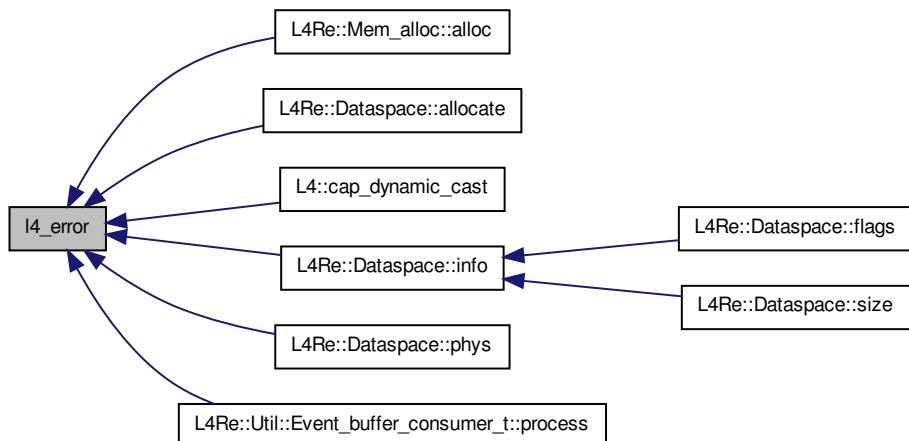
References [l4_utcb\(\)](#).

Referenced by [L4Re::Mem_alloc::alloc\(\)](#), [L4Re::Dataspace::allocate\(\)](#), [L4::cap_dynamic_cast\(\)](#), [L4Re::Dataspace::info\(\)](#), [L4Re::Dataspace::phys\(\)](#), and [L4Re::Util::Event_buffer_consumer_t< PAYLOAD >::process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.54.3.3 int l4_ipc_is_snd_error (l4_utcb_t * *utcb*) throw () [inline]

Returns whether an error occurred in send phase of an invocation.

Precondition

`l4_mshtag_has_error(tag) == true`

Parameters

utcb UTCB to check.

Returns

Boolean value.

Definition at line 453 of file [ipc.h](#).

References [l4_thread_regs_t::error](#).

9.54.3.4 int l4_ipc_is_recv_error (l4_utcb_t * *utcb*) throw () [inline]

Returns whether an error occurred in receive phase of an invocation.

Precondition

`l4_mshtag_has_error(tag) == true`

Parameters

utcb UTCB to check.

Returns

Boolean value.

Definition at line 456 of file [ipc.h](#).

References [l4_thread_regs_t::error](#).

9.54.3.5 int l4_ipc_error_code (l4_utcb_t * *utcb*) throw () [inline]

Get the error condition of the last invocation from the TCR.

Precondition

`l4_mshtag_has_error(tag) == true`

Parameters

utcb UTCB to check.

Returns

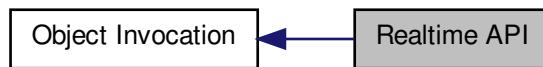
Error condition of type `l4_ipc_tcr_error_t`.

Definition at line 459 of file [ipc.h](#).

References [l4_thread_regs_t::error](#).

9.55 Realtime API

Collaboration diagram for Realtime API:



9.56 IRQs

The IRQ and IRQ class.

Collaboration diagram for IRQs:



Data Structures

- class [L4::Irq](#)
C++ version of an L4 IRQ.

Enumerations

- enum [L4_irq_flow_type](#) {

`L4_IRQ_F_NONE = 0, 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 = 0x3, L4_IRQ_F_LEVEL_LOW = 0x7,`

`L4_IRQ_F_POS_EDGE = 0x1, L4_IRQ_F_NEG_EDGE = 0x5, L4_IRQ_F_BOTH_EDGE = 0x9, L4_IRQ_F_MASK = 0xf }`

Interrupt flow types.

Functions

- `l4_msgtag_t l4_irq_attach (l4_cap_idx_t irq, l4_umword_t label, l4_cap_idx_t thread) throw ()`
Attach to an interrupt source.
- `l4_msgtag_t l4_irq_chain (l4_cap_idx_t irq, l4_umword_t label, l4_cap_idx_t slave) throw ()`
Chain an IRQ to another master IRQ source.
- `l4_msgtag_t l4_irq_detach (l4_cap_idx_t irq) throw ()`
Detach from an interrupt source.
- `l4_msgtag_t l4_irq_trigger (l4_cap_idx_t irq) throw ()`
Trigger an IRQ.
- `l4_msgtag_t l4_irq_receive (l4_cap_idx_t irq, l4_timeout_t to) throw ()`
Unmask and wait for specified IRQ.
- `l4_msgtag_t l4_irq_wait (l4_cap_idx_t irq, l4_umword_t *label, l4_timeout_t to) throw ()`
Unmask IRQ and wait for any message.
- `l4_msgtag_t l4_irq_unmask (l4_cap_idx_t irq) throw ()`
Unmask IRQ.

9.56.1 Detailed Description

The IRQ and IRQ class. `#include <l4/sys/irq.h>`

The IRQ class provides access to abstract interrupts provided by the micro kernel. Interrupts may be hardware interrupts provided by the platform interrupt controller, virtual device interrupts provided by the micro kernel virtual devices (virtual serial or trace buffer), or IRQs (virtual interrupts that can be triggered by user programs).

IRQ objects can be created using a Factory, see [Factory \(l4_factory_create_irq\(\)\)](#).

9.56.2 Enumeration Type Documentation

9.56.2.1 enum L4_irq_flow_type

Interrupt flow types.

Enumerator:

- `L4_IRQ_F_NONE`** None.
- `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.

Definition at line 61 of file [icu.h](#).

9.56.3 Function Documentation

9.56.3.1 `l4_mshtag_t l4_irq_attach (l4_cap_idx_t irq, l4_umword_t label, l4_cap_idx_t thread) throw () [inline]`

Attach to an interrupt source.

Parameters

irq IRQ to attach to.
label Identifier of the IRQ.
thread Thread to attach the interrupt to.

Returns

Syscall return tag

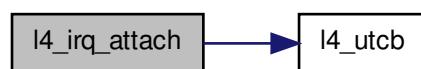
Examples:

[examples/sys/isr/main.c](#).

Definition at line 281 of file [irq.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.56.3.2 `l4_mshtag_t l4_irq_chain (l4_cap_idx_t irq, l4_umword_t label, l4_cap_idx_t slave) throw () [inline]`

Chain an IRQ to another master IRQ source.

The chaining feature of IRQ objects allows to deal with shared IRQs. For chaining IRQs there must be a master IRQ object, bound to the real IRQ source. Note, the master IRQ must not have a thread attached to it. This function allows to add a limited number of slave IRQs to this master IRQ, with the semantics that each of the slave IRQs is triggered whenever the master IRQ is triggered. The master IRQ will be masked automatically when an IRQ is delivered and shall be unmasked when all attached slave IRQs are unmasked.

Parameters

irq The master IRQ object.

label Identifier of the IRQ.

slave The slave that shall be attached to the master.

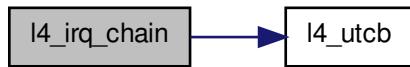
Returns

Syscall return tag

Definition at line 288 of file [irq.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.56.3.3 l4_msgtag_t l4_irq_detach (l4_cap_idx_t irq) throw () [inline]

Detach from an interrupt source.

Parameters

irq IRQ to detach from.

Returns

Syscall return tag

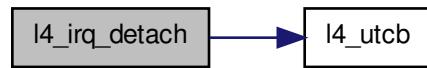
Examples:

[examples/sys/isr/main.c](#).

Definition at line 295 of file [irq.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.56.3.4 l4_mshtag_t l4_irq_trigger (l4_cap_idx_t *irq*) throw () [inline]

Trigger an IRQ.

Parameters

irq IRQ to trigger.

Precondition

irq must be a reference to an IRQ.

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 301 of file [irq.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.56.3.5 l4_mshtag_t l4_irq_receive (l4_cap_idx_t *irq*, l4_timeout_t *to*) throw () [inline]

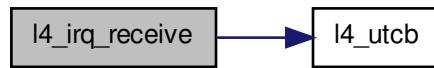
Unmask and wait for specified IRQ.

Parameters*irq* IRQ to wait for.*to* Timeout.**Returns**

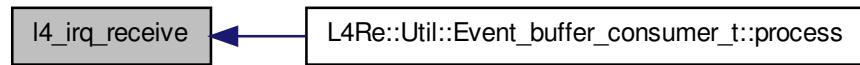
Syscall return tag

Examples:[examples/sys/isr/main.c](#).Definition at line 307 of file [irq.h](#).References [l4_utcb\(\)](#).Referenced by [L4Re::Util::Event_buffer_consumer_t< PAYLOAD >::process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.56.3.6 `l4_msgtag_t l4_irq_wait (l4_cap_idx_t irq, l4_umword_t * label, l4_timeout_t to)` throw () [inline]

Unmask IRQ and wait for any message.

Parameters*irq* IRQ to wait for.*label* Receive label.*to* Timeout.

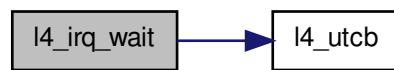
Returns

Syscall return tag

Definition at line 313 of file [irq.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:

**9.56.3.7 l4_mshtag_t l4_irq_unmask (l4_cap_idx_t irq) throw () [inline]**

Unmask IRQ.

Parameters

irq IRQ to unmask.

Returns

Syscall return tag

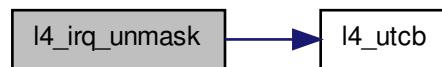
Note

`l4_irq_wait` and `l4_irq_receive` are doing the unmask themselves.

Definition at line 320 of file [irq.h](#).

References [l4_utcb\(\)](#).

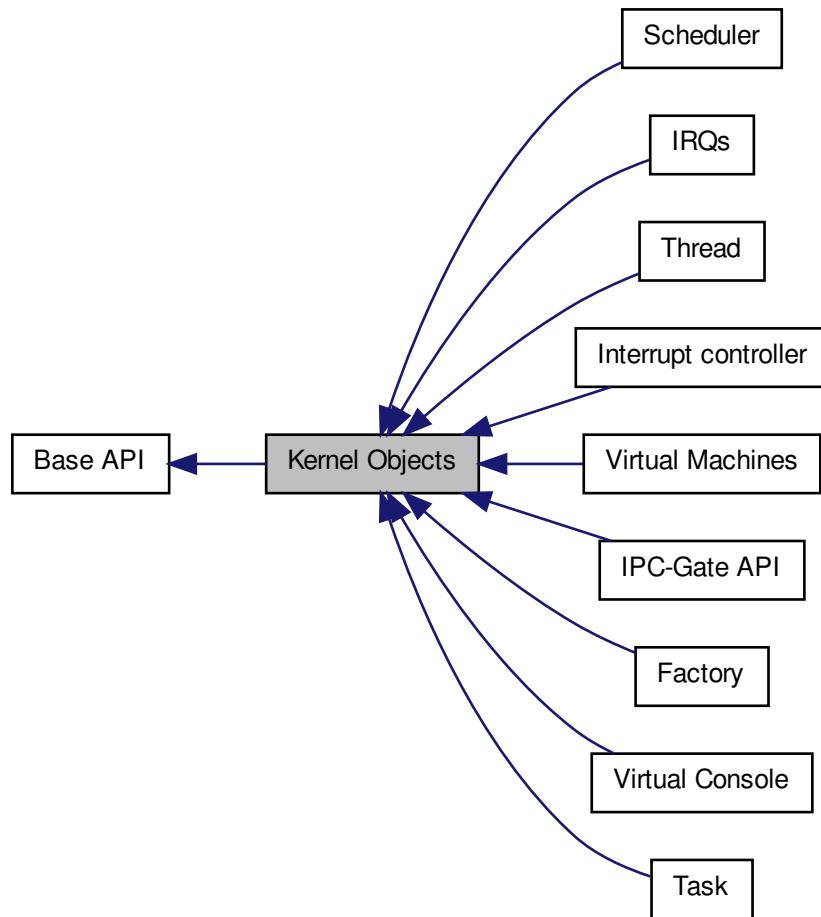
Here is the call graph for this function:



9.57 Kernel Objects

API of kernel objects.

Collaboration diagram for Kernel Objects:



Data Structures

- class [L4::Kobject](#)
Base class for all kinds of kernel objects, referred to by capabilities.
- class [L4::Meta](#)
Meta interface that shall be implemented by each [L4Re](#) object and gives access to the dynamic type information for [L4Re](#) objects.

Modules

- [IPC-Gate API](#)

Secure communication object.

- **Factory**

A factory is used to create all kinds of kernel objects.

- **Virtual Machines**

Virtual Machine API.

- **Interrupt controller**

The ICU class.

- **IRQs**

The IRQ and IRQ class.

- **Scheduler**

Scheduler object.

- **Task**

Class definition of the Task kernel object.

- **Thread**

Thread object.

- **Virtual Console**

Virtual console for simple character based input and output.

Defines

- **#define L4_DISABLE_COPY(_class)**

Disable copy of a class.

- **#define L4_KOBJECT_DISABLE_COPY(_class)**

Disable copy and instantiation of a class.

- **#define L4_KOBJECT(_class) L4_KOBJECT_DISABLE_COPY(_class)**

Declare a kernel object class.

9.57.1 Detailed Description

API of kernel objects. #include <14/sys/capability>
#include <14/sys/kernel_object.h>

9.57.2 Define Documentation

9.57.2.1 #define L4_DISABLE_COPY(*_class*)

Value:

```
private:
    _class(_class const &);           \
    _class operator = (_class const &); \
```

Disable copy of a class.

Parameters

_class is the 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 396 of file [capability](#).

9.57.2.2 #define L4_KOBJECT_DISABLE_COPY(*_class*)

Value:

```
protected:
    _class();
    L4_DISABLE_COPY(_class)
```

Disable copy and instantiation of a class.

Parameters

_class is the name of the class to be not copyable and not instantiatable.

The typical use looks like:

```
class Type
{
    L4_KOBJECT_DISABLE_COPY(Type)
};
```

Definition at line 415 of file [capability](#).

9.57.2.3 #define L4_KOBJECT(*_class*) L4_KOBJECT_DISABLE_COPY(*_class*)

Declare a kernel object class.

Parameters

_class is the class name.

The use of this macro disables copy and instantiation of the class as needed for kernel object classes derived from [L4::Kobject](#).

The typical use looks like:

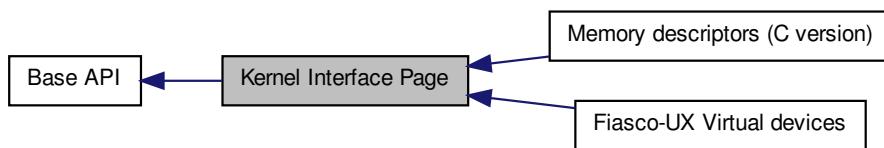
```
class Type : public L4::Kobject_t<Type, L4::Kobject>
{
    L4_KOBJECT(Type)
};
```

Definition at line 437 of file [capability](#).

9.58 Kernel Interface Page

Kernel Interface Page.

Collaboration diagram for Kernel Interface Page:



Data Structures

- struct [l4_kernel_info_t](#)
L4 Kernel Interface Page.
- class [L4::Kip::Mem_desc](#)
Memory descriptors stored in the kernel interface page.

Modules

- [Memory descriptors \(C version\)](#)
C Interface for KIP memory descriptors.
- [Fiasco-UX Virtual devices](#)
Virtual hardware devices, provided by Fiasco-UX.

Defines

- #define [L4_KERNEL_INFO_MAGIC](#) (0x4BE6344CL)
Kernel Info Page identifier ("L4&K").

Typedefs

- `typedef struct l4_kernel_info_t l4_kernel_info_t`
`L4 Kernel Interface Page.`
- `typedef struct l4_kernel_info_t l4_kernel_info_t`
`L4 Kernel Interface Page.`

Functions

- `l4_umword_t l4_kip_version (l4_kernel_info_t *kip) throw ()`
`Get the kernel version.`
- `const char * l4_kip_version_string (l4_kernel_info_t *kip) throw ()`
`Get the kernel version string.`
- `int l4_kernel_info_version_offset (l4_kernel_info_t *kip) throw ()`
`Return offset in bytes of version_strings relative to the KIP base.`

9.58.1 Detailed Description

Kernel Interface Page. C interface for the Kernel Interface Page:

```
#include <l4/sys/kip.h>
```

C++ interface for the Kernel Interface Page:

```
#include <l4/sys/kip>
```

9.58.2 Function Documentation

9.58.2.1 `l4_umword_t l4_kip_version (l4_kernel_info_t * kip) throw () [inline]`

Get the kernel version.

Parameters

`kip` Kernel Interface Page.

Returns

Kernel version string. 0 if KIP could not be mapped.

Definition at line 103 of file `kip.h`.

9.58.2.2 `const char * l4_kip_version_string (l4_kernel_info_t * kip) throw () [inline]`

Get the kernel version string.

Parameters

kip Kernel Interface Page.

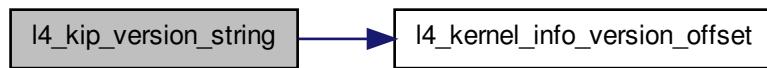
Returns

Kernel version string.

Definition at line 107 of file [kip.h](#).

References [l4_kernel_info_version_offset\(\)](#).

Here is the call graph for this function:



9.58.2.3 int l4_kernel_info_version_offset (l4_kernel_info_t * *kip*) throw () [inline]

Return offset in bytes of version_strings relative to the KIP base.

Parameters

kip Pointer to the kernel into page (KIP).

Returns

offset of version_strings relative to the KIP base address, in bytes.

Definition at line 111 of file [kip.h](#).

Referenced by [l4_kip_version_string\(\)](#).

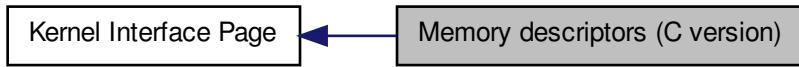
Here is the caller graph for this function:



9.59 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_bootloader = 0xe, l4_mem_type_archspecific = 0xf }`
Type of a memory descriptor.

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.

9.59.1 Detailed Description

C Interface for KIP memory descriptors. `#include <l4/sys/memdesc.h>`

This module contains the C functions to access the memory descriptor in the kernel interface page (KIP).

9.59.2 Typedef Documentation

9.59.2.1 `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.

9.59.3 Enumeration Type Documentation

9.59.3.1 `enum l4_mem_type_t`

Type of a memory descriptor.

Enumerator:

- `l4_mem_type_undefined` Undefined, unused descriptor.
- `l4_mem_type_conventional` Conventional memory.
- `l4_mem_type_reserved` Reserved memory for kernel etc.
- `l4_mem_type_dedicated` Dedicated memory (some device memory).

l4_mem_type_shared Shared memory (not implemented).

l4_mem_type_bootloader Memory owned by the boot loader.

l4_mem_type_archspecific Architecture specific memory (e.g., ACPI memory).

Definition at line 44 of file [memdesc.h](#).

9.59.4 Function Documentation

9.59.4.1 `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 178 of file [memdesc.h](#).

9.59.4.2 `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

md Pointer to memory descriptor

start Start of region

end End of region

type Type of region

virt 1 if virtual region, 0 if physical region

sub_type Sub type.

Definition at line 185 of file [memdesc.h](#).

9.59.4.3 `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 200 of file [memdesc.h](#).

**9.59.4.4 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 207 of file [memdesc.h](#).

**9.59.4.5 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 214 of file [memdesc.h](#).

**9.59.4.6 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 221 of file [memdesc.h](#).

**9.59.4.7 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

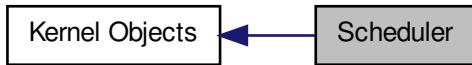
1 if region is virtual memory, 0 if region is physical memory

Definition at line 228 of file [memdesc.h](#).

9.60 Scheduler

Scheduler object.

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_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) throw ()`
Get scheduler information.
- `l4_mshtag_t l4_scheduler_info (l4_cap_idx_t scheduler, l4_umword_t *cpu_max, l4_sched_cpu_set_t *cpus) throw ()`
Construct scheduler parameter.
- `l4_sched_param_t l4_sched_param (unsigned prio, l4_cpu_time_t quantum=0) throw ()`
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) throw ()`
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) throw ()`
Query idle time of a CPU, in μ s.
- `int l4_scheduler_is_online (l4_cap_idx_t scheduler, l4_umword_t cpu) throw ()`
Query if a CPU is online.

9.60.1 Detailed Description

Scheduler object. `#include <l4/sys/scheduler.h>`

9.60.2 Enumeration Type Documentation

9.60.2.1 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 185 of file `scheduler.h`.

9.60.3 Function Documentation

9.60.3.1 l4_sched_cpu_set_t l4_sched_cpu_set (l4_umword_t offset, unsigned char granularity, l4_umword_t map = 1) throw () [inline]

Parameters

`offset` Offset.

`granularity` Granularity in log2 notation.

`map` Bitmap of CPUs, defaults to 1 in C++.

Returns

CPU set.

Examples:

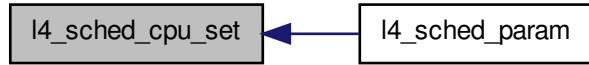
`examples/sys/migrate/thread_migrate.cc`.

Definition at line 195 of file [scheduler.h](#).

References [l4_sched_cpu_set_t::granularity](#), [l4_sched_cpu_set_t::map](#), and [l4_sched_cpu_set_t::offset](#).

Referenced by [l4_sched_param\(\)](#).

Here is the caller graph for this function:



9.60.3.2 l4_mshtag_t l4_scheduler_info (l4_cap_idx_t *scheduler*, l4_umword_t * *cpu_max*, l4_sched_cpu_set_t * *cpus*) throw () [inline]

Get scheduler information.

Parameters

scheduler Scheduler object.

Return values

cpu_max maximum number of CPUs ever available.

Parameters

cpus *cpus.offset* is first CPU of interest. *cpus.granularity* (see [l4_sched_cpu_set_t](#)).

Return values

cpus *cpus.map* Bitmap of online CPUs.

Returns

0 on success, <0 error code otherwise.

Definition at line 284 of file [scheduler.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.60.3.3 `l4_mshtag_t l4_scheduler_run_thread (l4_cap_idx_t scheduler, l4_cap_idx_t thread, l4_sched_param_t const * sp) throw () [inline]`

Run a thread on a Scheduler.

Parameters

scheduler Scheduler object.

thread Thread to run.

sp Scheduling parameters.

Returns

0 on success, <0 error code otherwise.

Definition at line 291 of file [scheduler.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.60.3.4 `l4_mshtag_t l4_scheduler_idle_time (l4_cap_idx_t scheduler, l4_sched_cpu_set_t const * cpus) throw () [inline]`

Query idle time of a CPU, in μ s.

Parameters

scheduler Scheduler object.

cpus Set of CPUs to query.

The consumed time is returned as `l4_kernel_clock_t` at UTCB message register 0.

Definition at line 298 of file [scheduler.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.60.3.5 `int l4_scheduler_is_online (l4_cap_idx_t scheduler, l4_umword_t cpu) throw () [inline]`

Query if a CPU is online.

Parameters

scheduler Scheduler object.

cpu CPU number.

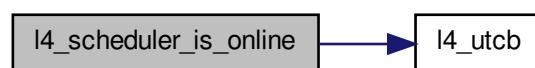
Returns

true if online, false if not (or any other query error).

Definition at line 304 of file [scheduler.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.61 Task

Class definition of the Task kernel object.

Collaboration diagram for Task:



Data Structures

- class [L4::Task](#)

An L4 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_map \(l4_cap_idx_t dst_task, l4_cap_idx_t src_task, l4_fpage_t const snd_fpage, l4_addr_t snd_base\) L4_NOTHROW](#)
Map resources available in the source task to a destination task.
- [l4_msgtag_t l4_task_unmap \(l4_cap_idx_t task, l4_fpage_t const fpage, l4_umword_t map_mask\) L4_NOTHROW](#)
Revoke rights from the task.
- [l4_msgtag_t l4_task_unmap_batch \(l4_cap_idx_t task, l4_fpage_t const *fpages, unsigned num_fpates, unsigned long map_mask\) 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 capability.
- [l4_msgtag_t l4_task_cap_valid \(l4_cap_idx_t task, l4_cap_idx_t cap\) L4_NOTHROW](#)
Test whether a capability selector points to a valid capability.
- [l4_msgtag_t l4_task_cap_has_child \(l4_cap_idx_t task, l4_cap_idx_t cap\) L4_NOTHROW](#)

Test whether a capability has child mappings (in another task).

- [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 rights.

- [l4_msgtag_t l4_task_add_ku_mem \(l4_cap_idx_t task, l4_fpage_t const ku_mem\) L4_NOTHROW](#)

Add kernel-user memory.

9.61.1 Detailed Description

Class definition of the Task kernel object. `#include <14/sys/task.h>`

The [L4](#) task class represents a combination of the address spaces provided by the [L4](#) 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.

A task object can be created using a Factory, see [Factory \(l4_factory_create_task\(\)\)](#).

9.61.2 Enumeration Type Documentation

9.61.2.1 enum l4_unmap_flags_t

Flags for the unmap operation.

See also

[L4::Task::unmap\(\)](#) and [l4_task_unmap\(\)](#)

Enumerator:

[L4_FP_ALL_SPACES](#) Flag to tell the unmap operation to unmap all child mappings including the mapping in the invoked task.

See also

[L4::Task::unmap\(\)](#) [l4_task_unmap\(\)](#)

[L4_FP_DELETE_OBJ](#) Flag that indicates that the unmap operation on a capability shall try to delete the corresponding objects immediately.

See also

[L4::Task::unmap\(\)](#) [l4_task_unmap\(\)](#)

[L4_FP_OTHER_SPACES](#) Counterpart to [L4_FP_ALL_SPACES](#), unmap only child mappings.

See also

[L4::Task::unmap\(\)](#) [l4_task_unmap\(\)](#)

Definition at line 163 of file `consts.h`.

9.61.3 Function Documentation

9.61.3.1 `l4_mshtag_t l4_task_map (l4_cap_idx_t dst_task, l4_cap_idx_t src_task, l4_fpage_t const snd_fpage, l4_addr_t snd_base) [inline]`

Map resources available in the source task to a destination task.

Parameters

dst_task Capability selector of destination task

src_task Capability selector of source task

snd_fpage Send flexpage that describes an area in the address space or object space of the source task

snd_base Send base that describes an offset in the receive window of the destination task.

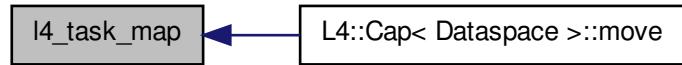
Returns

Syscall return tag

This method allows for asynchronous rights delegation from one task to another. It can be used to share memory as well as to delegate access to objects.

Referenced by [L4::Cap< Dataspace >::move\(\)](#).

Here is the caller graph for this function:



9.61.3.2 `l4_mshtag_t l4_task_unmap (l4_cap_idx_t task, l4_fpage_t const fpage, l4_umword_t map_mask) [inline]`

Revoke rights from the task.

Parameters

task Capability selector of destination task

fpage Flexpage that describes an area in the address space or object space of the destination task

map_mask Unmap mask, see [l4_unmap_flags_t](#)

Returns

Syscall return tag

This method allows to revoke rights from the destination task and from all the tasks that got the rights delegated from that task (i.e., this operation does a recursive rights revocation).

Note

Calling this function on the object space can cause a root capability of an object to be destructed, which destroys the object itself.

9.61.3.3 `l4_msntag_t l4_task_unmap_batch (l4_cap_idx_t task, l4_fpage_t const * fpages, unsigned num_fpages, unsigned long map_mask) [inline]`

Revoke rights from a task.

Parameters

`task` Capability selector of destination task

`fpages` An array of flexpages that describes an area in the address space or object space of the destination task each

`num_fpages` The size of the fpages array in elements (number of fpages sent).

`map_mask` Unmap mask, see [l4_unmap_flags_t](#)

Returns

Syscall return tag

This method allows to revoke rights from the destination task and from all the tasks that got the rights delegated from that task (i.e., this operation does a recursive rights revocation).

Precondition

The caller needs to take care that num_fpages is not bigger than L4_UTCB_GENERIC_DATA_SIZE - 2.

Note

Calling this function on the object space can cause a root capability of an object to be destructed, which destroys the object itself.

Definition at line 373 of file [task.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.61.3.4 l4_mshtag_t l4_task_delete_obj (l4_cap_idx_t task, l4_cap_idx_t obj) [inline]

Release capability and delete object.

Parameters

task Capability selector of destination task

obj Capability selector of object to delete

Returns

Syscall return tag

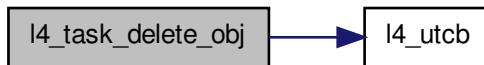
The object will be deleted if the obj has sufficient rights. No error will be reported if the rights are insufficient, however, the capability is removed in all cases.

This is operating calls [l4_task_unmap\(\)](#) with [L4_FP_DELETE_OBJ](#).

Definition at line 389 of file [task.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:

**9.61.3.5 l4_mshtag_t l4_task_release_cap (l4_cap_idx_t task, l4_cap_idx_t cap) [inline]**

Release capability.

Parameters

task Capability selector of destination task

cap Capability selector to release

Returns

Syscall return tag

This operation unmaps the capability from the specified task.

Definition at line 404 of file [task.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.61.3.6 `l4_mshtag_t l4_task_cap_valid(l4_cap_idx_t task, l4_cap_idx_t cap) [inline]`

Test whether a capability selector points to a valid capability.

Parameters

task Capability selector of the destination task to do the lookup in
cap Capability selector to look up in the destination task

Returns

label contains 1 if valid, 0 if invalid

Definition at line 410 of file [task.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.61.3.7 `l4_mshtag_t l4_task_cap_has_child(l4_cap_idx_t task, l4_cap_idx_t cap) [inline]`

Test whether a capability has child mappings (in another task).

Parameters

task Capability selector of the destination task to do the lookup in
cap Capability selector to look up in the destination task

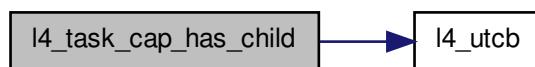
Returns

label contains 1 if it has at least one child, 0 if not or invalid

Definition at line 416 of file [task.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.61.3.8 `l4_mshtag_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 rights.

Parameters

task Capability selector of the destination task to do the lookup in

cap_a Capability selector to compare

cap_b Capability selector to compare

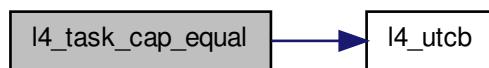
Returns

label contains 1 if equal, 0 if not equal

Definition at line 422 of file [task.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.61.3.9 `l4_mshtag_t l4_task_add_ku_mem (l4_cap_idx_t task, l4_fpage_t const ku_mem) [inline]`

Add kernel-user memory.

Parameters

task Capability selector of the task to add the memory to

ku_mem Flexpage describing the virtual area the memory goes to.

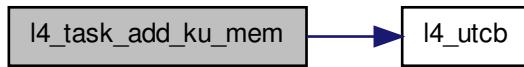
Returns

Syscall return tag

Definition at line 429 of file [task.h](#).

References [l4_utcb\(\)](#).

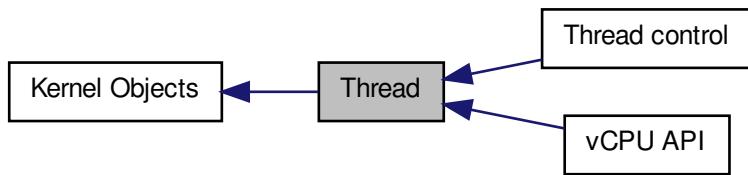
Here is the call graph for this function:



9.62 Thread

Thread object.

Collaboration diagram for Thread:



Data Structures

- class [L4::Thread](#)

L4 kernel thread.

Modules

- Thread control

API for Thread Control method.

- vCPU API

vCPU API

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_GDT_X86_OP` = 0x10UL, `L4_THREAD_SET_FS_AMD64_OP` = 0x12UL, `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_UX_NATIVE` = 0x0800000,

`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_EXCEPTION` = 0x20000UL }

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_mshtag_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_mshtag_t l4_thread_yield](#) (void) L4_NOTHROW

Yield current time slice.

- [l4_mshtag_t l4_thread_switch](#) (*l4_cap_idx_t* *to_thread*) L4_NOTHROW

Switch to another thread (and donate the remaining time slice).

- [l4_mshtag_t l4_thread_stats_time](#) (*l4_cap_idx_t* *thread*) L4_NOTHROW

Get consumed timed of thread in μ s.

- [l4_mshtag_t l4_thread_vcpu_resume_start](#) (void) L4_NOTHROW

vCPU return from event handler.

- [l4_mshtag_t l4_thread_vcpu_resume_commit](#) (*l4_cap_idx_t* *thread*, *l4_mshtag_t* *tag*) L4_NOTHROW

Commit vCPU resume.

- [l4_mshtag_t l4_thread_vcpu_control](#) (*l4_cap_idx_t* *thread*, *l4_addr_t* *vcpu_state*) L4_NOTHROW

Enable or disable the vCPU feature for the thread.

- [l4_mshtag_t l4_thread_vcpu_control_ext](#) (*l4_cap_idx_t* *thread*, *l4_addr_t* *ext_vcpu_state*) L4_NOTHROW

Enable or disable the extended vCPU feature for the thread.

- [l4_mshtag_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_mshtag_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_mshtag_t* **tag*) L4_NOTHROW

Add a modification pattern to a sender modification sequence.

- [l4_mshtag_t l4_thread_modify_sender_commit](#) (*l4_cap_idx_t* *thread*, *l4_mshtag_t* *tag*) L4_NOTHROW

Apply (commit) a sender modification sequence.

9.62.1 Detailed Description

Thread object. #include <14/sys/thread.h>

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 [Factory \(l4_factory_create_thread\(\)\)](#).

An [L4](#) thread encapsulates:

- CPU state
 - General-purpose registers
 - Program counter
 - Stack pointer
- FPU state
- Scheduling parameters
 - CPU-set
 - Priority (0-255)
 - Time slice length
- 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.

9.62.2 Enumeration Type Documentation

9.62.2.1 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_GDT_X86_OP*** Gdt.
- L4_THREAD_SET_FS_AMD64_OP*** Set FS/TLS.
- L4_THREAD_OPCODE_MASK*** Mask for opcodes.

Definition at line [587](#) of file [thread.h](#).

9.62.2.2 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_UX_NATIVE* Fiasco-UX only: pass-through of host system calls is set.
- L4_THREAD_CONTROL_SET_EXC_HANDLER* The exception handler of the thread will be given.

Definition at line 613 of file [thread.h](#).

9.62.2.3 enum L4_thread_control_mr_indices

Indices for the values in the message register for thread control.

Enumerator:

See also

- L4_THREAD_CONTROL_MR_IDX_FLAGS* [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 flex-page for bind.

Definition at line 636 of file [thread.h](#).

9.62.2.4 enum L4_thread_ex_regs_flags

Flags for the thread ex-reg operation.

Enumerator:

- L4_THREAD_EX_REGS_CANCEL* Cancel ongoing IPC in the thread.
- L4_THREAD_EX_REGS_TRIGGER_EXCEPTION* Trigger artificial exception in thread.

Definition at line 651 of file [thread.h](#).

9.62.3 Function Documentation

9.62.3.1 *l4_mshtag_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

thread Thread to manipulate

ip New instruction pointer, use ~0UL to leave the instruction pointer unchanged

sp New stack pointer, use ~0UL to leave the stack pointer unchanged

flags Ex-reg 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*).

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 791 of file [thread.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.62.3.2 `l4_mshtag_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

[in] *thread* Thread to manipulate

[in, out] *ip* New instruction pointer, use ~0UL to leave the instruction pointer unchanged, return previous instruction pointer

[in, out] *sp* New stack pointer, use ~0UL to leave the stack pointer unchanged, returns previous stack pointer

[in, out] *flags* Ex-reg 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*).

Returned values are valid only if function returns successfully.

Definition at line 798 of file [thread.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.62.3.3 l4_mshtag_t l4_thread_yield (void) [inline]

Yield current time slice.

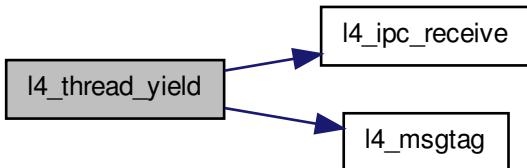
Returns

system call return tag

Definition at line 751 of file [thread.h](#).

References [L4_INVALID_CAP](#), [L4_IPC_BOTH_TIMEOUT_0](#), [l4_ipc_receive\(\)](#), and [l4_mshtag\(\)](#).

Here is the call graph for this function:



9.62.3.4 l4_mshtag_t l4_thread_switch (l4_cap_idx_t to_thread) [inline]

Switch to another thread (and donate the remaining time slice).

Parameters

to_thread The thread to switch to.

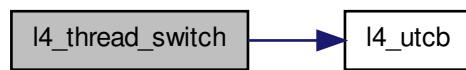
Returns

system call return tag

Definition at line 851 of file [thread.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:

**9.62.3.5 l4_mshtag_t l4_thread_stats_time (l4_cap_idx_t thread) [inline]**

Get consumed timed of thread in Ås.

Parameters

thread Thread to get the consumed time from.

The consumed time is returned as l4_kernel_clock_t at UTCB message register 0.

Definition at line 860 of file [thread.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:

**9.62.3.6 l4_mshtag_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_commit\(\)](#)

The vCPU resume functionality is split in multiple functions to allow the specification of additional send-flex-pages using [l4_sndfpage_add\(\)](#).

Definition at line 866 of file [thread.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.62.3.7 `l4_mshtag_t l4_thread_vcpu_resume_commit (l4_cap_idx_t thread, l4_mshtag_t tag) [inline]`

Commit vCPU resume.

Parameters

thread Thread to be resumed, using the invalid cap can be used for the current thread.

tag Tag to use, returned by [l4_thread_vcpu_resume_start\(\)](#)

Returns

System call result message tag. In extended vCPU mode and when the virtual interrupts are cleared, the return code 1 flags an incoming IPC message, whereas 0 indicates a VM exit.

To resume into another address space the capability to the target task must be set in the vCPU-state. The task needs to be set only once, consecutive resumes to the same address space should use an invalid capability. The kernel resets the field to [L4_INVALID_CAP](#). To release a task use a different task or use an invalid capability with the [L4_SYSF_REPLY](#) flag set.

See also

[l4_vcpu_state_t](#)

Definition at line 872 of file [thread.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.62.3.8 `l4_mshtag_t l4_thread_vcpu_control (l4_cap_idx_t thread, l4_addr_t vcpu_state) [inline]`

Enable or disable the vCPU feature for the thread.

Parameters

thread The thread for which the vCPU feature shall be enabled or disabled.

vcpu_state 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.

Returns

Systemcall result message tag.

This function enables the vCPU feature of the *thread* if *vcpu_state* is set to a valid kernel-user-memory address, or disables the vCPU feature if *vcpu_state* is 0.

Definition at line 909 of file [thread.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.62.3.9 `l4_mshtag_t l4_thread_vcpu_control_ext (l4_cap_idx_t thread, l4_addr_t ext_vcpu_state) [inline]`

Enable or disable the extended vCPU feature for the thread.

Parameters

thread The thread for which the extended vCPU feature shall be enabled or disabled.

vcpu_state 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.

Returns

Systemcall result message 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* if *vcpu_state* is set to a valid kernel-user-memory address, or disables the vCPU feature if *vcpu_state* is 0.

Definition at line 924 of file [thread.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.62.3.10 `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

thread Thread to register IRQ for.

irq Irq to register.

Returns

System call result message tag.

Definition at line 892 of file [thread.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.62.3.11 `l4_mshtag_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()`.

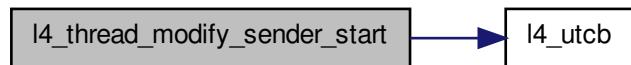
See also

`l4_thread_modify_sender_add`
`l4_thread_modify_sender_commit`

Definition at line 965 of file `thread.h`.

References `l4_utcb()`.

Here is the call graph for this function:



9.62.3.12 `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_mshtag_t * tag) [inline]`

Add a modification pattern to a sender modification sequence.

Parameters

tag Tag received from `l4_thread_modify_sender_start()` or previous `l4_thread_modify_sender_add()` calls from the same sequence.

match_mask Bitmask of bits to match the label.

match Bitmask that must be equal to the label after applying match_mask.

del_bits Bits to be deleted from the label.

add_bits Bits to be added to the label.

Returns

0 on sucess, <0 on error

In pseudo code: if ((sender_label & match_mask) == match) { label = (label & ~del_bits) | add_bits; }

Only the first match is applied.

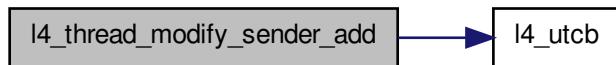
See also

[l4_thread_modify_sender_start](#)
[l4_thread_modify_sender_commit](#)

Definition at line 971 of file [thread.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.62.3.13 **l4_mshtag_t l4_thread_modify_sender_commit (l4_cap_idx_t thread, l4_mshtag_t tag) [inline]**

Apply (commit) a sender modification sequence.

See also

[l4_thread_modify_sender_start](#)
[l4_thread_modify_sender_add](#)

Definition at line 982 of file [thread.h](#).

References [l4_utcb\(\)](#).

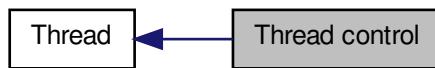
Here is the call graph for this function:



9.63 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.
- void [l4_thread_control_ux_host_syscall](#) (int on) L4_NOTHROW
Enable pass through of native host (Linux) system calls.
- [l4_msgtag_t](#) [l4_thread_control_commit](#) ([l4_cap_idx_t](#) thread) L4_NOTHROW

Commit the thread control parameters.

9.63.1 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_utcb_t *u = l4_utcb();
l4_thread_control_start(u);
l4_thread_control_pager(u, pager_cap);
l4_thread_control_bind (u, thread_utcb, task);
l4_thread_control_commit(u, thread_cap);
```

9.63.2 Function Documentation

9.63.2.1 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\(\)](#)
- [l4_thread_control_ux_host_syscall\(\)](#) (Fiasco-UX only)

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, 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 805 of file [thread.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.63.2.2 void l4_thread_control_pager (l4_cap_idx_t *pager*) [inline]

Set the pager.

Parameters

pager 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).

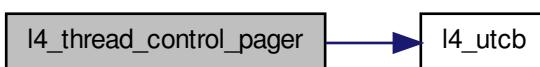
Examples:

[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 811 of file [thread.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.63.2.3 void l4_thread_control_exc_handler (l4_cap_idx_t *exc_handler*) [inline]

Set the exception handler.

Parameters

exc_handler 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 817 of file [thread.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.63.2.4 void l4_thread_control_bind (l4_utcb_t * *thread_utcb*, l4_cap_idx_t *task*) [inline]

Bind the thread to a task.

Parameters

thread_utcb The address of the UTCB in the target task.

task The target task of the thread.

Binding a thread to a task has the effect that the thread afterwards executes code within that task and has access to the resources visible within that task.

Note

There should not be more than one thread use a UTCB 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 824 of file [thread.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.63.2.5 void l4_thread_control_alien (int on) [inline]

Enable alien mode.

Parameters

on Boolean value defining the state of the feature.

Alien mode means the thread is not allowed to invoke L4 kernel objects directly and it is also not allowed to allocate FPU state. All those operations result in an exception IPC that gets sent through the pager capability. The responsible pager can then selectively allow an object invocation or allocate FPU state for the thread.

This feature can be used to attach a debugger to a thread and trace all object invocations.

Examples:

[examples/sys/aliens/main.c](#), and [examples/sys/singlestep/main.c](#).

Definition at line 830 of file [thread.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.63.2.6 void l4_thread_control_ux_host_syscall (int on) [inline]

Enable pass through of native host (Linux) system calls.

Parameters

on Boolean value defining the state of the feature.

Precondition

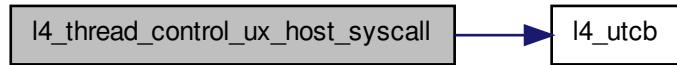
Running on Fiasco-UX

This enables the thread to do host system calls. This feature is only available in Fiasco-UX and ignored in other environments.

Definition at line 836 of file [thread.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.63.2.7 `l4_mshtag_t l4_thread_control_commit(l4_cap_idx_t thread) [inline]`

Commit the thread control parameters.

Parameters

thread Capability selector of target thread to commit to.

Returns

system call return tag

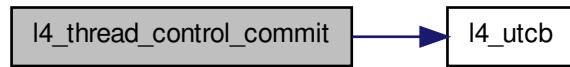
Examples:

[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 842 of file [thread.h](#).

References [l4_utcb\(\)](#).

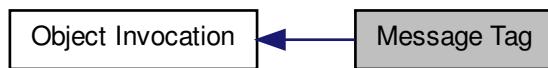
Here is the call graph for this function:



9.64 Message Tag

API related to the message tag data type.

Collaboration diagram for Message Tag:



Data Structures

- struct [l4_mshtag_t](#)
Message tag data structure.

Typedefs

- typedef struct [l4_mshtag_t](#) [l4_mshtag_t](#)
Message tag data structure.

Enumerations

- enum [l4_mshtag_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_PREEMPTION = -3L, L4_PROTO_SYS_EXCEPTION = -4L, L4_PROTO_EXCEPTION = -5L,`

```

L4_PROTO_SIGMA0 = -6L, L4_PROTO_IO_PAGE_FAULT = -8L, 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_META = -21L }

```

Message tag for IPC operations.

- enum l4_mshtag_flags {

```

L4_MSGTAG_ERROR, L4_MSGTAG_XCPU, L4_MSGTAG_TRANSFER_FPU, L4_MSGTAG_SCHEDULE,
L4_MSGTAG_PROPAGATE, L4_MSGTAG_FLAGS }

```

Flags for message tags.

Functions

- l4_mshtag_t l4_mshtag (long label, unsigned words, unsigned items, unsigned flags) throw ()

Create a message tag from the specified values.
- long l4_mshtag_label (l4_mshtag_t t) throw ()

Get the protocol of tag.
- unsigned l4_mshtag_words (l4_mshtag_t t) throw ()

Get the number of untyped words.
- unsigned l4_mshtag_items (l4_mshtag_t t) throw ()

Get the number of typed items.
- unsigned l4_mshtag_flags (l4_mshtag_t t) throw ()

Get the flags.
- unsigned l4_mshtag_has_error (l4_mshtag_t t) throw ()

Test for error indicator flag.
- unsigned l4_mshtag_is_page_fault (l4_mshtag_t t) throw ()

Test for page-fault protocol.
- unsigned l4_mshtag_is_preemption (l4_mshtag_t t) throw ()

Test for preemption protocol.
- unsigned l4_mshtag_is_sys_exception (l4_mshtag_t t) throw ()

Test for system-exception protocol.
- unsigned l4_mshtag_is_exception (l4_mshtag_t t) throw ()

Test for exception protocol.
- unsigned l4_mshtag_is_sigma0 (l4_mshtag_t t) throw ()

Test for sigma0 protocol.

- `unsigned l4_mshtag_is_io_page_fault (l4_mshtag_t t) throw ()`
Test for IO-page-fault protocol.

9.64.1 Detailed Description

API related to the message tag data type. `#include <l4/sys/types.h>`

9.64.2 Typedef Documentation

9.64.2.1 `typedef struct l4_mshtag_t l4_mshtag_t`

Message tag data structure.

```
#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.

9.64.3 Enumeration Type Documentation

9.64.3.1 `enum l4_mshtag_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_PREEMPTION`** Preemption message.
- `L4_PROTO_SYS_EXCEPTION`** System exception.
- `L4_PROTO_EXCEPTION`** Exception.
- `L4_PROTO_SIGMA0`** Sigma0 protocol.
- `L4_PROTO_IO_PAGE_FAULT`** I/O page fault message.
- `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_META Meta information protocol.

Definition at line 49 of file [types.h](#).

9.64.3.2 enum l4_mshtag_flags

Flags for message tags.

Enumerator:

L4_MSGTAG_ERROR Error indicator flag.

L4_MSGTAG_XCPU Cross-CPU invocation 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_PROPAGATE Enable IPC propagation. This flag enables IPC propagation, which means an IPC reply-connection from the current caller will be propagated to the new IPC receiver. This makes it possible to propagate an IPC call to a third thread, which may then directly answer to the caller.

L4_MSGTAG_FLAGS Mask for all flags.

Definition at line 89 of file [types.h](#).

9.64.4 Function Documentation

9.64.4.1 l4_mshtag_t l4_mshtag (long *label*, unsigned *words*, unsigned *items*, unsigned *flags*) throw () [inline]

Create a message tag from the specified values.

Message tag functions.

Parameters

label the user-defined label

words the number of untyped words within the UTCB

items the number of typed items (e.g., flex pages) within the UTCB

flags the IPC flags for realtime and FPU extensions

Returns

Message tag

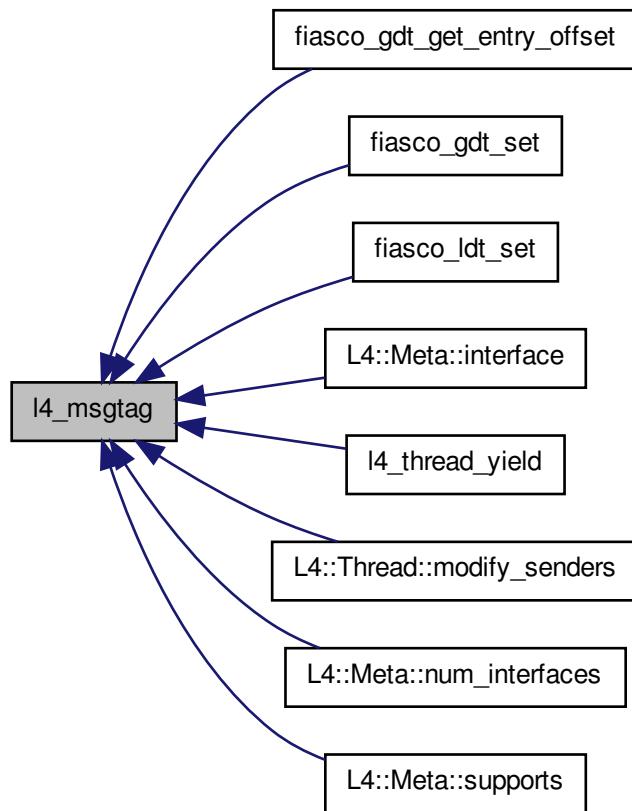
Examples:

[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 366 of file [types.h](#).

Referenced by [fiasco_gdt_get_entry_offset\(\)](#), [fiasco_gdt_set\(\)](#), [fiasco_ldt_set\(\)](#), [L4::Meta::interface\(\)](#), [l4_thread_yield\(\)](#), [L4::Thread::modify_senders\(\)](#), [L4::Meta::num_interfaces\(\)](#), and [L4::Meta::supports\(\)](#).

Here is the caller graph for this function:



9.64.4.2 long l4_mshtag_label (l4_mshtag_t t) throw () [inline]

Get the protocol of tag.

Parameters

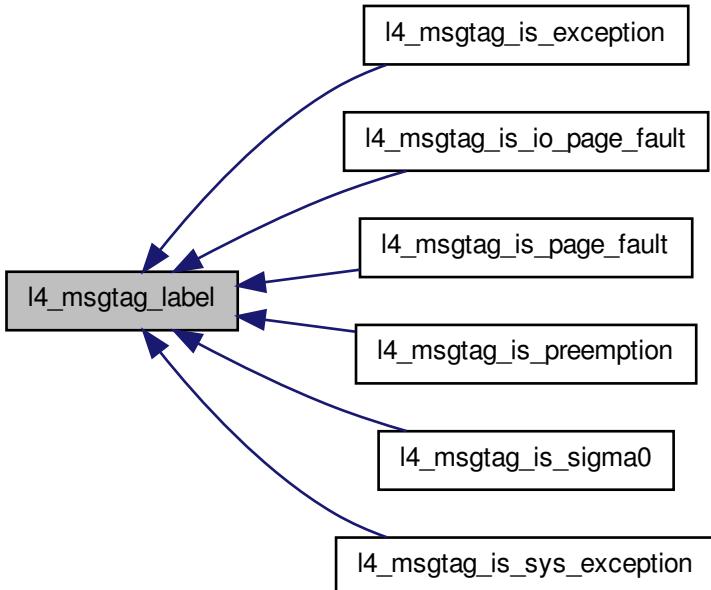
`t` The tag

Returns

Label

Examples:[examples/sys/singlestep/main.c](#), and [examples/sys/start-with-exc/main.c](#).Definition at line 377 of file [types.h](#).Referenced by [l4_mshtag_is_exception\(\)](#), [l4_mshtag_is_io_page_fault\(\)](#), [l4_mshtag_is_page_fault\(\)](#), [l4_mshtag_is_preemption\(\)](#), [l4_mshtag_is_sigma0\(\)](#), and [l4_mshtag_is_sys_exception\(\)](#).

Here is the caller graph for this function:

**9.64.4.3 unsigned l4_mshtag_words (l4_mshtag_t *t*) throw () [inline]**

Get the number of untyped words.

Parameters*t* The tag**Returns**

Number of words

Examples:[examples/sys/utcb-ipc/main.c](#).

Definition at line 381 of file [types.h](#).

9.64.4.4 `unsigned l4_mshtag_items (l4_mshtag_t t) throw () [inline]`

Get the number of typed items.

Parameters

t The tag

Returns

Number of items.

Definition at line 385 of file [types.h](#).

9.64.4.5 `unsigned l4_mshtag_flags (l4_mshtag_t t) throw () [inline]`

Get the flags.

The flag are defined by [l4_mshtag_flags](#).

Parameters

t the tag

Returns

Flags

Definition at line 389 of file [types.h](#).

9.64.4.6 `unsigned l4_mshtag_has_error (l4_mshtag_t t) throw () [inline]`

Test for error indicator flag.

Parameters

t the tag

Returns

>0 for yes, 0 for no

Return whether the kernel operation caused a communication error, e.g. with IPC. if true: `utcb->error` is valid, otherwise `utcb->error` is not valid

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 394 of file [types.h](#).

Referenced by [l4_ipc_error\(\)](#).

Here is the caller graph for this function:



9.64.4.7 `unsigned l4_mshtag_is_page_fault(l4_mshtag_t t) throw() [inline]`

Test for page-fault protocol.

Parameters

t the tag

Returns

Boolean value

Definition at line 399 of file [types.h](#).

References [l4_mshtag_label\(\)](#).

Here is the call graph for this function:



9.64.4.8 `unsigned l4_mshtag_is_preemption(l4_mshtag_t t) throw() [inline]`

Test for preemption protocol.

Parameters

t the tag

Returns

Boolean value

Definition at line 402 of file [types.h](#).

References [l4_msntag_label\(\)](#).

Here is the call graph for this function:



9.64.4.9 `unsigned l4_msntag_is_sys_exception (l4_msntag_t t) throw () [inline]`

Test for system-exception protocol.

Parameters

t the tag

Returns

Boolean value

Definition at line 405 of file [types.h](#).

References [l4_msntag_label\(\)](#).

Here is the call graph for this function:



9.64.4.10 `unsigned l4_msntag_is_exception (l4_msntag_t t) throw () [inline]`

Test for exception protocol.

Parameters

t the tag

Returns

Boolean value

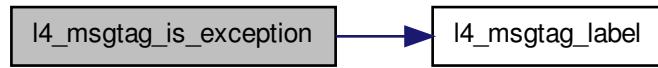
Examples:

[examples/sys/aliens/main.c](#), [examples/sys/singlestep/main.c](#), and [examples/sys/start-with-exc/main.c](#).

Definition at line 408 of file [types.h](#).

References [l4_msntag_label\(\)](#).

Here is the call graph for this function:

**9.64.4.11 unsigned l4_msntag_is_sigma0 (l4_msntag_t t) throw () [inline]**

Test for sigma0 protocol.

Parameters

t the tag

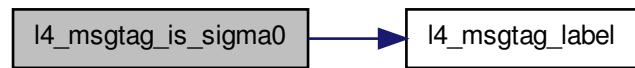
Returns

Boolean value

Definition at line 411 of file [types.h](#).

References [l4_msntag_label\(\)](#).

Here is the call graph for this function:



9.64.4.12 `unsigned l4_mshtag_is_io_page_fault(l4_mshtag_t t) throw() [inline]`

Test for IO-page-fault protocol.

Parameters

t the tag

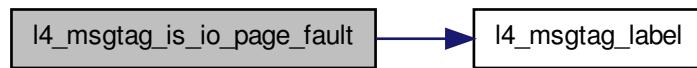
Returns

Boolean value

Definition at line 414 of file [types.h](#).

References [l4_mshtag_label\(\)](#).

Here is the call graph for this function:



9.65 Capabilities

Functions and definitions related to capabilities.

Collaboration diagram for Capabilities:



Data Structures

- class [L4::Cap_base](#)
Base class for all kinds of capabilities.
- class [L4::Cap< T >](#)
Capability Selector a la C++.

Typedefs

- `typedef unsigned long l4_cap_idx_t`

L4 Capability selector Type.

Enumerations

- `enum l4_cap_consts_t { L4_CAP_SHIFT, L4_CAP_SIZE, L4_CAP_MASK, L4_INVALID_CAP }`

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 }`

Default capabilities setup for the initial tasks.

Functions

- `template<typename T, typename F>`
`Cap< T > L4::cap_cast (Cap< F > const &c) throw ()`

static_cast for capabilities.

- `template<typename T, typename F>`
`Cap< T > L4::cap_reinterpret_cast (Cap< F > const &c) throw ()`

reinterpret_cast for capabilities.

- `template<typename T, typename F>`
`Cap< T > L4::cap_dynamic_cast (Cap< F > const &c) throw ()`

dynamic_cast for capabilities.

- `unsigned l4_is_invalid_cap (l4_cap_idx_t c) throw ()`

Test if a capability selector is the invalid capability.

- `unsigned l4_is_valid_cap (l4_cap_idx_t c) throw ()`

Test if a capability selector is a valid selector.

- `unsigned l4_capability_equal (l4_cap_idx_t c1, l4_cap_idx_t c2) throw ()`

Test if two capability selectors are equal.

9.65.1 Detailed Description

Functions and definitions related to capabilities. `#include <l4/sys/consts.h>`

C++ interface for capabilities:

```
#include <l4/sys/capability>
```

C interface for capabilities:

```
#include <l4/sys/types.h>
```

9.65.2 Typedef Documentation

9.65.2.1 **typedef unsigned long l4_cap_idx_t**

L4 Capability selector Type.

```
#include <l4/sys/types.h>
```

Definition at line 319 of file [types.h](#).

9.65.3 Enumeration Type Documentation

9.65.3.1 **enum l4_cap_consts_t**

Constants related to capability selectors.

Enumerator:

L4_CAP_SHIFT Capability index shift.

L4_CAP_SIZE 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 134 of file [consts.h](#).

9.65.3.2 **enum l4_default_caps_t**

Default capabilities setup for the initial tasks.

```
#include <l4/sys/consts.h>
```

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

This 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.

L4_BASE_LOG_CAP Capability selector for the log object.

L4_BASE_ICU_CAP Capability selector for the base icu object.

L4_BASE_SCHEDULER_CAP Capability selector for the scheduler cap.

Definition at line 248 of file [consts.h](#).

9.65.4 Function Documentation

9.65.4.1 template<typename T, typename F> Cap<T> L4::cap_cast (Cap<F> const & c) throw () [inline]

static_cast for capabilities.

Parameters

T is the target type of the capability

F is the source type (and is usually implicitly set)

c is 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 adjustment like C++ does on pointers.

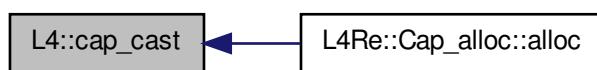
Example code:

```
L4::Cap<L4::Kobject> obj = ... ;
L4::Cap<L4::Icu> icu = L4::cap_cast<L4::Icu>(obj);
```

Definition at line 343 of file [capability](#).

Referenced by [L4Re::Cap_alloc::alloc\(\)](#).

Here is the caller graph for this function:



9.65.4.2 template<typename T, typename F> Cap<T> L4::cap_reinterpret_cast (Cap<F> const & c) throw () [inline]

reinterpret_cast for capabilities.

Parameters

- T is the target type of the capability
- F is the source type (and is usually implicitly set)
- c is 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.

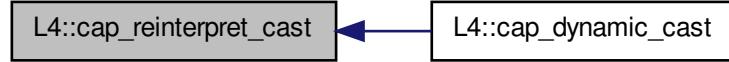
Example code:

```
L4::Cap<L4::Kobject> obj = ... ;
L4::Cap<L4::Icu> icu = L4::cap_reinterpret_cast<L4::Icu>(obj);
```

Definition at line 367 of file [capability](#).

Referenced by [L4::cap_dynamic_cast\(\)](#).

Here is the caller graph for this function:



9.65.4.3 `template<typename T, typename F> Cap<T> L4::cap_dynamic_cast (Cap<F> const & c) throw () [inline]`

dynamic_cast for capabilities.

Parameters

- T is the target type of the capability
- F is the source type (and is usually implicitly set)
- c is the source capability that shall be casted

Returns

A capability typed to the interface T . If the object does not support the target interface T or does not support the [L4::Meta](#) interface the result is the invalid capability selector.

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.

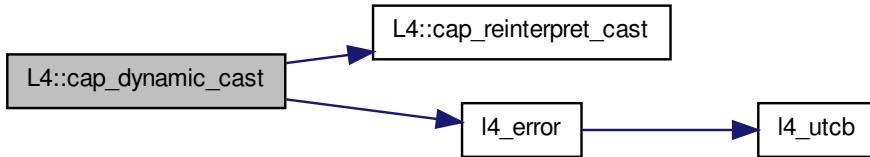
Example code:

```
L4::Cap<L4::Kobject> obj = ... ;
L4::Cap<L4::Icu> icu = L4::cap_dynamic_cast<L4::Icu>(obj);
```

Definition at line 547 of file [capability](#).

References [L4::cap_reinterpret_cast\(\)](#), and [l4_error\(\)](#).

Here is the call graph for this function:



9.65.4.4 `unsigned l4_is_invalid_cap(l4_cap_idx_t c) throw() [inline]`

Test if a capability selector is the invalid capability.

Parameters

c Capability selector

Returns

Boolean value

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 350 of file [types.h](#).

9.65.4.5 `unsigned l4_is_valid_cap(l4_cap_idx_t c) throw() [inline]`

Test if a capability selector is a valid selector.

Parameters

c Capability selector

Returns

Boolean value

Definition at line 354 of file [types.h](#).

9.65.4.6 `unsigned l4_capability_equal (l4_cap_idx_t c1, l4_cap_idx_t c2) throw () [inline]`

Test if two capability selectors are equal.

Parameters

- c1* Capability
- c2* Capability

Returns

1 if equal, 0 if not equal

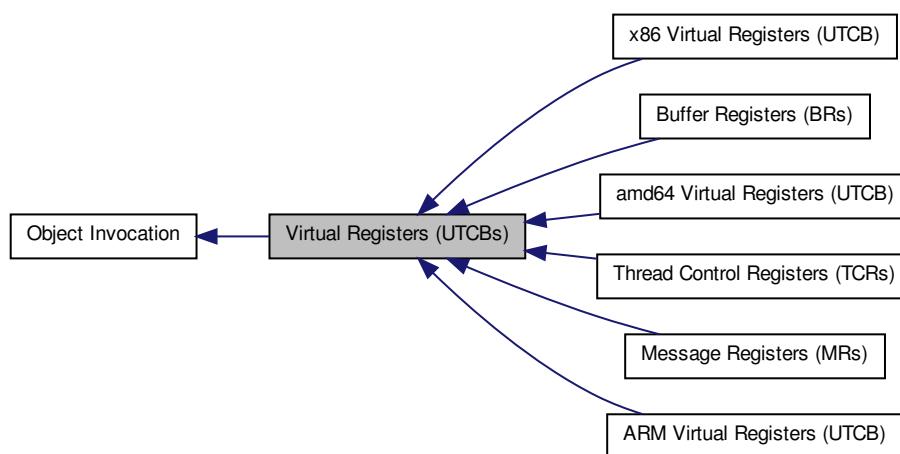
Definition at line 358 of file [types.h](#).

References [L4_CAP_SHIFT](#).

9.66 Virtual Registers (UTCBs)

[L4](#) Virtual Registers (UTCB).

Collaboration diagram for Virtual Registers (UTCBs):



Modules

- [Message Registers \(MRs\)](#)
- [Buffer Registers \(BRs\)](#)
- [Thread Control Registers \(TCRs\)](#)
- [ARM 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 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) throw()`
Get the UTCB address.
- `l4_msg_regs_t * l4_utcb_mr(void) throw()`
Get the message-register block of a UTCB.
- `l4_buf_regs_t * l4_utcb_br(void) throw()`
Get the buffer-register block of a UTCB.
- `l4_thread_regs_t * l4_utcb_tcr(void) throw()`
Get the thread-control-register block of a UTCB.

9.66.1 Detailed Description

[L4](#) Virtual Registers (UTCB). Includes:

```
#include <14/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.

9.66.2 Typedef Documentation

9.66.2.1 `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.

Definition at line 68 of file `utcb.h`.

9.66.3 Function Documentation

9.66.3.1 `l4_msg_regs_t * l4_utcb_mr (void) throw () [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 331 of file `utcb.h`.

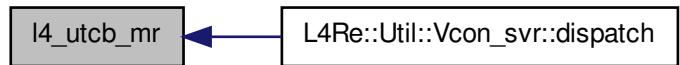
References `l4_utcb()`.

Referenced by `L4Re::Util::Vcon_svr< SVR >::dispatch()`.

Here is the call graph for this function:



Here is the caller graph for this function:



9.66.3.2 `l4_buf_regs_t * l4_utcb_br(void) throw() [inline]`

Get the buffer-register block of a UTCB.

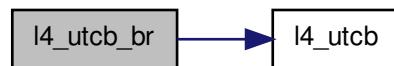
Returns

A pointer to the buffer-register block of `u`.

Definition at line 334 of file [utcb.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.66.3.3 `l4_thread_regs_t * l4_utcb_tcr(void) throw() [inline]`

Get the thread-control-register block of a UTCB.

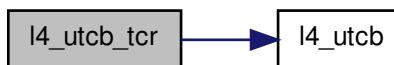
Returns

A pointer to the thread-control-register block of `u`.

Definition at line 337 of file [utcb.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.67 Message Registers (MRs)

Collaboration diagram for Message Registers (MRs):



Data Structures

- struct [l4_msg_regs_t](#)
Encapsulation of the message-register block in the UTCB.

Modules

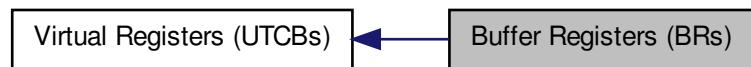
- [Exception registers](#)
Overly definition of the MRs for exception messages.

Typedefs

- [typedef struct l4_msg_regs_t l4_msg_regs_t](#)
Encapsulation of the message-register block in the UTCB.

9.68 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 }`
Constants for buffer descriptors.

Functions

- `void l4_utcb_inherit_fpu (int switch_on) throw ()`
Enable or disable inheritance of FPU state to receiver.

9.68.1 Enumeration Type Documentation

9.68.1.1 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 225 of file `consts.h`.

9.69 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

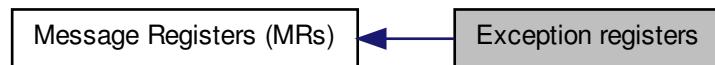
- typedef struct `l4_thread_regs_t` `l4_thread_regs_t`

Encapsulation of the thread-control-register block of the UTCB.

9.70 Exception registers

Overly definition of the MRs for exception messages.

Collaboration diagram for Exception registers:



Functions

- `l4_exc_regs_t * l4_utcb_exc (void) throw ()`
Get the message-register block of a UTCB (for an exception IPC).
- `l4_umword_t l4_utcb_exc_pc (l4_exc_regs_t *u) throw ()`
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) throw ()`
Set the program counter register in the exception state.
- `unsigned long l4_utcb_exc_typeval (l4_exc_regs_t *u) throw ()`
Get the value out of an exception UTCB that describes the type of exception.
- `int l4_utcb_exc_is_pf (l4_exc_regs_t *u) throw ()`
Check whether an exception IPC is a page fault.
- `l4_addr_t l4_utcb_exc_pfa (l4_exc_regs_t *u) throw ()`
Function to get the L4 style page fault address out of an exception.

9.70.1 Detailed Description

Overly definition of the MRs for exception messages.

9.70.2 Function Documentation

9.70.2.1 `l4_exc_regs_t * l4_utcb_exc (void) throw () [inline]`

Get the message-register block of a UTCB (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 [340](#) of file [utcb.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.70.2.2 `l4_umword_t l4_utcb_exc_pc (l4_exc_regs_t * u) throw () [inline]`

Access function to get the program counter of the exception state.

Parameters

`u` 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 [90](#) of file [utcb.h](#).

9.70.2.3 void l4_utcb_exc_pc_set (l4_exc_regs_t * *u*, l4_addr_t *pc*) throw () [inline]

Set the program counter register in the exception state.

Parameters

u UTCB

pc The program counter to set.

Definition at line 95 of file [utcb.h](#).

9.70.2.4 int l4_utcb_exc_is_pf (l4_exc_regs_t * *u*) throw () [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 105 of file [utcb.h](#).

9.71 Virtual Console

Virtual console for simple character based input and output.

Collaboration diagram for Virtual Console:



Data Structures

- class [L4::Vcon](#)

C++ L4 Vcon.

- 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_write_consts { L4_VCON_WRITE_SIZE = (L4_UTCB_GENERIC_DATA_SIZE - 2) * sizeof(l4_umword_t) }`
Constants for l4_vcon_write.
- `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_SET_ATTR_OP = 2UL, L4_VCON_GET_ATTR_OP = 3UL }`
Operations on the vcon objects.

Functions

- `l4_mshtag_t l4_vcon_send (l4_cap_idx_t vcon, char const *buf, int size) L4_NOTHROW`
Send data to virtual console.
- `long l4_vcon_write (l4_cap_idx_t vcon, char const *buf, int size) L4_NOTHROW`
Write data to virtual console.
- `int l4_vcon_read (l4_cap_idx_t vcon, char *buf, int size) L4_NOTHROW`
Read data from virtual console.
- `l4_mshtag_t l4_vcon_set_attr (l4_cap_idx_t vcon, l4_vcon_attr_t const *attr) L4_NOTHROW`
Set attributes of a Vcon.
- `l4_mshtag_t l4_vcon_get_attr (l4_cap_idx_t vcon, l4_vcon_attr_t *attr) L4_NOTHROW`
Get attributes of a Vcon.

9.71.1 Detailed Description

Virtual console for simple character based input and output. #include <14/sys/vcon.h>
 Interrupt for read events are provided by the virtual key interrupt.

9.71.2 Enumeration Type Documentation

9.71.2.1 enum L4_vcon_write_consts

Constants for l4_vcon_write.

Enumerator:

L4_VCON_WRITE_SIZE Maximum size that can be written with one l4_vcon_write call.

Definition at line 83 of file [vcon.h](#).

9.71.2.2 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 126 of file [vcon.h](#).

9.71.2.3 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 ouput CR.

Definition at line 137 of file [vcon.h](#).

9.71.2.4 enum L4_vcon_l_flags

Local flags.

Enumerator:

L4_VCON_ICANON Cannonical mode.

L4_VCON_ECHO Echo input.

Definition at line 148 of file [vcon.h](#).

9.71.2.5 enum L4_vcon_ops

Operations on the vcon objects.

Enumerator:

- L4_VCON_WRITE_OP* Write.
- L4_VCON_SET_ATTR_OP* Get console attributes.
- L4_VCON_GET_ATTR_OP* Set console attributes.

Definition at line 198 of file [vcon.h](#).

9.71.3 Function Documentation

9.71.3.1 l4_mshtag_t l4_vcon_send (l4_cap_idx_t vcon, char const * buf, int size) [inline]

Send data to virtual console.

Parameters

- vcon* Vcon object.
- buf* Pointer to data buffer.
- size* Size of buffer in bytes.

Returns

Syscall return tag

Note

Size must not exceed L4_VCON_WRITE_SIZE.

Definition at line 222 of file [vcon.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.71.3.2 long l4_vcon_write (l4_cap_idx_t vcon, char const * buf, int size) [inline]

Write data to virtual console.

Parameters

vcon Vcon object.
buf Pointer to data buffer.
size Size of buffer in bytes.

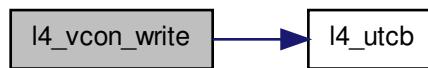
Returns

Number of bytes written to the virtual console.

Definition at line 243 of file [vcon.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.71.3.3 int l4_vcon_read (l4_cap_idx_t vcon, char * buf, int size) [inline]

Read data from virtual console.

Parameters

vcon Vcon object.
buf Pointer to data buffer.
size Size of buffer in bytes.

Returns

Negative error code on error, > size if more to read, size bytes are in the buffer, <= size bytes read

Definition at line 280 of file [vcon.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.71.3.4 `l4_mshtag_t l4_vcon_set_attr (l4_cap_idx_t vcon, l4_vcon_attr_t const * attr) [inline]`

Set attributes of a Vcon.

Parameters

vcon Vcon object.

attr Attribute structure.

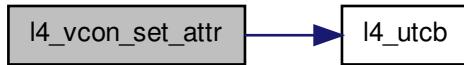
Returns

Syscall return tag

Definition at line 300 of file [vcon.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.71.3.5 `l4_mshtag_t l4_vcon_get_attr (l4_cap_idx_t vcon, l4_vcon_attr_t * attr) [inline]`

Get attributes of a Vcon.

Parameters

vcon Vcon object.

Return values

attr Attribute structure.

Returns

Syscall return tag

Definition at line 324 of file [vcon.h](#).

References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.72 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 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_DEBUG_EXC = 0x08,`
`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 = 0x400 }`
Offsets for vCPU state layouts.

9.72.1 Detailed Description

vCPU API

9.72.2 Enumeration Type Documentation

9.72.2.1 enum L4_vcpu_state_flags

State flags of a vCPU.

Enumerator:

`L4_VCPU_F_IRQ` IRQs (events) enabled.
`L4_VCPU_F_PAGE_FAULTS` Page faults enabled.
`L4_VCPU_F_EXCEPTIONS` Exception enabled.
`L4_VCPU_F_DEBUG_EXC` Debug exception enabled.
`L4_VCPU_F_USER_MODE` User task will be used.
`L4_VCPU_F_FPU_ENABLED` FPU enabled.

Definition at line 55 of file `vcpu.h`.

9.72.2.2 enum L4_vcpu_sticky_flags

Sticky flags of a vCPU.

Enumerator:

L4_VCPU_SF_IRQ_PENDING An event (e.g. IRQ) is pending.

Definition at line 69 of file [vcpu.h](#).

9.72.2.3 enum L4_vcpu_state_offset

Offsets for vCPU state layouts.

Enumerator:

L4_VCPU_OFFSET_EXT_STATE Offset where extended state begins.

Definition at line 78 of file [vcpu.h](#).

9.73 Fiasco-UX Virtual devices

Virtual hardware devices, provided by Fiasco-UX.

Collaboration diagram for Fiasco-UX Virtual devices:



Data Structures

- struct [l4_vhw_entry](#)
Description of a device.
- struct [l4_vhw_descriptor](#)
Virtual hardware devices description.

Enumerations

- enum [l4_vhw_entry_type](#) { [L4_TYPE_VHW_NONE](#), [L4_TYPE_VHW_FRAMEBUFFER](#), [L4_TYPE_VHW_INPUT](#), [L4_TYPE_VHW_NET](#) }
Type of device.

9.73.1 Detailed Description

Virtual hardware devices, provided by Fiasco-UX. #include <14/sys/vhw.h>

9.73.2 Enumeration Type Documentation

9.73.2.1 enum l4_vhw_entry_type

Type of device.

Enumerator:

L4_TYPE_VHW_NONE None entry.

L4_TYPE_VHW_FRAMEBUFFER Framebuffer device.

L4_TYPE_VHW_INPUT Input device.

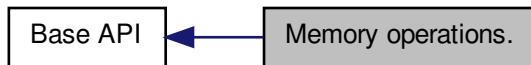
L4_TYPE_VHW_NET Network device.

Definition at line 44 of file [vhw.h](#).

9.74 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 memory from kernel privilege level.
- void **l4_mem_write** (unsigned long virtaddress, unsigned width, unsigned long value)
Write memory from kernel privilege level.

9.74.1 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 <14/sys/mem_op.h>
```

9.74.2 Enumeration Type Documentation

9.74.2.1 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 51 of file [mem_op.h](#).

9.74.3 Function Documentation

9.74.3.1 unsigned long l4_mem_read (unsigned long *virtaddress*, unsigned *width*) [inline]

Read memory from kernel privilege level.

Parameters

virtaddress Virtual address in the calling task.

width Width of access in bytes in log2,

See also

[L4_mem_op_widths](#)

Returns

Read value.

Upon a given invalid address or invalid width value the function does nothing.

Definition at line 141 of file [mem_op.h](#).

9.74.3.2 void l4_mem_write (unsigned long *virtaddress*, unsigned *width*, unsigned long *value*) [inline]

Write memory from kernel privilege level.

Parameters

virtaddress Virtual address in the calling task.

width Width of access in bytes in log2 (i.e. allowed values: 0, 1, 2)

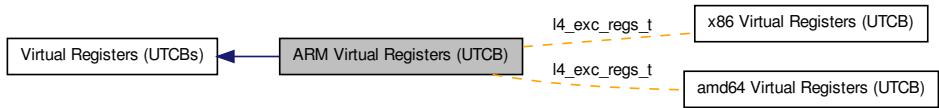
value Value to write.

Upon an given invalid address or invalid width value the function does nothing.

Definition at line 147 of file [mem_op.h](#).

9.75 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.

9.76 VM API for TZ

Virtual Machine API for ARM TrustZone.

Collaboration diagram for VM API for TZ:



Data Structures

- class [L4::Vm](#)
Virtual machine.
- struct [l4_vm_state](#)
state structure for TrustZone VMs

Functions

- [l4_msgtag_t l4_vm_run \(l4_cap_idx_t vm, l4_fpage_t const vm_state_fpage, l4_umword_t *label\)](#)
[L4_NO_THROW](#)
Run a VM.

9.76.1 Detailed Description

Virtual Machine API for ARM TrustZone.

9.76.2 Function Documentation

9.76.2.1 [l4_msgtag_t l4_vm_run \(l4_cap_idx_t vm, l4_fpage_t const vm_state_fpage, l4_umword_t * label \) \[inline\]](#)

Run a VM.

Parameters

vm Capability selector for VM

Definition at line [135](#) of file [vm.h](#).

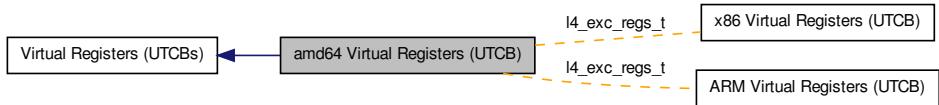
References [l4_utcb\(\)](#).

Here is the call graph for this function:



9.77 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.

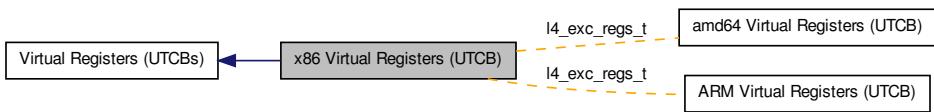
Enumerations

- enum [L4_utcb_consts_amd64](#)

UTCB constants for AMD64.

9.78 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 = 16, 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 }

9.78.1 Enumeration Type Documentation

9.78.1.1 enum L4_utcb_consts_x86

UTCB constants for x86.

Enumerator:

L4_UTCB_EXCEPTION_REGS_SIZE Number of 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.

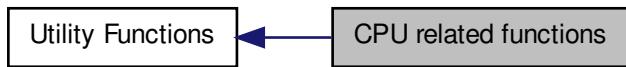
L4_UTCB_INHERIT_FPU BDR flag to accept reception of FPU state.

L4_UTCB_OFFSET Offset of two consecutive UTCBs.

Definition at line 41 of file [utcb.h](#).

9.79 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_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.

9.79.1 Function Documentation

9.79.1.1 `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 66 of file [cpu.h](#).

Referenced by [l4util_cpu_capabilities\(\)](#).

Here is the caller graph for this function:



9.79.1.2 unsigned int l4util_cpu_capabilities (void) [inline]

Returns the CPU capabilities if the "cpuid" instruction is available.

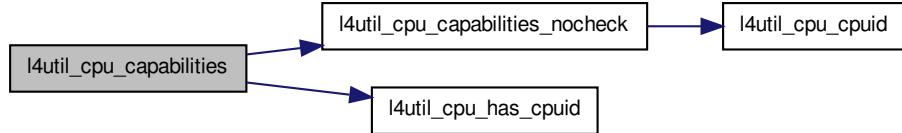
Returns

CPU capabilities if the "cpuid" instruction is available, 0 if the "cpuid" instruction is not supported.

Definition at line 115 of file [cpu.h](#).

References [l4util_cpu_capabilities_nocheck\(\)](#), and [l4util_cpu_has_cpuid\(\)](#).

Here is the call graph for this function:



9.79.1.3 unsigned int l4util_cpu_capabilities_nocheck (void) [inline]

Returns the CPU capabilities.

Returns

CPU capabilities.

Definition at line 104 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:



9.80 Functions to manipulate the local IDT

Collaboration diagram for Functions to manipulate the local IDT:



Data Structures

- struct [l4util_idt_desc_t](#)
IDT entry.
- struct [l4util_idt_header_t](#)
Header of an IDT table.

9.81 Timestamp Counter

Collaboration diagram for Timestamp Counter:



Files

- file [rdtsc.h](#)
time stamp counter related functions
- file [rdtsc.h](#)
time stamp counter related functions

Defines

- `#define L4_TSC_INIT_AUTO 0`
Automatic init.
- `#define L4_TSC_INIT_KERNEL 1`
Initialized by kernel.
- `#define L4_TSC_INIT_CALIBRATE 2`
Initialized by user-level.
- `#define L4_TSC_INIT_AUTO 0`
Automatic init.
- `#define L4_TSC_INIT_KERNEL 1`
Initialized by kernel.
- `#define L4_TSC_INIT_CALIBRATE 2`
Initialized by user-level.

Functions

- `l4_cpu_time_t l4_rdtsc (void)`
Read current value of CPU-internal time stamp counter.

- `l4_uint32_t l4_rdtsc_32 (void)`
Read the least significant 32 bit of the TSC.
- `l4_cpu_time_t l4_rdpmc (int nr)`
Return current value of CPU-internal performance measurement counter.
- `l4_uint32_t l4_rdpmc_32 (int nr)`
Return the least significant 32 bit of a performance counter.
- `l4_uint64_t l4_tsc_to_ns (l4_cpu_time_t tsc)`
Convert time stamp to ns value.
- `l4_uint64_t l4_tsc_to_us (l4_cpu_time_t tsc)`
Convert time stamp 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 time stamp 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 *kip)`
Calibrate scalers for time stamp calculations.
- `l4_uint32_t l4_tsc_init (int constraint, l4_kernel_info_t *kip)`
Initialitze scaler for TSC caliclations.
- `l4_uint32_t l4_get_hz (void)`
Get CPU frequency in Hz.

9.81.1 Function Documentation

9.81.1.1 `l4_cpu_time_t l4_rdtsc (void) [inline]`

Read current value of CPU-internal time stamp counter.

Returns

64-bit time stamp

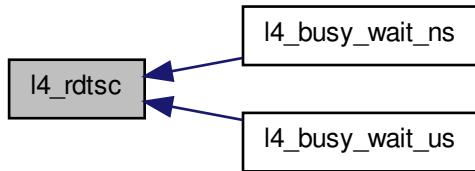
Examples:

`examples/sys/aliens/main.c`.

Definition at line 185 of file [rdtsc.h](#).

Referenced by [l4_busy_wait_ns\(\)](#), and [l4_busy_wait_us\(\)](#).

Here is the caller graph for this function:



9.81.1.2 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 246 of file [rdtsc.h](#).

9.81.1.3 l4_cpu_time_t l4_rdpmc(int nr) [inline]

Return current value of CPU-internal performance measurement counter.

Parameters

nr Number of counter (0 or 1)

Returns

64-bit PMC

Definition at line 205 of file [rdtsc.h](#).

9.81.1.4 l4_uint32_t l4_rdpmc_32(int nr) [inline]

Return the least significant 32 bit of a performance counter.

Useful for smaller differences, needs less cycles.

Definition at line 227 of file [rdtsc.h](#).

9.81.1.5 l4_uint64_t l4_tsc_to_ns(l4_cpu_time_t tsc) [inline]

Convert time stamp to ns value.

Parameters

tsc time value in CPU ticks

Returns

time value in ns

Examples:

[examples/sys/aliens/main.c](#).

Definition at line [260](#) of file [rdtsc.h](#).

9.81.1.6 l4_uint64_t l4_tsc_to_us (l4_cpu_time_t tsc) [inline]

Convert time stamp into micro seconds value.

Parameters

tsc time value in CPU ticks

Returns

time value in micro seconds

Definition at line [274](#) of file [rdtsc.h](#).

9.81.1.7 void l4_tsc_to_s_and_ns (l4_cpu_time_t tsc, l4_uint32_t * s, l4_uint32_t * ns) [inline]

Convert time stamp to s.ns value.

Parameters

tsc time value in CPU ticks

Return values

s seconds

ns nano seconds

Definition at line [288](#) of file [rdtsc.h](#).

9.81.1.8 l4_cpu_time_t l4_ns_to_tsc (l4_uint64_t ns) [inline]

Convert nano seconds into CPU ticks.

Parameters

ns nano seconds

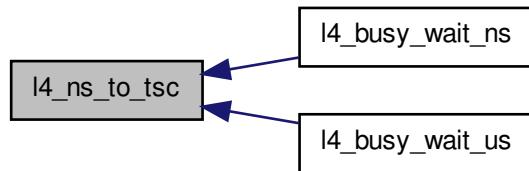
Returns

CPU ticks

Definition at line 303 of file [rdtsc.h](#).

Referenced by [l4_busy_wait_ns\(\)](#), and [l4_busy_wait_us\(\)](#).

Here is the caller graph for this function:



9.81.1.9 void l4_busy_wait_ns (l4_uint64_t ns) [inline]

Wait busy for a small amount of time.

Parameters

ns nano seconds to wait

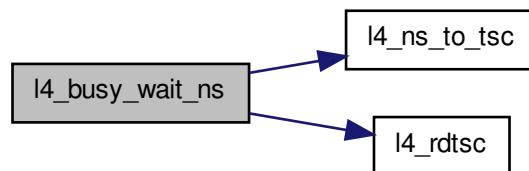
Attention

Not intended for any use!

Definition at line 317 of file [rdtsc.h](#).

References [l4_ns_to_tsc\(\)](#), and [l4_rdtsc\(\)](#).

Here is the call graph for this function:



9.81.1.10 void l4_busy_wait_us (l4_uint64_t us) [inline]

Wait busy for a small amount of time.

Parameters

us micro seconds to wait

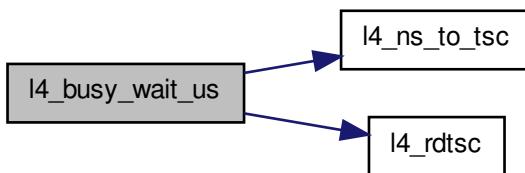
Attention

Not intended for any use!

Definition at line 327 of file [rdtsc.h](#).

References [l4_ns_to_tsc\(\)](#), and [l4_rdtsc\(\)](#).

Here is the call graph for this function:



9.81.1.11 l4_uint32_t l4_calibrate_tsc (l4_kernel_info_t * *kip*) [inline]

Calibrate scalers for time stamp calculations.

Determine some scalers to be able to convert between real time and CPU ticks. This test uses channel 0 of the PIT (i8254) or the kernel KIP, depending on availability. Just calls l4_tsc_init(L4_TSC_INIT_AUTO).

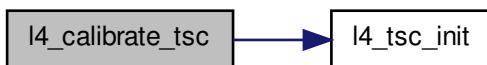
Examples:

[examples/sys/aliens/main.c](#).

Definition at line 179 of file [rdtsc.h](#).

References [l4_tsc_init\(\)](#), and [L4_TSC_INIT_AUTO](#).

Here is the call graph for this function:



9.81.1.12 `l4_uint32_t l4_tsc_init(int constraint, l4_kernel_info_t * kip)`

Initialize scaler for TSC calications.

Initialize the scalers needed by [l4_tsc_to_ns\(\)](#)/l4_ns_to_tsc() and so on. Current versions of Fiasco export these scalers from kernel into userland. The programmer may decide whether he allows to use these scalers or if an calibration should be performed.

Parameters

constraint programmers constraint:

- [L4_TSC_INIT_AUTO](#) if the kernel exports the scalers then use them. If not, perform calibration using channel 0 of the PIT (i8254). The latter case may lead into short (unpredictable) periods where interrupts are disabled.
- [L4_TSC_INIT_KERNEL](#) depend on retrieving the scalers from kernel. If the scalers are not available, return 0.
- [L4_TSC_INIT_CALIBRATE](#) Ignore possible scalers exported by the scaler, instead insist on calibration using the PIT.

kip KIP pointer

Returns

0 on error (no scalers exported by kernel, calibrating failed ...) otherwise returns $(2^{32} / (\text{tsc per tsec}))$. This value has the same semantics as the value returned by the calibrate_delay_loop() function of the Linux kernel.

Initialize the scalers needed by [l4_tsc_to_ns\(\)](#)/l4_ns_to_tsc() and so on. Current versions of Fiasco export these scalers from kernel into userland. The programmer may decide whether he allows to use these scalers or if an calibration should be performed.

Parameters

constraint programmers constraint:

- [L4_TSC_INIT_AUTO](#) if the kernel exports the scalers then use them. If not, perform calibration using channel 0 of the PIT (i8254). The latter case may lead into short (unpredictable) periods where interrupts are disabled.
- [L4_TSC_INIT_KERNEL](#) depend on retrieving the scalers from kernel. If the scalers are not available, return 0.
- [L4_TSC_INIT_CALIBRATE](#) Ignore possible scalers exported by the scaler, instead insist on calibration using the PIT.

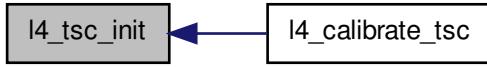
kip KIP pointer

Returns

0 on error (no scalers exported by kernel, calibrating failed ...) otherwise returns $(2^{32} / (\text{tsc per tsec}))$. This value has the same semantics as the value returned by the calibrate_delay_loop() function of the Linux kernel.

Referenced by [l4_calibrate_tsc\(\)](#).

Here is the caller graph for this function:



9.81.1.13 l4_uint32_t l4_get_hz (void)

Get CPU frequency in Hz.

Returns

frequency in Hz

9.82 Atomic Instructions

Collaboration diagram for Atomic Instructions:



Files

- file [atomic.h](#)
atomic operations header and generic implementations

Functions

- int [l4util_cmpxchg64](#) (volatile [l4_uint64_t](#) *dest, [l4_uint64_t](#) cmp_val, [l4_uint64_t](#) new_val)
Atomic compare and exchange (64 bit version).
- 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)

9.82.1 Function Documentation

9.82.1.1 `int l4util_cpxchg64 (volatile l4_uint64_t * dest, l4_uint64_t cmp_val, l4_uint64_t new_val) [inline]`

Atomic compare and exchange (64 bit version).

Parameters

`dest` destination operand

`cmp_val` compare value

`new_val` 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 361 of file `atomic.h`.

9.82.1.2 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

dest destination operand
cmp_val compare value
new_val 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 24 of file [atomic_arch.h](#).

9.82.1.3 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

dest destination operand
cmp_val compare value
new_val 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 375 of file [atomic.h](#).

9.82.1.4 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

dest destination operand
cmp_val compare value
new_val 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 368 of file [atomic.h](#).

9.82.1.5 `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

dest destination operand
cmp_val compare value
new_val new value for dest

Returns

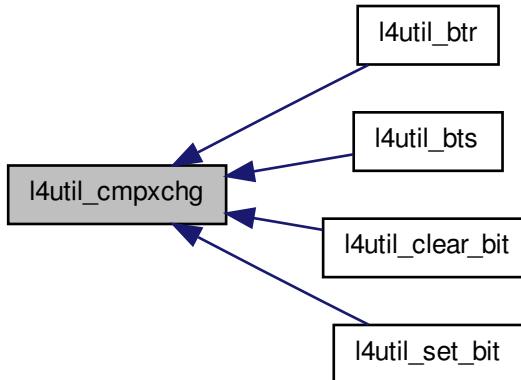
0 if comparison failed, 1 otherwise

Compare the value in *dest* with *cmp_val*, if equal set *dest* to *new_val*

Definition at line 382 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:



9.82.1.6 `l4_uint32_t l4util_xchg32 (volatile l4_uint32_t * dest, l4_uint32_t val) [inline]`

Atomic exchange (32 bit version).

Parameters

dest destination operand
val new value for dest

Returns

old value at destination

Definition at line 389 of file [atomic.h](#).

9.82.1.7 `l4_uint16_t l4util_xchg16 (volatile l4_uint16_t * dest, l4_uint16_t val) [inline]`

Atomic exchange (16 bit version).

Parameters

dest destination operand
val new value for dest

Returns

old value at destination

Definition at line 395 of file [atomic.h](#).

9.82.1.8 `l4_uint8_t l4util_xchg8 (volatile l4_uint8_t * dest, l4_uint8_t val) [inline]`

Atomic exchange (8 bit version).

Parameters

dest destination operand
val new value for dest

Returns

old value at destination

Definition at line 401 of file [atomic.h](#).

9.82.1.9 `l4_umword_t l4util_xchg (volatile l4_umword_t * dest, l4_umword_t val) [inline]`

Atomic exchange (machine wide fields).

Parameters

dest destination operand
val new value for dest

Returns

old value at destination

Definition at line 407 of file [atomic.h](#).

9.82.1.10 `void l4util_add8 (volatile l4_uint8_t * dest, l4_uint8_t val) [inline]`

Parameters

dest destination operand
val value to add/sub/and/or

Definition at line 413 of file [atomic.h](#).

9.82.1.11 `l4_uint8_t l4util_add8_res (volatile l4_uint8_t * dest, l4_uint8_t val) [inline]`**Parameters**

dest destination operand

val value to add/sub/and/or

Returns

res

Definition at line 486 of file [atomic.h](#).

9.82.1.12 `void l4util_inc8 (volatile l4_uint8_t * dest) [inline]`**Parameters**

dest destination operand

Definition at line 311 of file [atomic.h](#).

9.82.1.13 `l4_uint8_t l4util_inc8_res (volatile l4_uint8_t * dest) [inline]`**Parameters**

dest destination operand

Returns

res

Definition at line 337 of file [atomic.h](#).

9.82.1.14 `void l4util_atomic_add (volatile long * dest, long val) [inline]`

Atomic add.

Parameters

dest destination operand

val value to add

Definition at line 54 of file [atomic_arch.h](#).

9.82.1.15 `void l4util_atomic_inc (volatile long * dest) [inline]`

Atomic increment.

Parameters

dest destination operand

Definition at line 61 of file [atomic_arch.h](#).

9.83 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

9.84 Bit Manipulation

Collaboration diagram for Bit Manipulation:



Files

- file `bitops.h`
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` (const unsigned long *val*)
Find the next power of 2 for a given number.

9.84.1 Function Documentation

9.84.1.1 void `l4util_set_bit` (int *b*, volatile `l4_umword_t` * *dest*) [inline]

Set bit in memory.

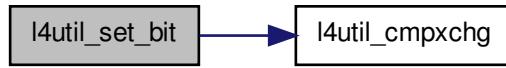
Parameters

- b* bit position
dest destination operand

Definition at line 231 of file `bitops.h`.

References `l4util_cmpxchg()`.

Here is the call graph for this function:



9.84.1.2 void l4util_clear_bit (int *b*, volatile l4_umword_t * *dest*) [inline]

Clear bit in memory.

Parameters

b bit position

dest destination operand

Definition at line 250 of file [bitops.h](#).

References [l4util_cmpxchg\(\)](#).

Here is the call graph for this function:



9.84.1.3 void l4util_complement_bit (int *b*, volatile l4_umword_t * *dest*) [inline]

Complement bit in memory.

Parameters

b bit position

dest destination operand

Definition at line 394 of file [bitops.h](#).

9.84.1.4 `int l4util_test_bit(int b, const volatile l4_umword_t * dest) [inline]`

Test bit (return value of bit).

Parameters

b bit position
dest destination operand

Returns

Value of bit *b*.

Definition at line 268 of file [bitops.h](#).

9.84.1.5 `int l4util_bts(int b, volatile l4_umword_t * dest) [inline]`

Bit test and set.

Parameters

b bit position
dest destination operand

Returns

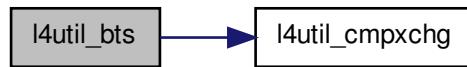
Old value of bit *b*.

Set the *b* bit of *dest* to 1 and return the old value.

Definition at line 291 of file [bitops.h](#).

References [l4util_cmpxchg\(\)](#).

Here is the call graph for this function:



9.84.1.6 `int l4util_btr(int b, volatile l4_umword_t * dest) [inline]`

Bit test and reset.

Parameters

b bit position

dest destination operand

Returns

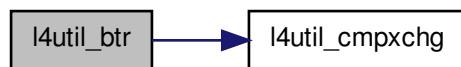
Old value of bit *b*.

Reset bit *b* and return old value.

Definition at line 313 of file [bitops.h](#).

References [l4util_cmpxchg\(\)](#).

Here is the call graph for this function:



9.84.1.7 int l4util_btc (int *b*, volatile l4_umword_t * *dest*) [inline]

Bit test and complement.

Parameters

b bit position

dest destination operand

Returns

Old value of bit *b*.

Complement bit *b* and return old value.

Definition at line 435 of file [bitops.h](#).

9.84.1.8 int l4util_bsr (l4_umword_t *word*) [inline]

Bit scan reverse.

Parameters

word value (machine size)

Returns

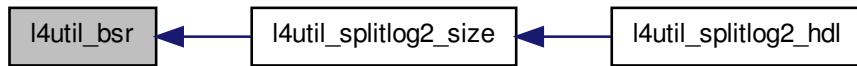
index of most significant set bit in word, -1 if no bit is set (*word* == 0)

"bit scan reverse", find most significant set bit in word (-> LOG2(word))

Definition at line 334 of file [bitops.h](#).

Referenced by [l4util_splitlog2_size\(\)](#).

Here is the caller graph for this function:



9.84.1.9 int l4util_bsf (l4_umword_t word) [inline]

Bit scan forward.

Parameters

word value (machine size)

Returns

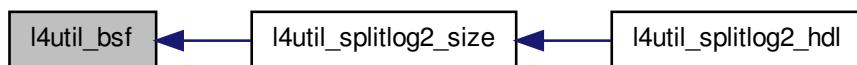
index of least significant bit set in word, -1 if no bit is set (word == 0)

"bit scan forward", find least significant bit set in word.

Definition at line 351 of file [bitops.h](#).

Referenced by [l4util_splitlog2_size\(\)](#).

Here is the caller graph for this function:



9.84.1.10 int l4util_find_first_set_bit (const void * dest, l4_size_t size) [inline]

Find the first set bit in a memory region.

Parameters

dest bit string

size size of string in bits (must be a multiple of 32!)

Returns

number of the first set bit, \geq size if no bit is set

Definition at line 441 of file [bitops.h](#).

9.84.1.11 int l4util_find_first_zero_bit (const void * dest, l4_size_t size) [inline]

Find the first zero bit in a memory region.

Parameters

dest bit string

size size of string in bits (must be a multiple of 32!)

Returns

number of the first zero bit, \geq size if no bit is set

Definition at line 368 of file [bitops.h](#).

9.84.1.12 int l4util_next_power2 (const unsigned long val) [inline]

Find the next power of 2 for a given number.

Parameters

val initial value

Returns

next-highest power of 2

Definition at line 408 of file [bitops.h](#).

9.85 ELF binary format

Functions and types related to ELF binaries.

Collaboration diagram for ELF binary format:



Data Structures

- struct [Elf32_Ehdr](#)
ELF32 header.
- struct [Elf64_Ehdr](#)
ELF64 header.
- struct [Elf32_Shdr](#)
ELF32 section header - figure 1-9, page 1-9.
- 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_Sym](#)
ELF32 symbol table entry.
- struct [Elf64_Sym](#)
ELF64 symbol table entry.

Files

- file [elf.h](#)
ELF definition.

Defines

- #define [EI_NIDENT](#) 16
number of characters
- #define [EI_CLASS](#) 4
ELF class byte index.
- #define [EI_CLASS](#) 4
ELF class byte index.

- #define **ELFCLASSNONE** 0
Invalid ELF class.
- #define **ELFCLASSNONE** 0
Invalid ELF class.
- #define **ELFCLASS32** 1
32-bit objects
- #define **ELFCLASS64** 2
64-bit objects
- #define **ELFCLASSNUM** 3
Mask for 32-bit or 64-bit class.
- #define **EI_DATA** 5
Data encoding byte index.
- #define **EI_DATA** 5
Data encoding byte index.
- #define **ELFDATANONE** 0
Invalid data encoding.
- #define **ELFDATANONE** 0
Invalid data encoding.
- #define **ELFDATA2LSB** 1
2's complement, little endian
- #define **ELFDATA2LSB** 1
2's complement, little endian
- #define **ELFDATA2MSB** 2
2's complement, big endian
- #define **ELFDATA2MSB** 2
2's complement, big endian
- #define **EI_VERSION** 6
File version byte index.
- #define **EI_VERSION** 6
File version byte index.
- #define **EI_OSABI** 7
OS ABI identification.
- #define **EI_OSABI** 7

OS ABI identification.

- `#define ELFOSABI_NONE 0`
UNIX System V ABI.
- `#define ELFOSABI_SYSV 0`
Alias.
- `#define ELFOSABI_SYSV 0`
Alias.
- `#define ELFOSABI_HPUX 1`
HP-UX.
- `#define ELFOSABI_HPUX 1`
HP-UX.
- `#define ELFOSABI_NETBSD 2`
NetBSD.
- `#define ELFOSABI_LINUX 3`
Linux.
- `#define ELFOSABI_SOLARIS 6`
Sun Solaris.
- `#define ELFOSABI_AIX 7`
IBM AIX.
- `#define ELFOSABI_IRIX 8`
SGI Irix.
- `#define ELFOSABI_FREEBSD 9`
FreeBSD.
- `#define ELFOSABI_TRU64 10`
Compaq TRU64 UNIX.
- `#define ELFOSABI_MODESTO 11`
Novell Modesto.
- `#define ELFOSABI_OPENBSD 12`
OpenBSD.
- `#define ELFOSABI_ARM 97`
ARM.
- `#define ELFOSABI_STANDALONE 255`
*Standalone (*embedded*) application.*

- #define **ELFOSABI_STANDALONE** 255
Standalone (embedded) application.
- #define **EL_ABIVERSION** 8
ABI version.
- #define **EL_ABIVERSION** 8
ABI version.
- #define **EL_PAD** 9
Byte index of padding bytes.
- #define **EL_PAD** 9
Byte index of padding bytes.
- #define **ET_NONE** 0
no file type
- #define **ET_REL** 1
relocatable file
- #define **ET_EXEC** 2
executable file
- #define **ET_DYN** 3
shared object file
- #define **ET_CORE** 4
core file
- #define **ET_LOPROC** 0xff00
processor-specific
- #define **ET_HIPROC** 0xffff
processor-specific
- #define **EM_NONE** 0
no machine
- #define **EM_M32** 1
AT&T WE 32100.
- #define **EM_SPARC** 2
SPARC.
- #define **EM_386** 3
Intel 80386.
- #define **EM_68K** 4
Motorola 68000.

- #define **EM_88K** 5
Motorola 88000.
- #define **EM_860** 7
Intel 80860.
- #define **EM_MIPS** 8
MIPS RS3000 big-endian.
- #define **EM_MIPS_RS4_BE** 10
MIPS RS4000 big-endian.
- #define **EM_SPARC64** 11
SPARC 64-bit.
- #define **EM_PARISC** 15
HP PA-RISC.
- #define **EM_VPP500** 17
Fujitsu VPP500.
- #define **EM_SPARC32PLUS** 18
Sun's V8plus.
- #define **EM_960** 19
Intel 80960.
- #define **EM_PPC** 20
PowerPC.
- #define **EM_V800** 36
NEC V800.
- #define **EM_FR20** 37
Fujitsu FR20.
- #define **EM_RH32** 38
TRW RH-32.
- #define **EM_RCE** 39
Motorola RCE.
- #define **EM_ARM** 40
Advanced RISC Machines ARM.
- #define **EM_ALPHA** 41
Digital Alpha.
- #define **EM_SH** 42

Hitachi SuperH.

- #define **EM_SPARCV9** 43
SPARC v9 64-bit.
- #define **EM_TRICORE** 44
Siemens Tricore embedded processor.
- #define **EM_ARC** 45
Argonaut RISC Core, Argonaut Techn Inc.
- #define **EM_H8_300** 46
Hitachi H8/300.
- #define **EM_H8_300H** 47
Hitachi H8/300H.
- #define **EM_H8S** 48
Hitachi H8/S.
- #define **EM_H8_500** 49
Hitachi H8/500.
- #define **EM_IA_64** 50
HP/Intel IA-64.
- #define **EM_MIPS_X** 51
Stanford MIPS-X.
- #define **EM_COLDFIRE** 52
Motorola Coldfire.
- #define **EM_68HC12** 53
Motorola M68HC12.
- #define **EV_NONE** 0
Invalid version.
- #define **EV_CURRENT** 1
Current version.
- #define **EI_MAG0** 0
file id
- #define **EI_MAG1** 1
file id
- #define **EI_MAG2** 2
file id

- #define **EL_MAG3** 3
file id
- #define **ELFMAG0** 0x7f
e_ident[EL_MAG0]
- #define **ELFMAG1** 'E'
e_ident[EL_MAG1]
- #define **ELFMAG2** 'L'
e_ident[EL_MAG2]
- #define **ELFMAG3** 'F'
e_ident[EL_MAG3]
- #define **ELFCLASS32** 1
32-bit object
- #define **ELFCLASS64** 2
64-bit object
- #define **SHN_UNDEF** 0
undefined section header entry
- #define **SHN_LORESERVE** 0xff00
lower bound of reserved indexes
- #define **SHN_LOPROC** 0xff00
lower bound of proc spec entr
- #define **SHN_HIPROC** 0xff1f
upper bound of proc spec entr
- #define **SHN_ABS** 0xffff1
absolute values for ref
- #define **SHN_COMMON** 0xffff2
common symbols
- #define **SHN_HIRESERVE** 0xfffff
upper bound of reserved indexes
- #define **SHT_INIT_ARRAY** 14
Array of constructors.
- #define **SHT_FINI_ARRAY** 15
Array of destructors.
- #define **SHT_PREINIT_ARRAY** 16
Array of pre-constructors.

- #define **SHT_GROUP** 17
Section group.
- #define **SHT_SYMTAB_SHNDX** 18
Extended section indeces.
- #define **SHT_NUM** 19
Number of defined types.
- #define **SHF_WRITE** 0x1
writeable during execution
- #define **SHF_ALLOC** 0x2
section occupies virt memory
- #define **SHF_EXECINSTR** 0x4
code section
- #define **SHF_MERGE** 0x10
Might be merged.
- #define **SHF_STRINGS** 0x20
Contains nul-terminated strings.
- #define **SHF_INFO_LINK** 0x40
'sh_info' contains SHT index
- #define **SHF_LINK_ORDER** 0x80
Preserve order after combining.
- #define **SHF_OS_NONCONFORMING** 0x100
Non-standard OS specific handling required.
- #define **SHF_GROUP** 0x200
Section is member of a group.
- #define **SHF_TLS** 0x400
Section hold thread-local data.
- #define **SHF_MASKOS** 0x0ff000000
OS-specific.
- #define **SHF_MASKPROC** 0xf0000000
proc spec mask
- #define **PT_NULL** 0
array is unused
- #define **PT_LOAD** 1

loadable

- `#define PT_DYNAMIC 2`
dynamic linking information
- `#define PT_INTERP 3`
path to interpreter
- `#define PT_NOTE 4`
auxiliary information
- `#define PT_SHLIB 5`
reserved
- `#define PT_PHDR 6`
location of the pht itself
- `#define PT_TLS 7`
Thread-local storage segment.
- `#define PT_NUM 8`
Number of defined types.
- `#define PT_LOOS 0x60000000`
os spec.
- `#define PT_HIOS 0x6fffffff`
os spec.
- `#define PT_LOPROC 0x70000000`
processor spec.
- `#define PT_HIPROC 0x7fffffff`
processor spec.
- `#define PT_GNU_EH_FRAME (PT_LOOS + 0x474e550)`
EH frame information.
- `#define PT_GNU_STACK (PT_LOOS + 0x474e551)`
Flags for stack.
- `#define PT_GNU_RELRO (PT_LOOS + 0x474e552)`
Read only after reloc.
- `#define PT_L4_STACK (PT_LOOS + 0x12)`
Address of the stack.
- `#define PT_L4_KIP (PT_LOOS + 0x13)`
Address of the KIP.

- #define **PT_L4_AUX** (PT_LOOS + 0x14)
Address of the AUX structures.
- #define **NT_PRSTATUS** 1
Contains copy of prstatus struct.
- #define **NT_FPREGSET** 2
Contains copy of fpregset struct.
- #define **NT_PRPSINFO** 3
Contains copy of prpsinfo struct.
- #define **NT_PRXREG** 4
Contains copy of prxregset struct.
- #define **NT_TASKSTRUCT** 4
Contains copy of task structure.
- #define **NT_PLATFORM** 5
String from sysinfo(SI_PLATFORM).
- #define **NT_AUXV** 6
Contains copy of auxv array.
- #define **NT_GWINDOWS** 7
Contains copy of gwindows struct.
- #define **NT_ASRS** 8
Contains copy of asrset struct.
- #define **NT_PSTATUS** 10
Contains copy of pstatus struct.
- #define **NT_PSINFO** 13
Contains copy of psinfo struct.
- #define **NT_PRCRED** 14
Contains copy of prcred struct.
- #define **NT_UTSNAME** 15
Contains copy of utsname struct.
- #define **NT_LWPSTATUS** 16
Contains copy of lwpstatus struct.
- #define **NT_LWPSINFO** 17
Contains copy of lwpinfo struct.
- #define **NT_PRFPXREG** 20
Contains copy of fprxregset struct.

- #define **NT_VERSION** 1
Contains a version string.
- #define **DT_NULL** 0
Dynamic Array Tags, d_tag - figure 2-10, page 2-12.
- #define **DT_NEEDED** 1
name of a needed library
- #define **DT_PLTRELSZ** 2
total size of relocation entry
- #define **DT_PLTGOT** 3
address assoc with prog link table
- #define **DT_HASH** 4
address of symbol hash table
- #define **DT_STRTAB** 5
address of string table
- #define **DT_SYMTAB** 6
address of symbol table
- #define **DT_REL** 7
address of relocation table
- #define **DT_RELASZ** 8
total size of relocation table
- #define **DT_RELAENT** 9
size of DT_REL relocation entry
- #define **DT_STRSZ** 10
size of the string table
- #define **DT_SYMENT** 11
size of a symbol table entry
- #define **DT_INIT** 12
address of initialization function
- #define **DT_FINI** 13
address of termination function
- #define **DT SONAME** 14
name of the shared object
- #define **DT_RPATH** 15

search library path

- #define **DT_SYMBOLIC** 16
alter symbol resolution algorithm
- #define **DT_REL** 17
address of relocation table
- #define **DT_RELSZ** 18
total size of DT_REL relocation table
- #define **DT_RELENT** 19
size of the DT_REL relocation entry
- #define **DT_PTRREL** 20
type of relocation entry
- #define **DT_DEBUG** 21
for debugging purposes
- #define **DT_TEXTREL** 22
at least on entry changes r/o section
- #define **DT_JMPREL** 23
address of relocation entries
- #define **DT_BIND_NOW** 24
Process relocations of object.
- #define **DT_INIT_ARRAY** 25
Array with addresses of init fct.
- #define **DT_FINI_ARRAY** 26
Array with addresses of fini fct.
- #define **DT_INIT_ARRAYSZ** 27
Size in bytes of DT_INIT_ARRAY.
- #define **DT_FINI_ARRAYSZ** 28
Size in bytes of DT_FINI_ARRAY.
- #define **DT_RUNPATH** 29
Library search path.
- #define **DT_FLAGS** 30
Flags for the object being loaded.
- #define **DT_ENCODING** 32
Start of encoded range.

- #define **DT_PREINIT_ARRAY** 32
Array with addresses of preinit fct.
- #define **DT_PREINIT_ARRAYSZ** 33
size in bytes of DT_PREINIT_ARRAY
- #define **DT_NUM** 34
Number used.
- #define **DT_LOOS** 0x6000000d
Start of OS-specific.
- #define **DT_HIOS** 0x6ffff000
End of OS-specific.
- #define **DT_LOPROC** 0x70000000
processor spec.
- #define **DT_HIPROC** 0x7fffffff
processor spec.
- #define **DF_ORIGIN** 0x00000001
Object may use DF_ORIGIN.
- #define **DF_SYMBOLIC** 0x00000002
Symbol resolutions starts here.
- #define **DF_TEXTREL** 0x00000004
Object contains text relocations.
- #define **DF_BIND_NOW** 0x00000008
No lazy binding for this object.
- #define **DF_STATIC_TLS** 0x00000010
Module uses the static TLS model.
- #define **DF_1_NOW** 0x00000001
Set RTLD_NOW for this object.
- #define **DF_1_GLOBAL** 0x00000002
Set RTLD_GLOBAL for this object.
- #define **DF_1_GROUP** 0x00000004
Set RTLD_GROUP for this object.
- #define **DF_1_NODELETE** 0x00000008
Set RTLD_NODELETE for this object.
- #define **DF_1_LOADFLTR** 0x00000010
Trigger filtee loading at runtime.

- #define **DF_1_INITFIRST** 0x00000020
Set RTLD_INITFIRST for this object.
- #define **DF_1_NOOPEN** 0x00000040
Set RTLD_NOOPEN for this object.
- #define **DF_1_ORIGIN** 0x00000080
\$ORIGIN must be handled.
- #define **DF_1_DIRECT** 0x00000100
Direct binding enabled.
- #define **DF_1_INTERPOSE** 0x00000400
Object is used to interpose.
- #define **DF_1_NODEFLIB** 0x00000800
Ignore default lib search path.
- #define **DF_1_NODUMP** 0x00001000
Object can't be dldump'ed.
- #define **DF_1_CONFALT** 0x00002000
Configuration alternative created.
- #define **DF_1_ENDFILTEE** 0x00004000
Filtee terminates filters search.
- #define **DF_1_DISPRELDNE** 0x00008000
Disp reloc applied at build time.
- #define **DF_1_DISPRELPND** 0x00010000
Disp reloc applied at run-time.
- #define **DF_P1_LAZYLOAD** 0x00000001
Lazyload following object.
- #define **DF_P1_GROUPPERM** 0x00000002
Symbols from next object are not generally available.
- #define **R_386_NONE** 0
none
- #define **R_386_32** 1
S + A.
- #define **R_386_PC32** 2
S + A - P.
- #define **R_386_GOT32** 3

- G + A - P.*
- `#define R_386_PLT32 4`
L + A - P.
 - `#define R_386_COPY 5`
none
 - `#define R_386_GLOB_DAT 6`
S.
 - `#define R_386 JMP_SLOT 7`
S.
 - `#define R_386_RELATIVE 8`
B + A.
 - `#define R_386_GOTOFF 9`
S + A - GOT.
 - `#define R_386_GOTPC 10`
GOT + A - P.
 - `#define STB_LOCAL 0`
not visible outside object file
 - `#define STB_GLOBAL 1`
visible to all objects beeing combined
 - `#define STB_WEAK 2`
resemble global symbols
 - `#define STB_LOOS 10`
os specific
 - `#define STB_HIOS 12`
os specific
 - `#define STB_LOPROC 13`
proc specific
 - `#define STB_HIPROC 15`
proc specific
 - `#define STT_NOTYPE 0`
symbol's type not specified
 - `#define STT_OBJECT 1`
associated with a data object

- `#define STT_FUNC 2`
associated with a function or other code
- `#define STT_SECTION 3`
associated with a section
- `#define STT_FILE 4`
source file name associated with object
- `#define STT_LOOS 10`
os specific
- `#define STT_HIOS 12`
os specific
- `#define STT_LOPROC 13`
proc specific
- `#define STT_HIPROC 15`
proc specific

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

9.85.1 Detailed Description

Functions and types related to ELF binaries.

9.85.2 Define Documentation

9.85.2.1 `#define EI_CLASS 4`

ELF class byte index.

file class

Definition at line 254 of file `elf.h`.

9.85.2.2 `#define EI_CLASS 4`

ELF class byte index.

file class

Definition at line 254 of file `elf.h`.

9.85.2.3 `#define ELFCLASSNONE 0`

Invalid ELF class.

Invalid class.

Definition at line 270 of file `elf.h`.

9.85.2.4 `#define ELFCLASSNONE 0`

Invalid ELF class.

Invalid class.

Definition at line 270 of file `elf.h`.

9.85.2.5 #define EI_DATA 5

Data encoding byte index.

data encoding

Definition at line [255](#) of file `elf.h`.

9.85.2.6 #define EI_DATA 5

Data encoding byte index.

data encoding

Definition at line [255](#) of file `elf.h`.

9.85.2.7 #define ELFDATANONE 0

Invalid data encoding.

invalid data encoding

Definition at line [276](#) of file `elf.h`.

9.85.2.8 #define ELFDATANONE 0

Invalid data encoding.

invalid data encoding

Definition at line [276](#) of file `elf.h`.

9.85.2.9 #define ELFDATA2LSB 1

2's complement, little endian

0x01020304 => [0x04|0x03|0x02|0x01]

Definition at line [277](#) of file `elf.h`.

9.85.2.10 #define ELFDATA2LSB 1

2's complement, little endian

0x01020304 => [0x04|0x03|0x02|0x01]

Definition at line [277](#) of file `elf.h`.

9.85.2.11 #define ELFDATA2MSB 2

2's complement, big endian

0x01020304 => [0x01|0x02|0x03|0x04]

Definition at line [278](#) of file `elf.h`.

9.85.2.12 #define ELFDATA2MSB 2

2's complement, big endian

0x01020304 => [0x01|0x02|0x03|0x04]

Definition at line [278](#) of file [elf.h](#).

9.85.2.13 #define EI_VERSION 6

File version byte index.

file version

Value must be EV_CURRENT

Definition at line [256](#) of file [elf.h](#).

9.85.2.14 #define EI_VERSION 6

File version byte index.

file version

Value must be EV_CURRENT

Definition at line [256](#) of file [elf.h](#).

9.85.2.15 #define EI_OSABI 7

OS ABI identification.

Operating system / ABI identification.

Definition at line [257](#) of file [elf.h](#).

9.85.2.16 #define EI_OSABI 7

OS ABI identification.

Operating system / ABI identification.

Definition at line [257](#) of file [elf.h](#).

9.85.2.17 #define ELFOSABI_SYSV 0

Alias.

UNIX System V ABI (this specification).

Definition at line [282](#) of file [elf.h](#).

9.85.2.18 #define ELFOSABI_SYSV 0

Alias.

UNIX System V ABI (this specification).

Definition at line 282 of file [elf.h](#).

9.85.2.19 #define ELFOSABI_HPUX 1

HP-UX.

HP-UX operating system.

Definition at line 283 of file [elf.h](#).

9.85.2.20 #define ELFOSABI_HPUX 1

HP-UX.

HP-UX operating system.

Definition at line 283 of file [elf.h](#).

9.85.2.21 #define ELFOSABI_NETBSD 2

NetBSD.

Definition at line 174 of file [elf.h](#).

9.85.2.22 #define ELFOSABI_LINUX 3

Linux.

Definition at line 175 of file [elf.h](#).

9.85.2.23 #define ELFOSABI_SOLARIS 6

Sun Solaris.

Definition at line 176 of file [elf.h](#).

9.85.2.24 #define ELFOSABI_AIX 7

IBM AIX.

Definition at line 177 of file [elf.h](#).

9.85.2.25 #define ELFOSABI_IRIX 8

SGI Irix.

Definition at line 178 of file [elf.h](#).

9.85.2.26 #define ELFOSABI_FREEBSD 9

FreeBSD.

Definition at line 179 of file [elf.h](#).

9.85.2.27 #define ELFOSABI_TRU64 10

Compaq TRU64 UNIX.

Definition at line 180 of file [elf.h](#).

9.85.2.28 #define ELFOSABI_MODESTO 11

Novell Modesto.

Definition at line 181 of file [elf.h](#).

9.85.2.29 #define ELFOSABI_OPENBSD 12

OpenBSD.

Definition at line 182 of file [elf.h](#).

9.85.2.30 #define EI_PAD 9

Byte index of padding bytes.

start of padding bytes

Definition at line 259 of file [elf.h](#).

9.85.2.31 #define EI_PAD 9

Byte index of padding bytes.

start of padding bytes

Definition at line 259 of file [elf.h](#).

9.85.2.32 #define EM_ARC 45

Argonaut RISC Core, Argonaut Techn Inc.

Definition at line 226 of file [elf.h](#).

9.85.2.33 #define SHT_NUM 19

Number of defined types.

Definition at line 348 of file [elf.h](#).

9.85.2.34 #define SHF_GROUP 0x200

Section is member of a group.

Definition at line 367 of file [elf.h](#).

9.85.2.35 #define SHF_TLS 0x400

Section hold thread-local data.

Definition at line 368 of file [elf.h](#).

9.85.2.36 #define SHF_MASKOS 0x0ff00000

OS-specific.

Definition at line 369 of file [elf.h](#).

9.85.2.37 #define PT_LOOS 0x60000000

os spec.

Definition at line 412 of file [elf.h](#).

9.85.2.38 #define PT_HIOS 0x6fffffff

os spec.

Definition at line 413 of file [elf.h](#).

9.85.2.39 #define PT_LOPROC 0x70000000

processor spec.

Definition at line 414 of file [elf.h](#).

9.85.2.40 #define PT_HIPROC 0x7fffffff

processor spec.

Definition at line 415 of file [elf.h](#).

9.85.2.41 #define PT_GNU_EH_FRAME (PT_LOOS + 0x474e550)

EH frame information.

Definition at line 417 of file [elf.h](#).

9.85.2.42 #define PT_GNU_STACK (PT_LOOS + 0x474e551)

Flags for stack.

Definition at line 418 of file [elf.h](#).

9.85.2.43 #define PT_GNU_RELRO (PT_LOOS + 0x474e552)

Read only after reloc.

Definition at line 419 of file [elf.h](#).

9.85.2.44 #define PT_L4_STACK (PT_LOOS + 0x12)

Address of the stack.

Definition at line [421](#) of file [elf.h](#).

9.85.2.45 #define PT_L4_KIP (PT_LOOS + 0x13)

Address of the KIP.

Definition at line [422](#) of file [elf.h](#).

9.85.2.46 #define PT_L4_AUX (PT_LOOS + 0x14)

Address of the AUX strcutures.

Definition at line [423](#) of file [elf.h](#).

9.85.2.47 #define NT_VERSION 1

Contains a version string.

Definition at line [454](#) of file [elf.h](#).

9.85.2.48 #define DT_NULL 0

Dynamic Array Tags, d_tag - figure 2-10, page 2-12.

end of _DYNAMIC array

Definition at line [478](#) of file [elf.h](#).

9.85.2.49 #define DT_LOPROC 0x70000000

processor spec.

Definition at line [515](#) of file [elf.h](#).

9.85.2.50 #define DT_HIPROC 0x7fffffff

processor spec.

Definition at line [516](#) of file [elf.h](#).

9.85.2.51 #define DF_1_NOW 0x00000001

Set RTLD_NOW for this object.

Definition at line [527](#) of file [elf.h](#).

9.85.2.52 #define DF_1_GLOBAL 0x00000002

Set RTLD_GLOBAL for this object.

Definition at line [528](#) of file `elf.h`.

9.85.2.53 #define DF_1_GROUP 0x00000004

Set RTLD_GROUP for this object.

Definition at line [529](#) of file `elf.h`.

9.85.2.54 #define DF_1_NODELETE 0x00000008

Set RTLD_NODELETE for this object.

Definition at line [530](#) of file `elf.h`.

9.85.2.55 #define DF_1_LOADFLTR 0x00000010

Trigger filtee loading at runtime.

Definition at line [531](#) of file `elf.h`.

9.85.2.56 #define DF_1_NOOPEN 0x00000040

Set RTLD_NOOPEN for this object.

Definition at line [533](#) of file `elf.h`.

9.85.2.57 #define DF_1_ORIGIN 0x00000080

\$ORIGIN must be handled.

Definition at line [534](#) of file `elf.h`.

9.85.2.58 #define DF_1_DIRECT 0x00000100

Direct binding enabled.

Definition at line [535](#) of file `elf.h`.

9.85.2.59 #define DF_1_INTERPOSE 0x00000400

Object is used to interpose.

Definition at line [537](#) of file `elf.h`.

9.85.2.60 #define DF_1_NODEFLIB 0x00000800

Ignore default lib search path.

Definition at line [538](#) of file `elf.h`.

9.85.2.61 #define DF_1_NODUMP 0x00001000

Object can't be dldump'ed.

Definition at line [539](#) of file [elf.h](#).

9.85.2.62 #define DF_1_CONFALT 0x00002000

Configuration alternative created.

Definition at line [540](#) of file [elf.h](#).

9.85.2.63 #define DF_1_ENDFILTEE 0x00004000

Filtee terminates filters search.

Definition at line [541](#) of file [elf.h](#).

9.85.2.64 #define DF_1_DISPRLDNE 0x00008000

Disp reloc applied at build time.

Definition at line [542](#) of file [elf.h](#).

9.85.2.65 #define DF_1_DISPRLPND 0x00010000

Disp reloc applied at run-time.

Definition at line [543](#) of file [elf.h](#).

9.85.2.66 #define DF_P1_LAZYLOAD 0x00000001

Lazyload following object.

Definition at line [550](#) of file [elf.h](#).

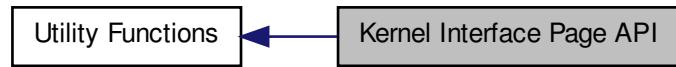
9.85.2.67 #define DF_P1_GROUPPERM 0x00000002

Symbols from next object are not generally available.

Definition at line [551](#) of file [elf.h](#).

9.86 Kernel Interface Page API

Collaboration diagram for Kernel Interface Page API:



Files

- file [kip.h](#)

Defines

- `#define l4util_kip_for_each_feature(s) for (s += strlen(s) + 1; *s; s += strlen(s) + 1)`
Cycle through kernel features given in the KIP.

Functions

- `int l4util_kip_kernel_is_ux (l4_kernel_info_t *)`
Return whether the kernel is running native or under UX.
- `int l4util_kip_kernel_has_feature (l4_kernel_info_t *, const char *str)`
Check if kernel supports a feature.
- `unsigned long l4util_kip_kernel_abi_version (l4_kernel_info_t *)`
Return kernel ABI version.
- `l4_addr_t l4util_memdesc_vm_high (l4_kernel_info_t *kinfo)`
Return end of virtual memory.

9.86.1 Define Documentation

9.86.1.1 `#define l4util_kip_for_each_feature(s) for (s += strlen(s) + 1; *s; s += 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 *) initialized with l4util_kip_version_string().

Definition at line [74](#) of file [kip.h](#).

9.86.2 Function Documentation

9.86.2.1 `int l4util_kip_kernel_is_ux(l4_kernel_info_t *)`

Return whether the kernel is running native or under UX.

Returns whether the kernel is running natively or under UX. The KIP will be mapped if not already mapped. The KIP will not be unmapped again.

Returns

1 when running under UX, 0 if not running under UX

9.86.2.2 `int l4util_kip_kernel_has_feature(l4_kernel_info_t *, const char * str)`

Check if kernel supports a feature.

Parameters

str 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. The KIP will be mapped if not already mapped. The KIP will not be unmapped again.

9.86.2.3 `unsigned long l4util_kip_kernel_abi_version(l4_kernel_info_t *)`

Return kernel ABI version.

Returns

Kernel ABI version.

9.86.2.4 `l4_addr_t l4util_memdesc_vm_high(l4_kernel_info_t * kinfo)`

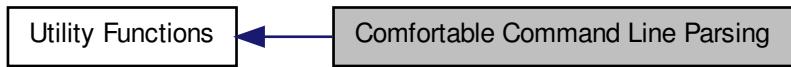
Return end of virtual memory.

Returns

0 if memory descriptor could not be found, last address of address space otherwise

9.87 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

- `int parse_cmdline (int *argc, const char ***argv, char arg0, ...)`
Parse the command-line for specified arguments and store the values into variables.

9.87.1 Function Documentation

9.87.1.1 `int parse_cmdline (int * argc, const char *** argv, char 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 preceded 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 preceded 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.

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

- argc** pointer to number of command line parameters as passed to main
- argv** pointer to array of command line parameters as passed to main
- arg0** format list describing the command line options to parse for

Returns

0 if the command-line was successfully parsed, otherwise:

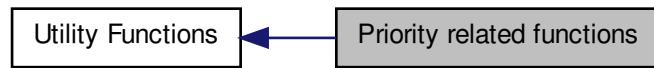
- -1 if the given descriptors are somehow wrong.

- -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.

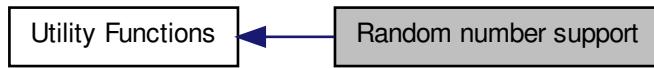
9.88 Priority related functions

Collaboration diagram for Priority related functions:



9.89 Random number support

Collaboration diagram for Random number support:



Functions

- `l4_uint32_t l4util_rand (void)`
Deliver next random number.
- `void l4util_strand (l4_uint32_t seed)`
Initialize random number generator.

9.89.1 Function Documentation

9.89.1.1 `l4_uint32_t l4util_rand (void)`

Deliver next random number.

Returns

A new random number

9.89.1.2 `void l4util_srand (l4_uint32_t seed)`

Initialize random number generator.

Parameters

seed Value to initialize

9.90 Machine Restarting Function

Collaboration diagram for Machine Restarting Function:



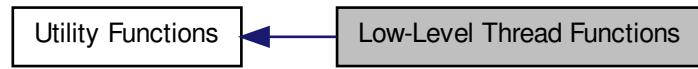
Functions

- `void l4util_reboot (void)`

Machine reboot.

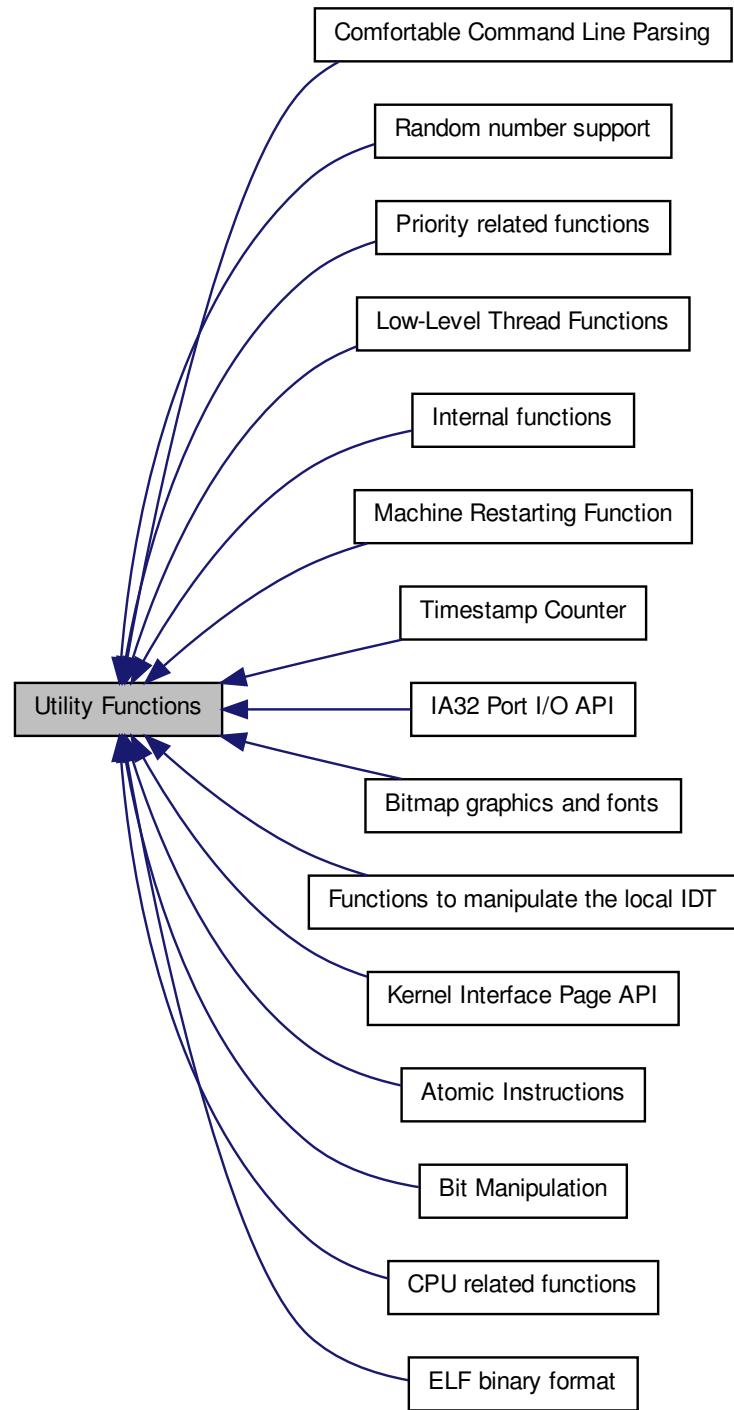
9.91 Low-Level Thread Functions

Collaboration diagram for Low-Level Thread Functions:



9.92 Utility Functions

Collaboration diagram for Utility Functions:



Modules

- CPU related functions
- Functions to manipulate the local IDT
- 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
- Priority related functions
- Random number support
- Machine Restarting Function
- Low-Level Thread Functions
- IA32 Port I/O API
- Bitmap graphics and fonts

This library provides some functions for bitmap handling in frame buffers.

Files

- file `rand.h`

Simple Pseudo-Random Number Generator.

Functions

- `void l4_sleep_forever (void) throw ()`

Go sleep and never wake up.

- `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_timeout_s l4util_micros2l4to (unsigned int mus) throw ()`

Calculate l4 timeouts.

9.92.1 Function Documentation

9.92.1.1 void l4_sleep_forever (void) throw () [inline]

Go sleep and never wake up.

< never timeout

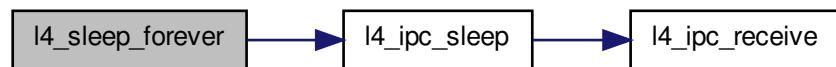
Examples:

[examples/libs/libirq/loop.c](#), and [examples/libs/shmc/prodcons.c](#).

Definition at line 47 of file [util.h](#).

References [L4_IPC_NEVER](#), and [l4_ipc_sleep\(\)](#).

Here is the call graph for this function:



9.92.1.2 long l4util_splitlog2_hdl (l4_addr_t start, l4_addr_t end, long(*)(l4_addr_t s, l4_addr_t e, int log2size) handler) [inline]

Split a range into log2 base and size aligned chunks.

Parameters

start Start of range

end End of range (inclusive) (e.g. 2-4 is len 3)

handler 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.

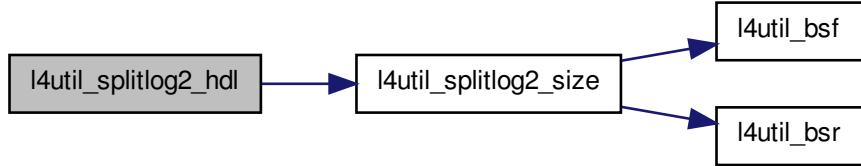
Returns

0 on success, != 0 otherwise

Definition at line 53 of file [splitlog2.h](#).

References [l4util_splitlog2_size\(\)](#).

Here is the call graph for this function:



9.92.1.3 `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

start Start of range

end End of range (inclusive) (e.g. 2-4 is len 3)

Returns

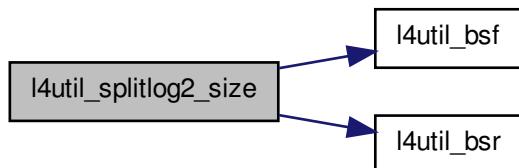
length of elements in log2size (length is $1 << \log2size$)

Definition at line 72 of file [splitlog2.h](#).

References [l4util_bsfc\(\)](#), and [l4util_bsr\(\)](#).

Referenced by [l4util_splitlog2_hdl\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.92.1.4 l4_timeout_s l4util_micros2l4to (unsigned int *mus*) throw ()

Calculate l4 timeouts.

Parameters

mus time in microseconds. Special cases:

- 0 -> timeout 0
- ~0U -> timeout NEVER

Returns

the corresponding l4_timeout value

9.93 IA32 Port I/O API

Collaboration diagram for IA32 Port I/O API:



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

9.93.1 Function Documentation

9.93.1.1 `l4_uint8_t l4util_in8 (l4_uint16_t port) [inline]`

Read byte from I/O port.

Parameters

`port` I/O port address

Returns

value

Definition at line 173 of file `port_io.h`.

9.93.1.2 l4_uint16_t l4util_in16 (l4_uint16_t *port*) [inline]

Read 16-bit-value from I/O port.

Parameters

port I/O port address

Returns

value

Definition at line 181 of file [port_io.h](#).

9.93.1.3 l4_uint32_t l4util_in32 (l4_uint16_t *port*) [inline]

Read 32-bit-value from I/O port.

Parameters

port I/O port address

Returns

value

Definition at line 189 of file [port_io.h](#).

9.93.1.4 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

port I/O port address

addr address of buffer

count number of I/O operations

Definition at line 197 of file [port_io.h](#).

9.93.1.5 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

port I/O port address

addr address of buffer

count number of I/O operations

Definition at line 206 of file [port_io.h](#).

9.93.1.6 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

port I/O port address

addr address of buffer

count number of I/O operations

Definition at line 215 of file [port_io.h](#).

9.93.1.7 void l4util_out8 (l4_uint8_t *value*, l4_uint16_t *port*) [inline]

Write byte to I/O port.

Parameters

port I/O port address

value value to write

Definition at line 224 of file [port_io.h](#).

9.93.1.8 void l4util_out16 (l4_uint16_t *value*, l4_uint16_t *port*) [inline]

Write 16-bit-value to I/O port.

Parameters

port I/O port address

value value to write

Definition at line 230 of file [port_io.h](#).

9.93.1.9 void l4util_out32 (l4_uint32_t *value*, l4_uint16_t *port*) [inline]

Write 32-bit-value to I/O port.

Parameters

port I/O port address

value value to write

Definition at line 236 of file [port_io.h](#).

**9.93.1.10 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

port I/O port address

addr address of buffer

count number of I/O operations

Definition at line 242 of file [port_io.h](#).

**9.93.1.11 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

port I/O port address

addr address of buffer

count number of I/O operations

Definition at line 251 of file [port_io.h](#).

**9.93.1.12 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

port I/O port address

addr address of buffer

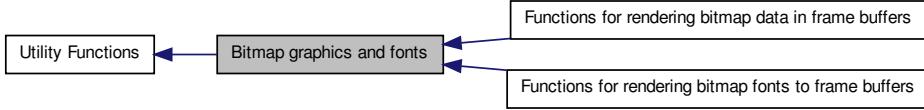
count number of I/O operations

Definition at line 260 of file [port_io.h](#).

9.94 Bitmap graphics and fonts

This library provides some functions for bitmap handling in frame buffers.

Collaboration diagram for Bitmap graphics and fonts:



Modules

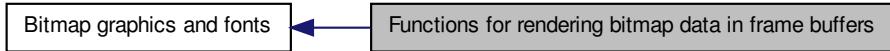
- Functions for rendering bitmap data in frame buffers
- Functions for rendering bitmap fonts to frame buffers

9.94.1 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.

9.95 Functions for rendering bitmap data in frame buffers

Collaboration diagram for Functions for rendering bitmap data in frame buffers:



Data Structures

- struct [gfxbitmap_offset](#)
offsets in pmap[] and bmap[]

TypeDefs

- [typedef unsigned int gfxbitmap_color_t](#)
Standard color type.
- [typedef unsigned int gfxbitmap_color_pix_t](#)
Specific color type.

Functions

- `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_int32_t w, l4_int32_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_int32_t w, l4_int32_t h, l4_int32_t xoffs, l4_int32_t yoffs, l4_uint8_t *pmap, struct gfxbitmap_offset *offset, l4_uint32_t pwidht)`
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.

9.95.1 Typedef Documentation

9.95.1.1 `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 55 of file `bitmap.h`.

9.95.1.2 `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 64 of file `bitmap.h`.

9.95.2 Function Documentation

9.95.2.1 `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.

9.95.2.2 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

vfb Frame buffer.
fbi Frame buffer information structure.
x X position of area.
y Y position of area.
w Width of area.
h Height of area.
color Color of area.

9.95.2.3 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.

Parameters

vfb Frame buffer.
fbi Frame buffer information structure.
x X position of area.
y Y position of area.
w Width of area.
h Height of area.
bmap Bitmap pattern.
fgc Foreground color.
bgc Background color.
offset Offsets.
mode Mode (

See also

`pSLIM_BMAP_START_MSB` and `* #pSLIM_BMAP_START_LSB`).

9.95.2.4 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.

Parameters

vfb Frame buffer.

fbi Frame buffer information structure.

x X position of area.

y Y position of area.

w Width of area.

h Height of area.

pmap Source.

xoffs X offset.

yoffs Y offset.

offset Offsets.

ewidth Width of source in bytes.

9.95.2.5 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

dest Destination frame buffer.

src Source frame buffer.

fbi Frame buffer information structure.

x Source X position of area.

y Source Y position of area.

w Width of area.

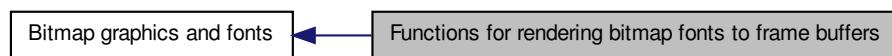
h Height of area.

dx Source X position of area.

dy Source Y position of area.

9.96 Functions for rendering bitmap fonts to frame buffers

Collaboration diagram for Functions for rendering bitmap fonts to frame buffers:



Defines

- #define [GFXBITMAP_DEFAULT_FONT](#) (void *)0

Constant to use for the default font.

Typedefs

- `typedef void * gfxbitmap_font_t`
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.

9.96.1 Enumeration Type Documentation

9.96.1.1 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`.

9.96.2 Function Documentation

9.96.2.1 `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

9.96.2.2 `gfxbitmap_font_t gfxbitmap_font_get (const char * name)`

Get a font descriptor.

Parameters

name Name of the font.

Returns

A (opaque) font descriptor, or NULL if font could not be found.

9.96.2.3 `unsigned gfxbitmap_font_width (gfxbitmap_font_t font)`

Get the font width.

Parameters

font Font.

Returns

Font width, 0 if font width could not be retrieved.

9.96.2.4 `unsigned gfxbitmap_font_height (gfxbitmap_font_t font)`

Get the font height.

Parameters

font Font.

Returns

Font height, 0 if font height could not be retrieved.

9.96.2.5 `void* gfxbitmap_font_data (gfxbitmap_font_t font, unsigned c)`

Get bitmap font data for a specific character.

Parameters

font Font.

c Character.

Returns

Pointer to bmap data, NULL on error.

9.96.2.6 `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

fb Pointer to frame buffer.

fbi Frame buffer info structure.

font Font.

text Text string.

len Length of the text string.

x Horizontal position in the frame buffer.

y Vertical position in the frame buffer.

fg Foreground color.

bg Background color.

9.96.2.7 `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

fb Pointer to frame buffer.

fbi Frame buffer info structure.

font Font.

text Text string.

len Length of the text string.

x Horizontal position in the frame buffer.

y Vertical position in the frame buffer.

fg Foreground color.

bg Background color.

scale_x Horizontal scale factor.

scale_y Vertical scale factor.

9.97 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_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 to a specified region.
- `long l4io_release_iomem (l4_addr_t virt, unsigned long size)`
Release an IO memory region.
- `long l4io_search_iomem_region (l4_addr_t phys, l4_addr_t size, l4_addr_t *rstart, l4_addr_t *rsize)`
Search for a 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.

9.97.1 Typedef Documentation

9.97.1.1 [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 69 of file [types.h](#).

9.97.2 Enumeration Type Documentation

9.97.2.1 [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 16 of file [types.h](#).

9.97.2.2 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 38 of file [types.h](#).

9.97.2.3 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 50 of file [types.h](#).

9.97.3 Function Documentation

9.97.3.1 long l4io_request_iomem (l4_addr_t *phys*, unsigned long *size*, int *flags*, l4_addr_t * *virt*)

Request an IO memory region.

Parameters

phys Physical address of the I/O memory region
size Size of the region in Bytes, granularity pages.
flags See [l4io_iomem_flags_t](#)

Return values

virt Virtual address the region is available at.

Returns

0 on success, <0 on error

Note

This function uses [L4Re](#) functionality to reserve a part of the virtual address space of the caller.

9.97.3.2 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 to a specified region.

Parameters

phys Physical address of the I/O memory region

virt Virtual address.

size Size of the region in Bytes, granularity pages.

flags See [l4io_iomem_flags_t](#)

Returns

0 on success, <0 on error

Note

This function uses [L4Re](#) functionality to reserve a part of the virtual address space of the caller.

9.97.3.3 long l4io_release_iomem (*l4_addr_t virt*, *unsigned long size*)

Release an IO memory region.

Parameters

virt Virtual address of region to free, see [l4io_request_iomem](#)

size Size of the region to release.

Returns

0 on success, <0 on error

9.97.3.4 long l4io_search_iomem_region (*l4_addr_t phys*, *l4_addr_t size*, *l4_addr_t * rstart*, *l4_addr_t * rsize*)

Search for a IO memory region.

Parameters

phys Physical address to look for

size Size of requested memory area

Return values

rstart Start address for region

rsize Size of region in bytes

Returns

0 if an IO region was found, <0 if not

9.97.3.5 long l4io_request_ioport (*unsigned portnum, unsigned len*)

Request an IO port region.

Parameters

portnum Start of port range to request
len Length of range to request

Returns

0 on success, <0 on error

Note

X86 architecture only

9.97.3.6 long l4io_release_ioport (*unsigned portnum, unsigned len*)

Release an IO port region.

Parameters

portnum Start of port range to release
len Length of range to request

Returns

0 on success, <0 on error

Note

X86 architecture only

9.97.3.7 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

devname Name of device

Return values

dev_handle Device handle for found device, can be NULL.
dev Device information, filled by the function, can be NULL.
res_handle Resource handle, can be NULL.

Returns

0 on success, error code otherwise

9.97.3.8 `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

devhandle Device handle.

type Type of resource to request (see #l4io_resource_types_t)

reshandle Resource handle, start with handle returned by device functions.

Return values

reshandle Next resource handle.

res Device descriptor

Returns

0 on success, error code otherwise, esp. -L4_ENOENT if no more resources found

9.97.3.9 `l4_addr_t l4io_request_resource_iomem (l4io_device_handle_t devhandle, l4io_resource_handle_t * reshandle)`

Request IO memory.

Parameters

devhandle Device handle.

Return values

reshandle Resource handle, input and ouput, return next resource handle

Returns

0 on error, virtual address otherwise

9.97.3.10 `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

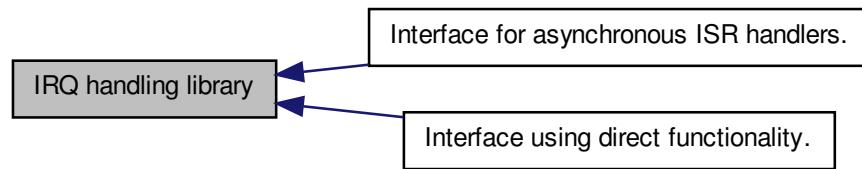
type Type of resource

start Minimal value.

end Maximum value.

9.98 IRQ handling library

Collaboration diagram for IRQ handling library:



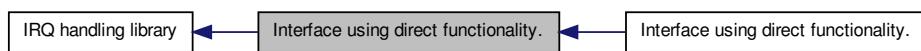
Modules

- Interface using direct functionality.
- Interface for asynchronous ISR handlers.

This interface has just two (main) functions.

9.99 Interface using direct functionality.

Collaboration diagram for Interface using direct functionality.:



Modules

- Interface using direct functionality.

Functions

- `l4irq_t * l4irq_attach (int irqnum)`
Attach/connect to IRQ.
- `l4irq_t * l4irq_attach_ft (int irqnum, unsigned flow_type)`
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 flow_type)`
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.

9.99.1 Function Documentation

9.99.1.1 `l4irq_t* l4irq_attach (int irqnum)`

Attach/connect to IRQ.

Parameters

irqnum 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](#).

9.99.1.2 `l4irq_t* l4irq_attach_ft (int irqnum, unsigned flow_type)`

Attach/connect to IRQ using given type.

Parameters

irqnum IRQ number to request

flow_type Interrupt type,

See also

[L4_irq_flow_type](#)

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.

9.99.1.3 l4irq_t* l4irq_attach_thread (int irqnum, l4_cap_idx_t to_thread)

Attach/connect to IRQ.

Parameters

irqnum IRQ number to request

to_thread 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.

9.99.1.4 l4irq_t* l4irq_attach_thread_ft (int irqnum, l4_cap_idx_t to_thread, unsigned flow_type)

Attach/connect to IRQ using given type.

Parameters

irqnum IRQ number to request

to_thread Attach IRQ to this specified thread.

flow_type Interrupt type,

See also

[L4_irq_flow_type](#)

Returns

Pointer to l4irq_t structure, 0 on error

The pointer to the IRQ structure is used as a label in the IRQ object.

9.99.1.5 long l4irq_wait (l4irq_t * irq)

Wait for specified IRQ.

Parameters

irq IRQ data structure

Returns

0 on success, != 0 on error

Examples:

[examples/libs/libirq/loop.c](#).

9.99.1.6 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

unmask_irq IRQ data structure for unmask.

Return values

ret_irq Received interrupt.

Returns

0 on success, != 0 on error

9.99.1.7 long l4irq_wait_any (l4irq_t ** irq)

Wait for any attached IRQ.

Return values

irq Received interrupt.

Returns

0 on success, != 0 on error

9.99.1.8 long l4irq_unmask (l4irq_t * irq)

Unmask a specific IRQ.

Parameters

irq 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.

9.99.1.9 long l4irq_detach (l4irq_t * *irq*)

Detach from IRQ.

Parameters

irq IRQ data structure

Returns

0 on success, != 0 on error

9.100 Interface for asynchronous ISR handlers.

This interface has just two (main) functions.

Collaboration diagram for Interface for asynchronous ISR handlers.:



Modules

- 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 flow_type)

Attach asynchronous ISR handler to IRQ.

- long [l4irq_release](#) (l4irq_t *irq)

Release asynchronous ISR handler and free resources.

9.100.1 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.

9.100.2 Function Documentation

9.100.2.1 `l4irq_t* l4irq_request (int irqnum, void(*)(void *) isr_handler, void * isr_data, int irq_thread_prio, unsigned flow_type)`

Attach asynchronous ISR handler to IRQ.

Parameters

irqnum IRQ number to request

isr_handler Handler routine that is called when an interrupt triggers

isr_data Pointer given as argument to *isr_handler*

irq_thread_prio L4 thread priority of the ISR handler. Give -1 for same priority as creator.

flow_type Interrupt type,

See also

[L4_irq_flow_type](#)

Returns

Pointer to `l4irq_t` structure, 0 on error

Examples:

[examples/libs/libirq/async_isr.c](#).

9.100.2.2 `long l4irq_release (l4irq_t * irq)`

Release asynchronous ISR handler and free resources.

Parameters

irq IRQ data structure

Returns

0 sucess, != 0 failure

Examples:

[examples/libs/libirq/async_isr.c](#).

9.101 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 flow_type)`
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 flow_type)`
Attach/connect to IRQ using given type.

9.101.1 Function Documentation

9.101.1.1 `l4irq_t* l4irq_attach_cap (l4_cap_idx_t irqcap)`

Attach/connect to IRQ.

Parameters

irqcap 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.

9.101.1.2 `l4irq_t* l4irq_attach_cap_ft (l4_cap_idx_t irqcap, unsigned flow_type)`

Attach/connect to IRQ using given type.

Parameters

irqcap IRQ capability

flow_type Interrupt type,

See also

[L4_irq_flow_type](#)

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.

9.101.1.3 `l4irq_t* l4irq_attach_thread_cap(l4_cap_idx_t irqcap, l4_cap_idx_t to_thread)`

Attach/connect to IRQ.

Parameters

irqcap IRQ capability

to_thread 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.

9.101.1.4 `l4irq_t* l4irq_attach_thread_cap_ft(l4_cap_idx_t irqcap, l4_cap_idx_t to_thread, unsigned flow_type)`

Attach/connect to IRQ using given type.

Parameters

irqcap IRQ capability

to_thread Attach IRQ to this thread.

flow_type Interrupt type,

See also

[L4_irq_flow_type](#)

Returns

Pointer to l4irq_t structure, 0 on error

The pointer to the IRQ structure is used as a label in the IRQ object.

9.102 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 flow_type)`
Attach asynchronous ISR handler to IRQ.

9.102.1 Detailed Description

This group is just an enhanced version to [l4irq_request\(\)](#) which takes a capability object instead of a plain number.

9.102.2 Function Documentation

9.102.2.1 `l4irq_t* l4irq_request_cap (l4_cap_idx_t irqcap, void(*)(void *) isr_handler, void *isr_data, int irq_thread_prio, unsigned flow_type)`

Attach asynchronous ISR handler to IRQ.

Parameters

- irqcap* IRQ capability
- isr_handler* Handler routine that is called when an interrupt triggers
- isr_data* Pointer given as argument to isr_handler
- irq_thread_prio* [L4](#) thread priority of the ISR handler. Give -1 for same priority as creator.
- flow_type* Interrupt type,

See also

[L4_irq_flow_type](#)

Returns

Pointer to l4irq_t structure, 0 on error

9.103 Sigma0 API

Sigma0 API bindings.

Collaboration diagram for Sigma0 API:



Modules

- 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_SMALLERFPAGE` }

Return flags of libsigma0 functions.

Functions

- `l4_kernel_info_t * l4sigma0_map_kip (l4_cap_idx_t sigma0, void *addr, unsigned log2_size)`

Map the kernel info page from pager 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.
- `int l4sigma0_map_tbuf (l4_cap_idx_t sigma0, l4_addr_t virt)`

Request Fiasco trace buffer.
- `void l4sigma0_debug_dump (l4_cap_idx_t sigma0)`

Request sigma0 to dump internal debug information.
- `int l4sigma0_new_client (l4_cap_idx_t sigma0, l4_cap_idx_t gate)`

Create a new IPC gate for a new Sigma0 client.
- `char const * l4sigma0_map_errstr (int err)`

Get a user readable error messages for the return codes.

9.103.1 Detailed Description

Sigma0 API bindings. Convenience bindings for the Sigma0 protocol.

9.103.2 Enumeration Type Documentation

9.103.2.1 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 81 of file [sigma0.h](#).

9.103.3 Function Documentation

9.103.3.1 l4_kernel_info_t* l4sigma0_map_kip (l4_cap_idx_t sigma0, void * addr, unsigned log2_size)

Map the kernel info page from pager to addr.

Parameters

sigma0 Capability selector for the sigma0 gate.

addr Start of the receive window to receive KIP in.

log2_size Size of the receive window to receive KIP in.

Returns

Address KIP was mapped to, 0 indicates an error.

9.103.3.2 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

sigma0 ID of service talking the sigma0 protocol.

phys the physical address of the requested page (must be at least aligned to the minimum page size).

virt the virtual address where the paged should be mapped in the local address space (must be at least aligned to the minimum page size).

size the size of the requested page, this must be a multiple of the minimum page size.

Returns

0 on success, !0 else (see [l4sigma0_map_errstr\(\)](#)).

9.103.3.3 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.

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.

Parameters

sigma0 usually the thread id of sigma0.

phys the physical address to be requested (page aligned).

virt the virtual address where the memory should be mapped to (page aligned).

size the size of the IO memory area to be mapped (multiple of page size)

cached requests cacheable IO memory if 1, and uncached if 0.

Returns

0 on success, !0 else (see [l4sigma0_map_errstr\(\)](#)).

9.103.3.4 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.

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\(\)](#).

Parameters

sigma0 usually the thread id of sigma0.

map_area the base address of the local virtual memory area where the page should be mapped.

log2_map_size 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.

Return values

base physical address of the page received (i.e., the send base of the received mapping if any).

Parameters

sz Size to map by the server, in $2^{\log2_map_size}$ bytes.

Returns

0 on success, !0 else (see [l4sigma0_map_errstr\(\)](#)).

9.103.3.5 int l4sigma0_map_tbuf (l4_cap_idx_t sigma0, l4_addr_t virt)

Request Fiasco trace buffer.

This is a Fiasco specific feature. Where you can request the kernel internal trace buffer for user-level evaluation. This is for special debugging tools, such as Ferret.

Parameters

sigma0 as usual the sigma0 thread id.

virt the virtual address where the trace buffer should be mapped,

Returns

0 on success, !0 else (see [l4sigma0_map_errstr\(\)](#)).

9.103.3.6 void l4sigma0_debug_dump (l4_cap_idx_t sigma0)

Request sigma0 to dump internal debug information.

The debug information, such as internal memory maps, as well as statistics about the internal allocators is dumped to the kernel debugger.

Parameters

sigma0 the sigma0 thread id.

9.103.3.7 int l4sigma0_new_client (l4_cap_idx_t sigma0, l4_cap_idx_t gate)

Create a new IPC gate for a new Sigma0 client.

Parameters

sigma0 Capability selector for sigma0 gate.

gate Capability selector to use for the new gate.

9.103.3.8 char const * l4sigma0_map_errstr (int err) [inline]

Get a user readable error messages for the return codes.

Parameters

err the error code reported by the *map* functions.

Returns

a string containing the error message.

Definition at line [208](#) of file [sigma0.h](#).

9.104 Internal constants

Internal sigma0 definitions.

Collaboration diagram for Internal constants:



Defines

- #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_TBUF** 0xB0
TBUF.
- #define **SIGMA0_REQ_ID_DEBUG_DUMP** 0xC0
Debug dump.
- #define **SIGMA0_REQ_ID_NEW_CLIENT** 0xD0
New client.

- `#define SIGMA0_IS_MAGIC_REQ(d1) ((d1 & SIGMA0_REQ_MASK) == SIGMA0_REQ_MAGIC)`
Check if magic.
- `#define SIGMA0_REQ(x) (SIGMA0_REQ_MAGIC + SIGMA0_REQ_ID_## 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_TBUF (SIGMA0_REQ(TBUF))`
TBUF.
- `#define SIGMA0_REQ_DEBUG_DUMP (SIGMA0_REQ(DEBUG_DUMP))`
Debug dump.
- `#define SIGMA0_REQ_NEW_CLIENT (SIGMA0_REQ(NEW_CLIENT))`
New client.

9.104.1 Detailed Description

Internal sigma0 definitions.

9.105 vCPU Support Library

vCPU handling functionality.

Collaboration diagram for vCPU Support Library:



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.

Modules

- [Extended vCPU support](#)
extended vCPU handling functionality.

Typedefs

- typedef enum [l4vcpu_irq_state_t](#) [l4vcpu_irq_state_t](#)
IRQ/Event enable and disable flags.

Enumerations

- enum [l4vcpu_irq_state_t](#) { [L4VCPU_IRQ_STATE_DISABLED](#) = 0, [L4VCPU_IRQ_STATE_ENABLED](#) = [L4_VCPU_F_IRQ](#) }
IRQ/Event enable and disable flags.

Functions

- [l4vcpu_state_t](#) [l4vcpu_state](#) ([l4_vcpu_state_t](#) const *vcpu) throw()
Return the state flags of a vCPU.
- void [l4vcpu_irq_disable](#) ([l4_vcpu_state_t](#) *vcpu) throw()
Disable a vCPU for event delivery.

- `l4vcpu_irq_state_t l4vcpu_irq_disable_save (l4_vcpu_state_t *vcpu) throw ()`
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_hdl_t do_event_work_cb, l4vcpu_setup_ipc_t setup_ipc) throw ()`
Enable a vCPU for event delivery.
- `void l4vcpu_irq_restore (l4_vcpu_state_t *vcpu, l4vcpu_irq_state_t s, l4_utcb_t *utcb, l4vcpu_event_hdl_t do_event_work_cb, l4vcpu_setup_ipc_t setup_ipc) throw ()`
Restore a previously saved IRQ/event state.
- `void l4vcpu_halt (l4_vcpu_state_t *vcpu, l4_utcb_t *utcb, l4vcpu_event_hdl_t do_event_work_cb, l4vcpu_setup_ipc_t setup_ipc) throw ()`
Halt the vCPU (sleep).
- `void l4vcpu_print_state (l4_vcpu_state_t *vcpu, const char *prefix) throw ()`
Print the state of a vCPU.
- `int l4vcpu_is_irq_entry (l4_vcpu_state_t *vcpu) throw ()`
Return whether the entry reason was an IRQ/IPC message.
- `int l4vcpu_is_page_fault_entry (l4_vcpu_state_t *vcpu) throw ()`
Return whether the entry reason was a page fault.

9.105.1 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.

9.105.2 Enumeration Type Documentation

9.105.2.1 enum l4vcpu_irq_state_t

IRQ/Event enable and disable flags.

Enumerator:

`L4VCPU_IRQ_STATE_DISABLED` IRQ/Event delivery disabled.
`L4VCPU_IRQ_STATE_ENABLED` IRQ/Event delivery enabled.

Definition at line 44 of file `vcpu.h`.

9.105.3 Function Documentation

9.105.3.1 l4vcpu_state_t l4vcpu_state (l4_vcpu_state_t const * vcpu) throw () [inline]

Return the state flags of a vCPU.

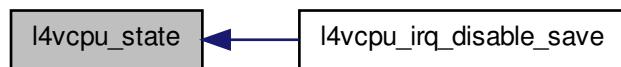
Parameters

vcpu Pointer to vCPU area.

Definition at line 225 of file [vcpu.h](#).

Referenced by [l4vcpu_irq_disable_save\(\)](#).

Here is the caller graph for this function:



9.105.3.2 void l4vcpu_irq_disable (l4_vcpu_state_t * vcpu) throw () [inline]

Disable a vCPU for event delivery.

Parameters

vcpu Pointer to vCPU area.

Definition at line 232 of file [vcpu.h](#).

References [l4_barrier\(\)](#).

Referenced by [l4vcpu_irq_disable_save\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.105.3.3 l4vcpu_irq_state_t l4vcpu_irq_disable_save (l4_vcpu_state_t * vcpu) throw () [inline]

Disable a vCPU for event delivery and return previous state.

Parameters

vcpu Pointer to vCPU area.

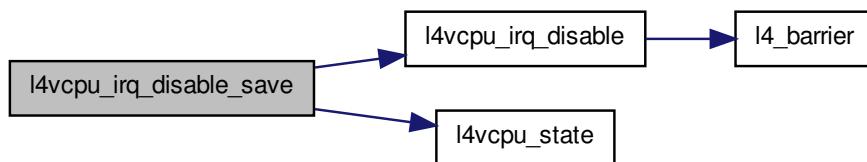
Returns

IRQ state before disabling IRQs.

Definition at line 240 of file [vcpu.h](#).

References [l4vcpu_irq_disable\(\)](#), and [l4vcpu_state\(\)](#).

Here is the call graph for this function:



9.105.3.4 void l4vcpu_irq_enable (l4_vcpu_state_t * vcpu, l4_utcb_t * utcb, l4vcpu_event_hdlr_t do_event_work_cb, l4vcpu_setup_ipc_t setup_ipc) throw () [inline]

Enable a vCPU for event delivery.

Parameters

vcpu Pointer to vCPU area.

utc Utcb pointer of the calling vCPU.

do_event_work_cb Call-back function that is called in case an event (such as an interrupt) is pending.

setup_ipc Function call-back that is called right before any IPC operation.

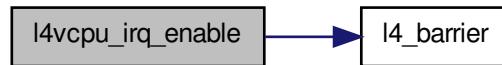
< 0 receive and send timeout

Definition at line 263 of file [vcpu.h](#).

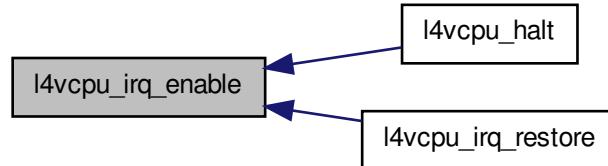
References [EXPECT_TRUE](#), [l4_barrier\(\)](#), [L4_IPC_BOTH_TIMEOUT_0](#), and [L4_VCPU_SF_IRQ_PENDING](#).

Referenced by [l4vcpu_halt\(\)](#), and [l4vcpu_irq_restore\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.105.3.5 `void l4vcpu_irq_restore (l4_vcpu_state_t * vcpu, l4vcpu_irq_state_t s, l4_utcb_t * utc, l4vcpu_event_hdlr_t do_event_work_cb, l4vcpu_setup_ipc_t setup_ipc) throw () [inline]`

Restore a previously saved IRQ/event state.

Parameters

vcpu Pointer to vCPU area.

s IRQ state to be restored.

utc Utcb pointer of the calling vCPU.

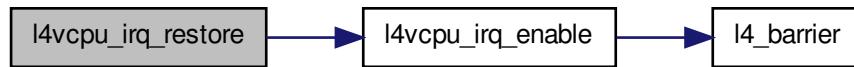
do_event_work_cb Call-back function that is called in case an event (such as an interrupt) is pending after enabling.

setup_ipc Function call-back that is called right before any IPC operation.

Definition at line 282 of file [vcpu.h](#).

References [L4_VCPU_F_IRQ](#), and [l4vcpu_irq_enable\(\)](#).

Here is the call graph for this function:



9.105.3.6 void l4vcpu_halt (l4_vcpu_state_t * vcpu, l4_utcb_t * utcb, l4vcpu_event_hdl_t do_event_work_cb, l4vcpu_setup_ipc_t setup_ipc) throw () [inline]

Halt the vCPU (sleep).

Parameters

vcpu Pointer to vCPU area.

utcb Utcb pointer of the calling vCPU.

do_event_work_cb Call-back function that is called when the vCPU awakes and needs to handle an event/IRQ.

setup_ipc Function call-back that is called right before any IPC operation.

< never timeout

Definition at line 293 of file [vcpu.h](#).

References [L4_IPC_NEVER](#), and [l4vcpu_irq_enable\(\)](#).

Here is the call graph for this function:



9.105.3.7 void l4vcpu_print_state (l4_vcpu_state_t * *vcpu*, const char * *prefix*) throw ()

Print the state of a vCPU.

Parameters

vcpu Pointer to vCPU area.

prefix A prefix for each line printed.

9.105.3.8 int l4vcpu_is_irq_entry (l4_vcpu_state_t * *vcpu*) throw () [inline]

Return whether the entry reason was an IRQ/IPC message.

Parameters

vcpu Pointer to vCPU area.

return 0 if not, !=0 otherwise.

9.105.3.9 int l4vcpu_is_page_fault_entry (l4_vcpu_state_t * *vcpu*) throw () [inline]

Return whether the entry reason was a page fault.

Parameters

vcpu Pointer to vCPU area.

return 0 if not, !=0 otherwise.

9.106 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*) throw ()

Allocate state area for an extended vCPU.

9.106.1 Detailed Description

extended vCPU handling functionality.

9.106.2 Function Documentation

9.106.2.1 `int l4vcpu_ext_alloc (l4_vcpu_state_t ** vcpu, l4_addr_t * ext_state, l4_cap_idx_t task, l4_cap_idx_t regmgr) throw ()`

Allocate state area for an extended vCPU.

Return values

vcpu Allocated vcpu-state area.

ext_state Allocated extended vcpu-state area.

Parameters

task Task to use for allocation.

regmgr Region manager to use for allocation.

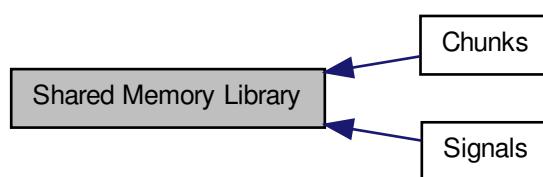
Returns

0 for success, error code otherwise

9.107 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:



Modules

- [Chunks](#)
- [Signals](#)

Functions

- long `l4shmc_create` (const char *shmc_name, `l4_umword_t` shm_size)
Create a shared memory area.
- long `l4shmc_attach` (const char *shmc_name, `l4shmc_area_t` *shmarea)
Attach to a shared memory area.
- long `l4shmc_attach_to` (const char *shmc_name, `l4_umword_t` timeout_ms, `l4shmc_area_t` *shmarea)
Attach to a shared memory area, with limited waiting.
- 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` *shmarea)
Get size of shared memory area.
- long `l4shmc_area_size_free` (`l4shmc_area_t` *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.

9.107.1 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).

9.107.2 Function Documentation

9.107.2.1 long l4shmc_create (*const char * shmc_name, l4_umword_t shm_size*)

Create a shared memory area.

Parameters

shmc_name Name of the shared memory area.
shm_size Size of the whole shared memory area.

Returns

0 on success, <0 on error

Examples:

[examples/libs/shmc/prodcons.c](#).

9.107.2.2 long l4shmc_attach (*const char * shmc_name, l4shmc_area_t * shmarea*) [inline]

Attach to a shared memory area.

Parameters

shmc_name Name of the shared memory area.

Return values

shmarea Pointer to shared memory area descriptor to be filled with information for the shared memory area.

Returns

0 on success, <0 on error

Examples:

[examples/libs/shmc/prodcons.c](#).

9.107.2.3 long l4shmc_attach_to (*const char * shmc_name, l4_umword_t timeout_ms, l4shmc_area_t * shmarea*)

Attach to a shared memory area, with limited waiting.

Parameters

shmc_name Name of the shared memory area.
timeout_ms Timeout to wait for shm area in milliseconds.

Return values

shmarea Pointer to shared memory area descriptor to be filled with information for the shared memory area.

Returns

0 on success, <0 on error

9.107.2.4 long l4shmc_connect_chunk_signal (l4shmc_chunk_t * *chunk*, l4shmc_signal_t * *signal*)

Connect a signal with a chunk.

Parameters

chunk Chunk to attach the signal to.

signal Signal to attach.

Returns

0 on success, <0 on error

Examples:

[examples/libs/shmc/prodcons.c](#).

9.107.2.5 long l4shmc_area_size (l4shmc_area_t * *shmarea*) [inline]

Get size of shared memory area.

Parameters

shmarea Shared memory area.

Returns

<0 on error, otherwise: size of the shared memory area

9.107.2.6 long l4shmc_area_size_free (l4shmc_area_t * *shmarea*)

Get free size of shared memory area.

To get the max size to pass to l4shmc_add_chunk, subtract [l4shmc_chunk_overhead\(\)](#).

Parameters

shmarea Shared memory area.

Returns

<0 on error, otherwise: free capacity in the area.

9.107.2.7 long l4shmc_area_overhead (void)

Get memory overhead per area that is not available for chunks.

Returns

size of the overhead in bytes

9.107.2.8 `long l4shmc_chunk_overhead (void)`

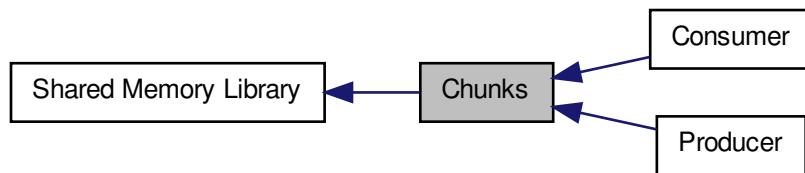
Get memory overhead required in addition to the chunk capacity for adding one chunk.

Returns

size of the overhead in bytes

9.108 Chunks

Collaboration diagram for Chunks:



Modules

- [Producer](#)
- [Consumer](#)

Functions

- `long l4shmc_add_chunk (l4shmc_area_t *shmarea, const char *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, const char *chunk_name, l4shmc_chunk_t *chunk)`
Get chunk out of shared memory area.
- `long l4shmc_get_chunk_to (l4shmc_area_t *shmarea, const char *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 *shmarea, const char **chunk_name, long offs)`
Iterate over names of all existing chunks.
- `void * l4shmc_chunk_ptr (l4shmc_chunk_t *chunk)`
Get data pointer to chunk.

- long [l4shmc_chunk_capacity](#) (l4shmc_chunk_t *chunk)
Get capacity of a chunk.
- l4shmc_signal_t * [l4shmc_chunk_signal](#) (l4shmc_chunk_t *chunk)
Get the signal of a chunk.

9.108.1 Function Documentation

9.108.1.1 long [l4shmc_add_chunk](#) (l4shmc_area_t * *shmarea*, const char * *chunk_name*, l4_umword_t *chunk_capacity*, l4shmc_chunk_t * *chunk*)

Add a chunk in the shared memory area.

Parameters

shmarea The shared memory area to put the chunk in.
chunk_name Name of the chunk.
chunk_capacity Capacity for payload of the chunk in bytes.

Return values

chunk Chunk structure to fill in.

Returns

0 on success, <0 on error

Examples:

[examples/libs/shmc/prodcons.c](#).

9.108.1.2 long [l4shmc_get_chunk](#) (l4shmc_area_t * *shmarea*, const char * *chunk_name*, l4shmc_chunk_t * *chunk*) [inline]

Get chunk out of shared memory area.

Parameters

shmarea Shared memory area.
chunk_name Name of the chunk.

Return values

chunk Chunk data structure to fill.

Returns

0 on success, <0 on error

Examples:

[examples/libs/shmc/prodcons.c](#).

9.108.1.3 long l4shmc_get_chunk_to (l4shmc_area_t * *shmarea*, const char * *chunk_name*, l4_umword_t *timeout_ms*, l4shmc_chunk_t * *chunk*)

Get chunk out of shared memory area, with timeout.

Parameters

shmarea Shared memory area.

chunk_name Name of the chunk.

timeout_ms Timeout in milliseconds to wait for the chunk to appear in the shared memory area.

Return values

chunk chunk data structure to fill.

Returns

0 on success, <0 on error

9.108.1.4 long l4shmc_iterate_chunk (l4shmc_area_t * *shmarea*, const char ** *chunk_name*, long *offs*)

Iterate over names of all existing chunks.

Parameters

shmarea Shared memory area.

chunk_name Where the name of the current chunk will be stored

offs 0 to start iteration, return value of previous call to [l4shmc_iterate_chunk\(\)](#) to get next chunk

Returns

<0 on error, 0 if no more chunks, >0 iterator value for next call

9.108.1.5 void* l4shmc_chunk_ptr (l4shmc_chunk_t * *chunk*) [inline]

Get data pointer to chunk.

Parameters

chunk Chunk.

Returns

0 on success, <0 on error

Examples:

[examples/libs/shmc/prodcons.c](#).

9.108.1.6 `long l4shmc_chunk_capacity (l4shmc_chunk_t * chunk) [inline]`

Get capacity of a chunk.

Parameters

chunk Chunk.

Returns

0 on success, <0 on error

9.108.1.7 `l4shmc_signal_t* l4shmc_chunk_signal (l4shmc_chunk_t * chunk) [inline]`

Get the signal of a chunk.

Parameters

chunk Chunk.

Returns

0 if no signal has been register with this chunk, signal otherwise

9.109 Producer

Collaboration diagram for Producer:



Functions

- `long l4shmc_chunk_try_to_take (l4shmc_chunk_t *chunk)`
Try to mark chunk busy.
- `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 *chunk)`
Check whether chunk is free.

9.109.1 Function Documentation

9.109.1.1 long l4shmc_chunk_try_to_take (*l4shmc_chunk_t* * *chunk*) [inline]

Try to mark chunk busy.

Parameters

chunk chunk to mark.

Returns

0 if chunk could be taken, <0 if not (try again then)

Examples:

[examples/libs/shmc/prodcons.c](#).

9.109.1.2 long l4shmc_chunk_ready (*l4shmc_chunk_t* * *chunk*, *l4_umword_t* *size*) [inline]

Mark chunk as filled (ready).

Parameters

chunk chunk.

size Size of data in the chunk, in bytes.

Returns

0 on success, <0 on error

9.109.1.3 long l4shmc_chunk_ready_sig (*l4shmc_chunk_t* * *chunk*, *l4_umword_t* *size*) [inline]

Mark chunk as filled (ready) and signal consumer.

Parameters

chunk chunk.

size Size of data in the chunk, in bytes.

Returns

0 on success, <0 on error

Examples:

[examples/libs/shmc/prodcons.c](#).

9.109.1.4 long l4shmc_is_chunk_clear (l4shmc_chunk_t * *chunk*) [inline]

Check whether chunk is free.

Parameters

chunk Chunk to check.

Returns

0 on success, <0 on error

9.110 Consumer

Collaboration diagram for Consumer:



Functions

- 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 *chunk)
Check whether data is available.
- long [l4shmc_chunk_size](#) (l4shmc_chunk_t *chunk)
Get current size of a chunk.

9.110.1 Function Documentation

9.110.1.1 long l4shmc_enable_chunk (l4shmc_chunk_t * *chunk*)

Enable a signal connected with a chunk.

Parameters

chunk Chunk to enable.

Returns

0 on success, <0 on 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.

9.110.1.2 long l4shmc_wait_chunk (l4shmc_chunk_t * *chunk*) [inline]

Wait on a specific chunk.

Parameters

chunk Chunk to wait for.

Returns

0 on success, <0 on error

Examples:

[examples/libs/shmc/prodcons.c](#).

9.110.1.3 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

chunk Chunk to check.

timeout Timeout.

Returns

0 on success, <0 on 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.

9.110.1.4 long l4shmc_wait_chunk_try (l4shmc_chunk_t * *chunk*) [inline]

Check whether a specific chunk has an event pending.

Parameters

chunk Chunk to check.

Returns

0 on success, <0 on 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.

9.110.1.5 long l4shmc_chunk_consumed (l4shmc_chunk_t * *chunk*) [inline]

Mark a chunk as free.

Parameters

chunk Chunk to mark as free.

Returns

0 on success, <0 on error

Examples:

[examples/libs/shmc/prodcons.c](#).

9.110.1.6 long l4shmc_is_chunk_ready (l4shmc_chunk_t * *chunk*) [inline]

Check whether data is available.

Parameters

chunk Chunk to check.

Returns

0 on success, <0 on error

9.110.1.7 long l4shmc_chunk_size (l4shmc_chunk_t * *chunk*) [inline]

Get current size of a chunk.

Parameters

chunk Chunk.

Returns

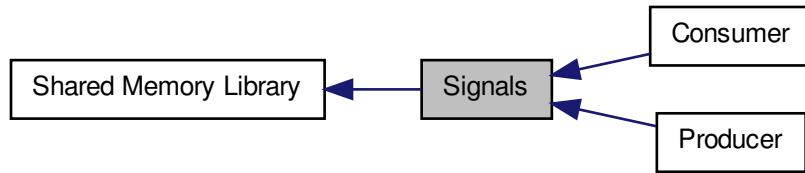
0 on success, <0 on error

Examples:

[examples/libs/shmc/prodcons.c](#).

9.111 Signals

Collaboration diagram for Signals:



Modules

- [Producer](#)
- [Consumer](#)

Functions

- long [l4shmc_add_signal](#) (l4shmc_area_t *shmarea, const char *signal_name, l4shmc_signal_t *signal)
Add a signal for the shared memory area.
- long [l4shmc_attach_signal](#) (l4shmc_area_t *shmarea, const char *signal_name, [l4_cap_idx_t](#) thread, l4shmc_signal_t *signal)
Attach to signal.
- long [l4shmc_attach_signal_to](#) (l4shmc_area_t *shmarea, const char *signal_name, [l4_cap_idx_t](#) thread, [l4_umword_t](#) timeout_ms, l4shmc_signal_t *signal)
Attach to signal, with timeout.
- long [l4shmc_get_signal_to](#) (l4shmc_area_t *shmarea, const char *signal_name, [l4_umword_t](#) timeout_ms, l4shmc_signal_t *signal)
Get signal object from the shared memory area.
- [l4_cap_idx_t l4shmc_signal_cap](#) (l4shmc_signal_t *signal)
Get the signal capability of a signal.
- long [l4shmc_check_magic](#) (l4shmc_chunk_t *chunk)
Check magic value of a chunk.

9.111.1 Function Documentation

9.111.1.1 long l4shmc_add_signal (*l4shmc_area_t* * *shmarea*, *const char* * *signal_name*, *l4shmc_signal_t* * *signal*)

Add a signal for the shared memory area.

Parameters

shmarea The shared memory area to put the chunk in.

signal_name Name of the signal.

Return values

signal Signal structure to fill in.

Returns

0 on success, <0 on error

Examples:

[examples/libs/shmc/prodcons.c](#).

9.111.1.2 long l4shmc_attach_signal (*l4shmc_area_t* * *shmarea*, *const char* * *signal_name*, *l4_cap_idx_t* *thread*, *l4shmc_signal_t* * *signal*) [inline]

Attach to signal.

Parameters

shmarea Shared memory area.

signal_name Name of the signal.

thread Thread capability index to attach the signal to.

Return values

signal Signal data structure to fill.

Returns

0 on success, <0 on error

9.111.1.3 long l4shmc_attach_signal_to (*l4shmc_area_t* * *shmarea*, *const char* * *signal_name*, *l4_cap_idx_t* *thread*, *l4_umword_t* *timeout_ms*, *l4shmc_signal_t* * *signal*)

Attach to signal, with timeout.

Parameters

shmarea Shared memory area.

signal_name Name of the signal.

thread Thread capability index to attach the signal to.

timeout_ms Timeout in milliseconds to wait for the chunk to appear in the shared memory area.

Return values

signal Signal data structure to fill.

Returns

0 on success, <0 on error

Examples:

[examples/libs/shmc/prodcons.c](#).

9.111.1.4 long l4shmc_get_signal_to (l4shmc_area_t * *shmarea*, const char * *signal_name*, l4_umword_t *timeout_ms*, l4shmc_signal_t * *signal*)

Get signal object from the shared memory area.

Parameters

9.111.1.5 l4_cap_idx_t l4shmc_signal_cap (l4shmc_signal_t * *signal*) [inline]

Get the signal capability of a signal.

Parameters

signal Signal.

Returns

Capability of the signal object.

9.111.1.6 long l4shmc_check_magic (l4shmc_chunk_t * *chunk*) [inline]

Check magic value of a chunk.

Parameters

chunk Chunk.

Returns

True if chunk is ok (magic value valid), false if not.

9.112 Producer

Collaboration diagram for Producer:



Functions

- long `l4shmc_trigger` (`l4shmc_signal_t *signal`)

Trigger a signal.

9.112.1 Function Documentation

9.112.1.1 long `l4shmc_trigger` (`l4shmc_signal_t * signal`) [inline]

Trigger a signal.

Parameters

signal Signal to trigger.

Returns

0 on success, <0 on error

Examples:

[examples/libs/shmc/prodcons.c](#).

9.113 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.

9.113.1 Function Documentation

9.113.1.1 long `l4shmc_enable_signal` (`l4shmc_signal_t * signal`)

Enable a signal.

Parameters

signal Signal to enable.

Returns

0 on success, <0 on 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.

9.113.1.2 long `l4shmc_wait_any` (`l4shmc_signal_t ** retsignal`) [inline]

Wait on any signal.

Return values

retsignal Signal received.

Returns

0 on success, <0 on error

9.113.1.3 long l4shmc_wait_any_try (*l4shmc_signal_t* ** *retsignal*) [inline]

Check whether any waited signal has an event pending.

Return values

retsignal Signal that has the event pending if any.

Returns

0 on success, <0 on 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.

9.113.1.4 long l4shmc_wait_any_to (*l4_timeout_t* *timeout*, *l4shmc_signal_t* ** *retsignal*)

Wait for any signal with timeout.

Parameters

timeout Timeout.

Return values

retsignal Signal that has the event pending if any.

Returns

0 on success, <0 on 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.

9.113.1.5 long l4shmc_wait_signal (*l4shmc_signal_t* * *signal*) [inline]

Wait on a specific signal.

Parameters

signal Signal to wait for.

Returns

0 on success, <0 on error

Examples:

[examples/libs/shmc/prodcons.c](#)

9.113.1.6 long l4shmc_wait_signal_to (l4shmc_signal_t * *signal*, l4_timeout_t *timeout*)

Wait on a specific signal, with timeout.

Parameters

signal Signal to wait for.

timeout Timeout.

Returns

0 on success, <0 on error

9.113.1.7 long l4shmc_wait_signal_try (l4shmc_signal_t * *signal*) [inline]

Check whether a specific signal has an event pending.

Parameters

signal Signal to check.

Returns

0 on success, <0 on 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.

9.114 Integer Types

```
#include<l4/sys/l4int.h>
```

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, amd64 version.

- file [l4int.h](#)

Fixed sized integer types, x86 version.

Defines

- `#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 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`
Signed size type.
- `typedef signed int l4_ssize_t`
Unsigned size type.
- `typedef unsigned long l4_size_t`
Signed size type.
- `typedef signed long l4_ssize_t`
Unsigned size type.
- `typedef unsigned int l4_size_t`
Signed size type.
- `typedef signed int l4_ssize_t`
Unsigned size type.

9.114.1 Detailed Description

```
#include<l4/sys/l4int.h>
```

9.114.2 Typedef Documentation

9.114.2.1 `typedef signed char l4_int8_t`

Signed 8bit value.

Definition at line 35 of file [l4int.h](#).

9.114.2.2 `typedef unsigned char l4_uint8_t`

Unsigned 8bit value.

Definition at line 36 of file [l4int.h](#).

9.114.2.3 `typedef signed short int l4_int16_t`

Signed 16bit value.

Definition at line [37](#) of file [l4int.h](#).

9.114.2.4 `typedef unsigned short int l4_uint16_t`

Unsigned 16bit value.

Definition at line [38](#) of file [l4int.h](#).

9.114.2.5 `typedef signed int l4_int32_t`

Signed 32bit value.

Definition at line [39](#) of file [l4int.h](#).

9.114.2.6 `typedef unsigned int l4_uint32_t`

Unsigned 32bit value.

Definition at line [40](#) of file [l4int.h](#).

9.114.2.7 `typedef signed long long l4_int64_t`

Signed 64bit value.

Definition at line [41](#) of file [l4int.h](#).

9.114.2.8 `typedef unsigned long long l4_uint64_t`

Unsigned 64bit value.

Definition at line [42](#) of file [l4int.h](#).

Chapter 10

Namespace Documentation

10.1 cxx Namespace Reference

Various kinds of C++ utilities.

Namespaces

- namespace [Bits](#)

Internal helpers for the cxx package.

Data Structures

- class [Auto_ptr](#)

Smart pointer with automatic deletion.

- class [Avl_map](#)

AVL tree based associative container.

- class [Avl_set](#)

AVL Tree for simple comparable items.

- class [Avl_tree_node](#)

Node of an AVL tree.

- class [Avl_tree](#)

A generic AVL tree.

- class [Bitmap_base](#)

Basic bitmap abstraction.

- class [Bitmap](#)

A static bit map.

- 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 [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 [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.

Functions

- template<typename T1 >
T1 min (T1 a, T1 b)
Get the minimum of a and b.
- template<typename T1 >
T1 max (T1 a, T1 b)
Get the maximum of a and b.

10.1.1 Detailed Description

Various kinds of C++ utilities.

10.2 cxx::Bits Namespace Reference

Internal helpers for the cxx package.

Data Structures

- 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).

10.2.1 Detailed Description

Internal helpers for the cxx package.

10.3 L4 Namespace Reference

[L4](#) low-level kernel interface.

Namespaces

- namespace [Ipc_svr](#)
Helper classes for [L4::Server](#) instantiation.

Data Structures

- 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](#)
Basic server loop for handling client requests.
- class [Server_object](#)
Abstract server object to be used with [L4::Server](#) and [L4::Basic_registry](#).
- class [Basic_registry](#)
This registry returns the corresponding server object based on the label of an [Ipc_gate](#).
- class [String](#)
A null-terminated string container class.
- struct [Type_info](#)
Dynamic Type Information for [L4Re](#) Interfaces.
- 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.
- class [Vm](#)
Virtual machine.

- class [Cap_base](#)
Base class for all kinds of capabilities.
- class [Cap](#)
Capability Selector a la C++.
- class [Kobject](#)
Base class for all kinds of kernel objects, referred to by capabilities.
- class [Debugger](#)
Debugger interface.
- class [Factory](#)
C++ [L4 Factory](#), to create all kinds of kernel objects.
- class [Ipc_gate](#)
[L4 IPC](#) gate.
- class [Irq](#)
C++ version of an [L4 IRQ](#).
- class [Icu](#)
C++ version of an interrupt controller.
- class [Meta](#)
Meta interface that shall be implemented by each [L4Re](#) object and gives access to the dynamic type information for [L4Re](#) objects.
- class [Scheduler](#)
Scheduler object.
- class [Smart_cap](#)
Smart capability class.
- class [Task](#)
An [L4 Task](#).
- class [Thread](#)
[L4 kernel](#) thread.
- class [Vcon](#)
C++ [L4 Vcon](#).

Functions

- template<typename T >
[Type_info](#) const * [kobject_typeid](#) ()
Get the [L4::Type_info](#) for the [L4Re](#) interface given in T.

- template<typename T, typename F>
`Cap< T > cap_cast (Cap< F > const &c) throw ()`
static_cast for capabilities.
- template<typename T, typename F>
`Cap< T > cap_reinterpret_cast (Cap< F > const &c) throw ()`
reinterpret_cast for capabilities.
- template<typename T, typename F>
`Cap< T > cap_dynamic_cast (Cap< F > const &c) throw ()`
dynamic_cast for capabilities.
- template<typename T, typename F, typename SMART>
`Smart_cap< T, SMART > cap_cast (Smart_cap< F, SMART > const &c) throw ()`
static_cast for capabilities.
- template<typename T, typename F, typename SMART>
`Smart_cap< T, SMART > cap_reinterpret_cast (Smart_cap< F, SMART > const &c) throw ()`
reinterpret_cast for capabilities.

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.

10.3.1 Detailed Description

[L4](#) low-level kernel interface.

10.3.2 Function Documentation

10.3.2.1 template<typename T> Type_info const* L4::kobject_typeid() [inline]

Get the [L4::Type_info](#) for the [L4Re](#) interface given in *T*.

Parameters

T 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 87 of file [__typeinfo.h](#).

10.4 L4::Ipc_svr Namespace Reference

Helper classes for [L4::Server](#) instantiation.

Data Structures

- 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 [Default_loop_hooks](#)
Default LOOP_HOOKS.

Enumerations

- enum [Reply_mode](#) { [Reply_compound](#), [Reply_separate](#) }
Reply mode for server loop.

10.4.1 Detailed Description

Helper classes for [L4::Server](#) instantiation.

10.4.2 Enumeration Type Documentation

10.4.2.1 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:

Reply_compound Server shall use a compound reply and wait (fast).

Reply_separate Server shall call reply and wait separately.

Definition at line 44 of file [ipc_server](#).

10.5 L4Re Namespace Reference

[L4](#) Runtime Environment.

Namespaces

- namespace [Vfs](#)

Virtual file system for interfaces POSIX libc.

Data Structures

- class [Cap_alloc](#)

Capability allocator interface.

- class [Smart_cap_auto](#)

Helper for Auto_cap and Auto_del_cap.

- class [Console](#)

Console class.

- class [Dataspace](#)

This class represents a data space.

- class [Debug_obj](#)

Debug interface.

- class [Env](#)

Initial Environment (C++ version).

- class [Event](#)

Event class.

- class [Event_buffer_t](#)

Event buffer class.

- class [Log](#)

Log interface class.

- class [Mem_alloc](#)

Memory allocator.

- class [Namespace](#)
Name-space interface.
- class [Parent](#)
Parent interface.
- class [Rm](#)
Region map.

10.5.1 Detailed Description

[L4](#) Runtime Environment.

10.6 L4Re::Vfs Namespace Reference

Virtual file system for interfaces 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 the 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 for an application.

Functions

- **L4Re::Vfs::Ops** *vfs_ops **asm** ("l4re_env_posix_vfs_ops")
*Reference to the applications **L4Re::Vfs::Ops** singleton.*

10.6.1 Detailed Description

Virtual file system for interfaces POSIX libc.

Chapter 11

Data Structure Documentation

11.1 L4::Alloc_list Class Reference

A simple list-based allocator.

```
#include <alloc.h>
```

11.1.1 Detailed Description

A simple list-based allocator.

Definition at line 33 of file [alloc.h](#).

The documentation for this class was generated from the following file:

- l4/cxx/alloc.h

11.2 L4::Thread::Attr Class Reference

[Thread](#) attributes used for `control_commit()`.

Public Member Functions

- [Attr \(l4_utcb_t *utcbl4_utcb\(\)\) throw \(\)](#)
Create a thread-attribute object with the given UTCB.
- [void `pager` \(Cap< void > const &pager\) throw \(\)](#)
Set the pager capability selector.
- [Cap< void > `pager` \(\) throw \(\)](#)
Get the capability selector used for page-fault messages.
- [void `exc_handler` \(Cap< void > const &exc_handler\) throw \(\)](#)
Set the exception-handler capability selector.

- `Cap< void > exc_handler () throw ()`
Get the capability selector used for exception messages.
- `void bind (l4_utcb_t *thread_utcb, Cap< Task > const &task) throw ()`
Bind the thread to a task.
- `void alien (int on) throw ()`
Set the thread to alien mode.
- `void ux_host_syscall (int on) throw ()`
Allow host system calls on Fiasco-UX.

Friends

- class [L4::Thread](#)

11.2.1 Detailed Description

[Thread](#) attributes used for control_commit(). This class is responsible for initializing various attributes of a thread in a UTCB for the control_commit() method.

See also

[Thread control](#) for some more details.

Definition at line 70 of file [thread](#).

11.2.2 Constructor & Destructor Documentation

11.2.2.1 `L4::Thread::Attr::Attr (l4_utcb_t * utcb = l4_utcb ()) throw () [inline, explicit]`

Create a thread-attribute object with the given UTCB.

Parameters

`utcb` the UTCB to use for the later L4::Thread::control_commit() function. Usually this is the UTCB of the calling thread.

Definition at line 82 of file [thread](#).

11.2.3 Member Function Documentation

11.2.3.1 `void L4::Thread::Attr::pager (Cap< void > const & pager) throw () [inline]`

Set the pager capability selector.

Parameters

pager 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 91 of file [thread](#).

References [pager\(\)](#).

Here is the call graph for this function:



11.2.3.2 Cap<void> L4::Thread::Attr::pager() throw() [inline]

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 99 of file [thread](#).

References [l4_msg_regs_t::mr](#).

Referenced by [pager\(\)](#).

Here is the caller graph for this function:



11.2.3.3 void L4::Thread::Attr::exc_handler(Cap< void > const & exc_handler) throw() [inline]

Set the exception-handler capability selector.

Parameters

pager 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 108 of file [thread](#).

References [exc_handler\(\)](#).

Here is the call graph for this function:



11.2.3.4 Cap<void> L4::Thread::Attr::exc_handler() throw() [inline]

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 116 of file [thread](#).

References [l4_msg_regs_t::mr](#).

Referenced by [exc_handler\(\)](#).

Here is the caller graph for this function:



11.2.3.5 void L4::Thread::Attr::bind(l4_utcb_t * thread_utcb, Cap< Task > const & task) throw() [inline]

Bind the thread to a task.

Parameters

thread_utcb the UTCB address of the thread within the task specified by *task*.

task the capability selector for the task the thread shall be bound to.

Binding a thread to a task means that the thread shall afterwards execute in the given task. To actually start execution you need to use [L4::Thread::ex_regs\(\)](#).

Definition at line 130 of file [thread](#).

11.2.3.6 void L4::Thread::Attr::ux_host_syscall (int on) throw () [inline]

Allow host system calls on Fiasco-UX.

Precondition

Running on Fiasco-UX.

Definition at line 143 of file [thread](#).

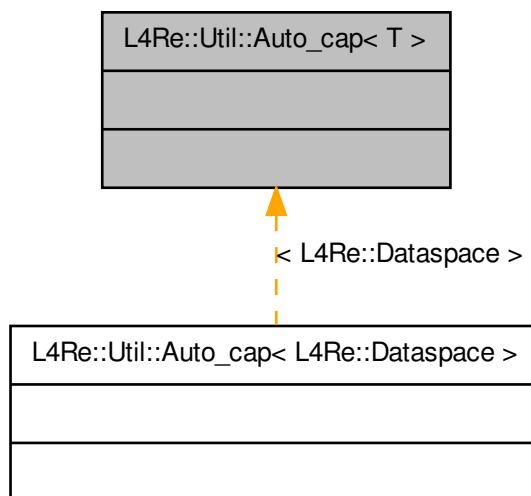
The documentation for this class was generated from the following file:

- [l4/sys/thread](#)

11.3 L4Re::Util::Auto_cap< T > Struct Template Reference

Automatic capability that implements automatic free and unmap of the capability selector.

Inheritance diagram for L4Re::Util::Auto_cap< T >:



11.3.1 Detailed Description

template<typename T> struct L4Re::Util::Auto_cap< T >

Automatic capability that implements automatic free and unmap of the capability selector.

Parameters

T the type of the object that is referred by the capability.

This kind of automatic capability is useful for capabilities with that shall have a lifetime that is strictly coupled to one C++ scope.

Usage:

```
{
    L4Re::Util::Auto_cap<L4Re::Dataspace>::Cap
    ds_cap(L4Re::Util::cap_alloc.alloc<L4Re::Dataspace>());
    // use the dataspace cap
    L4Re::chksys(mem_alloc->alloc(4096, ds_cap.get()));
    ...
    // At the end of the scope ds_cap is unmapped and the capability selector
    // is freed.
}
```

Definition at line 155 of file [cap_alloc](#).

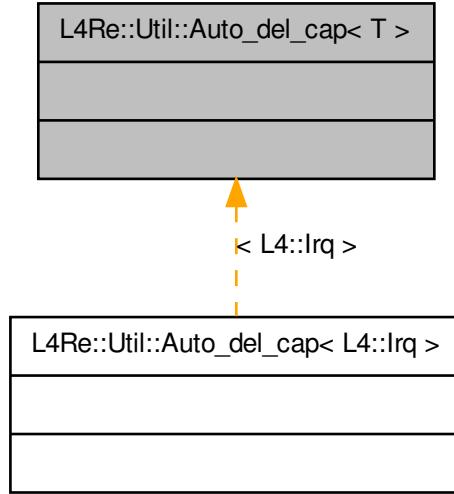
The documentation for this struct was generated from the following file:

- [l4/re/util/cap_alloc](#)

11.4 L4Re::Util::Auto_del_cap< T > Struct Template Reference

Automatic capability that implements automatic free and unmap+delete of the capability selector.

Inheritance diagram for L4Re::Util::Auto_del_cap< T >:



11.4.1 Detailed Description

template<typename T> struct L4Re::Util::Auto_del_cap< T >

Automatic capability that implements automatic free and unmap+delete of the capability selector.

Parameters

T the type of the object that is referred by the capability.

This kind of automatic capability is useful for capabilities with that shall have a lifetime that is strictly coupled to one C++ scope. The main difference to [Auto_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::Auto_del_cap<L4Re::Dataspace>::Cap
    ds_cap(L4Re::Util::cap_alloc.alloc<L4Re::Dataspace>());

    // use the dataspace cap
    L4Re::chksys(mem_alloc->alloc(4096, 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 be
    // also deleted (even if there are other references to this data space).
}

```

Definition at line 189 of file [cap_alloc](#).

The documentation for this struct was generated from the following file:

- [l4/re/util/cap_alloc](#)

11.5 `cxx::Auto_ptr< T >` Class Template Reference

Smart pointer with automatic deletion.

Public Types

- `typedef T Ref_type`
The referenced type.

Public Member Functions

- `Auto_ptr (T *p=0) throw ()`
Construction by assignment of a normal pointer.
- `Auto_ptr (Auto_ptr const &o) throw ()`
Copy construction, releases the original pointer.
- `Auto_ptr & operator=(Auto_ptr const &o) throw ()`
Assignment from another smart pointer.
- `~Auto_ptr () throw ()`
Destruction, shall delete the object.
- `T & operator* () const throw ()`
Dereference the pointer.
- `T * operator-> () const throw ()`
Member access for the object.
- `T * get () const throw ()`
Get the normal pointer.
- `T * release () throw ()`
Release the object and get the normal pointer back.
- `void reset (T *p=0) throw ()`
Delete the object and reset the smart pointer to NULL.
- `operator Priv_type * () const throw ()`
Operator for if (!ptr) ... < >.

11.5.1 Detailed Description

template<typename T> class cxx::Auto_ptr< T >

Smart pointer with automatic deletion.

Parameters

T The type of the referenced object.

This smart pointer calls the delete operator when the destructor is called. This has the effect that the object the pointer points to will be deleted when the pointer goes out of scope, or a new value gets assigned. The smart pointer provides a [release\(\)](#) method to get a normal pointer to the object and set the smart pointer to NULL.

Definition at line [36](#) of file [auto_ptr](#).

11.5.2 Member Typedef Documentation

11.5.2.1 template<typename T> typedef T cxx::Auto_ptr< T >::Ref_type

The referenced type.

Definition at line [41](#) of file [auto_ptr](#).

11.5.3 Constructor & Destructor Documentation

**11.5.3.1 template<typename T> cxx::Auto_ptr< T >::Auto_ptr (T * *p* = *o*) throw ()
[inline, explicit]**

Construction by assignment of a normal pointer.

Parameters

p The pointer to the object

Definition at line [51](#) of file [auto_ptr](#).

**11.5.3.2 template<typename T> cxx::Auto_ptr< T >::Auto_ptr (Auto_ptr< T > const & *o*)
throw () [inline]**

Copy construction, releases the original pointer.

Parameters

o The smart pointer, which shall be copied and released.

Definition at line [57](#) of file [auto_ptr](#).

11.5.3.3 template<typename T> cxx::Auto_ptr< T >::~Auto_ptr () throw () [inline]

Destruction, shall delete the object.

Definition at line [76](#) of file [auto_ptr](#).

11.5.4 Member Function Documentation

11.5.4.1 template<typename T> Auto_ptr& cxx::Auto_ptr< T >::operator= (Auto_ptr< T > const & *o*) throw () [inline]

Assignment from another smart pointer.

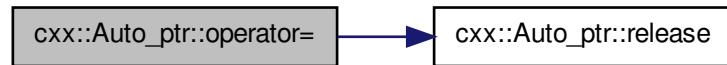
Parameters

o The source for the assignment (will be released).

Definition at line 65 of file [auto_ptr](#).

References [cxx::Auto_ptr< T >::release\(\)](#).

Here is the call graph for this function:



11.5.4.2 template<typename T> T& cxx::Auto_ptr< T >::operator* () const throw () [inline]

Dereference the pointer.

Definition at line 80 of file [auto_ptr](#).

11.5.4.3 template<typename T> T* cxx::Auto_ptr< T >::operator-> () const throw () [inline]

Member access for the object.

Definition at line 83 of file [auto_ptr](#).

11.5.4.4 template<typename T> T* cxx::Auto_ptr< T >::get () const throw () [inline]

Get the normal pointer.

Attention

This function will not release the object, the object will be deleted by the smart pointer.

Definition at line 90 of file [auto_ptr](#).

11.5.4.5 template<typename T> T* cxx::Auto_ptr< T >::release () throw () [inline]

Release the object and get the normal pointer back.

After calling this function the smart pointer will point to NULL and the object will not be deleted by the pointer anymore.

Definition at line 98 of file [auto_ptr](#).

Referenced by [cxx::Auto_ptr< T >::operator=\(\)](#).

Here is the caller graph for this function:



11.5.4.6 template<typename T> cxx::Auto_ptr< T >::operator Priv_type * () const throw () [inline]

Operator for `if (!ptr) ...< >`.

Definition at line 110 of file [auto_ptr](#).

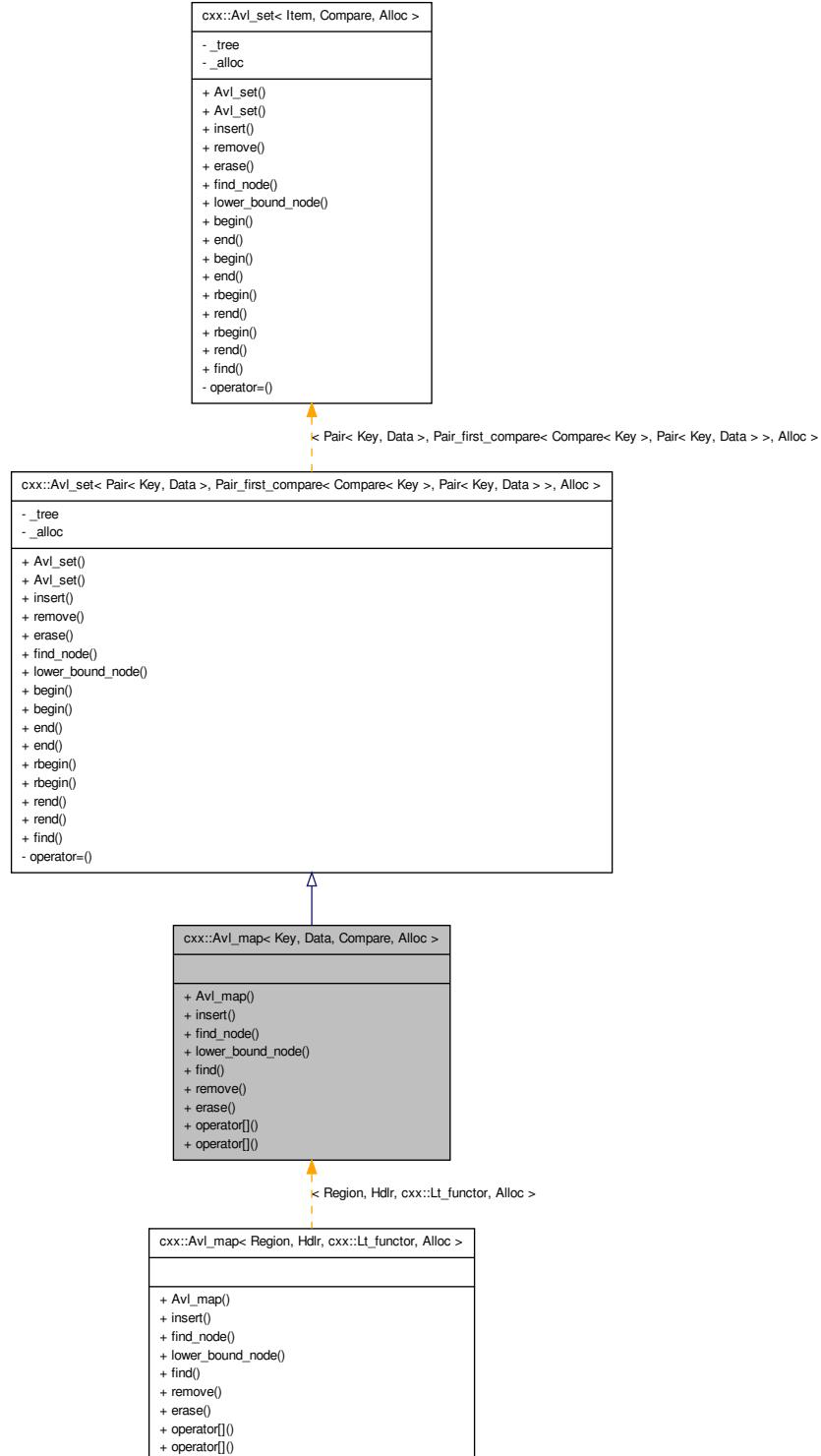
The documentation for this class was generated from the following file:

- l4/cxx/auto_ptr

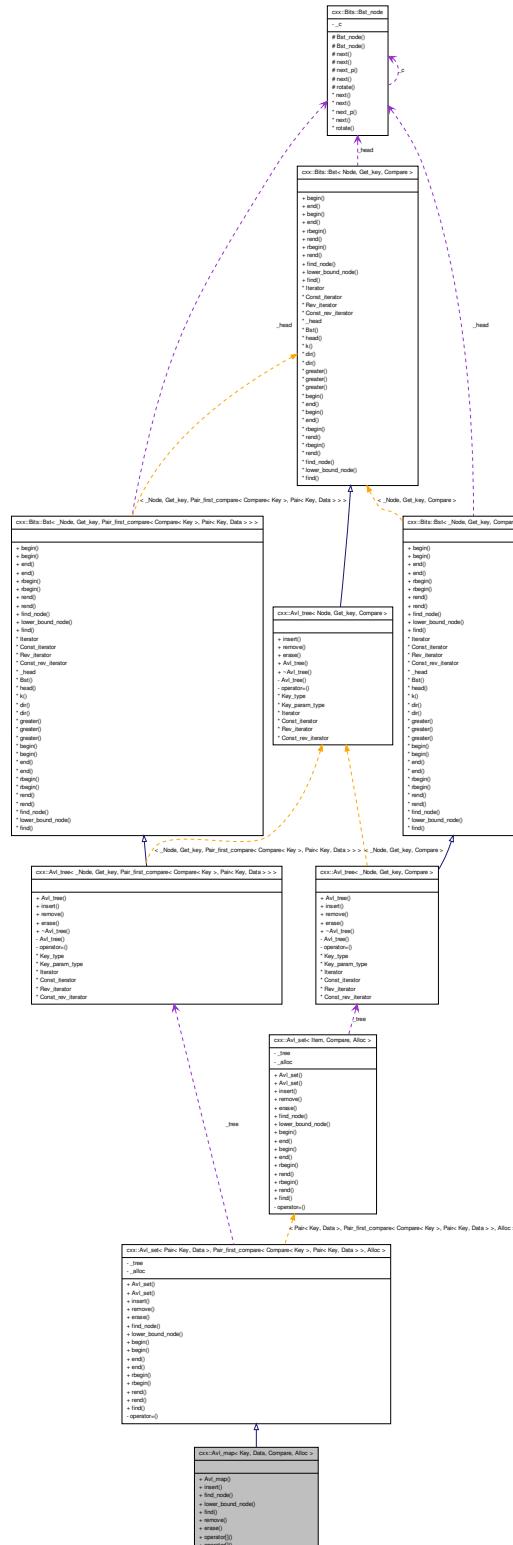
11.6 cxx::Avl_map< Key, Data, Compare, Alloc > Class Template Reference

AVL tree based associative container.

Inheritance diagram for `cxx::Avl_map< Key, Data, Compare, Alloc >`:



Collaboration diagram for cxx::Avl_map< Key, Data, Compare, Alloc >:



Public Types

- **typedef Compare< Key > Key_compare**
Type of the comparison functor.
- **typedef Key Key_type**
Type of the key values.
- **typedef Data 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.
- **typedef Base_type::Iterator Iterator**
Forward iterator for the set.
- **typedef Base_type::Const_iterator Const_iterator**
Constant forward iterator for the set.
- **typedef Base_type::Rev_iterator Rev_iterator**
Backward iterator for the set.
- **typedef Base_type::Const_rev_iterator 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.
- **Node find_node (Key_type const &key) const**
Find a <key, data> pair for a given key.
- **Node lower_bound_node (Key_type const &key) const**
Find the first node greater or equal to key.
- **Iterator find (Key_type const &key) const**
Find a <key, data> pair for a given key.
- **int remove (Key_type const &key)**
Remove the <key, data> pair for the given key.

- `int erase (Key_type const &key)`
Removed the element key.
- `Data_type const & operator[] (Key_type const &key) const`
Get the data for the given key.
- `Data_type & operator[] (Key_type const &key)`
Get the data for the given key.

11.6.1 Detailed Description

`template<typename Key, typename Data, template< typename A > class Compare = Lt_functor, template< typename B > class Alloc = New_allocator> class cxx::Avl_map< Key, Data, Compare, Alloc >`

AVL tree based associative container.

Parameters

- Key* Type of the key values.
Data Type of the data values.
Compare Type comparison functor for the key values.
Alloc Type of the allocator used for the nodes.

Definition at line 43 of file [avl_map](#).

11.6.2 Constructor & Destructor Documentation

11.6.2.1 `template<typename Key, typename Data, template< typename A > class Compare = Lt_functor, template< typename B > class Alloc = New_allocator> cxx::Avl_map< Key, Data, Compare, Alloc >::Avl_map (Node_allocator const & alloc = Node_allocator ()) [inline]`

Create an empty AVL-tree based map.

Parameters

- comp* The comparison functor.
alloc The node allocator.

Definition at line 62 of file [avl_map](#).

11.6.3 Member Function Documentation

11.6.3.1 `template<typename Key, typename Data, template< typename A > class Compare = Lt_functor, template< typename B > class Alloc = New_allocator> cxx::Pair<Iterator, int> cxx::Avl_map< Key, Data, Compare, Alloc >::insert (Key_type const & key, Data_type const & data) [inline]`

Insert a `<key, data>` pair into the map.

Parameters

key The key value.

data The data value to insert.

Definition at line 71 of file [avl_map](#).

11.6.3.2 template<typename Key, typename Data, template< typename A > class Compare = Lt_functor, template< typename B > class Alloc = New_allocator> Node cxx::Avl_map< Key, Data, Compare, Alloc >::find_node (Key_type const & key) const [inline]

Find a <key, data> pair for a given key.

Parameters

key The key value to use for the lookup.

Definition at line 78 of file [avl_map](#).

11.6.3.3 template<typename Key, typename Data, template< typename A > class Compare = Lt_functor, template< typename B > class Alloc = New_allocator> Node cxx::Avl_map< Key, Data, Compare, Alloc >::lower_bound_node (Key_type const & key) const [inline]

Find the first node greater or equal to *key*.

Parameters

key the key to look for.

Returns

The first node greater or equal to *key*.

Definition at line 86 of file [avl_map](#).

11.6.3.4 template<typename Key, typename Data, template< typename A > class Compare = Lt_functor, template< typename B > class Alloc = New_allocator> Iterator cxx::Avl_map< Key, Data, Compare, Alloc >::find (Key_type const & key) const [inline]

Find a <key, data> pair for a given key.

Parameters

key The key value to use for the lookup.

Definition at line 93 of file [avl_map](#).

11.6.3.5 `template<typename Key, typename Data, template< typename A > class Compare = Lt_functor, template< typename B > class Alloc = New_allocator> int cxx::Avl_map< Key, Data, Compare, Alloc >::remove (Key_type const & key) [inline]`

Remove the `<key, data>` pair for the given key.

Parameters

key The key value of the pair that shall be removed.

Definition at line 100 of file [avl_map](#).

11.6.3.6 `template<typename Key, typename Data, template< typename A > class Compare = Lt_functor, template< typename B > class Alloc = New_allocator> int cxx::Avl_map< Key, Data, Compare, Alloc >::erase (Key_type const & key) [inline]`

Removed the element *key*.

See also

[remove\(\)](#)

Definition at line 107 of file [avl_map](#).

11.6.3.7 `template<typename Key, typename Data, template< typename A > class Compare = Lt_functor, template< typename B > class Alloc = New_allocator> Data_type const& cxx::Avl_map< Key, Data, Compare, Alloc >::operator[] (Key_type const & key) const [inline]`

Get the data for the given key.

Parameters

key The key value to use for lookup.

Precondition

A `<key, data>` pair for the given key value must exist.

Definition at line 115 of file [avl_map](#).

11.6.3.8 `template<typename Key, typename Data, template< typename A > class Compare = Lt_functor, template< typename B > class Alloc = New_allocator> Data_type& cxx::Avl_map< Key, Data, Compare, Alloc >::operator[] (Key_type const & key) [inline]`

Get the data for the given key.

Parameters

key The key value to use for lookup.

Precondition

A `<key, data>` pair for the given key value must exist.

Definition at line 123 of file [avl_map](#).

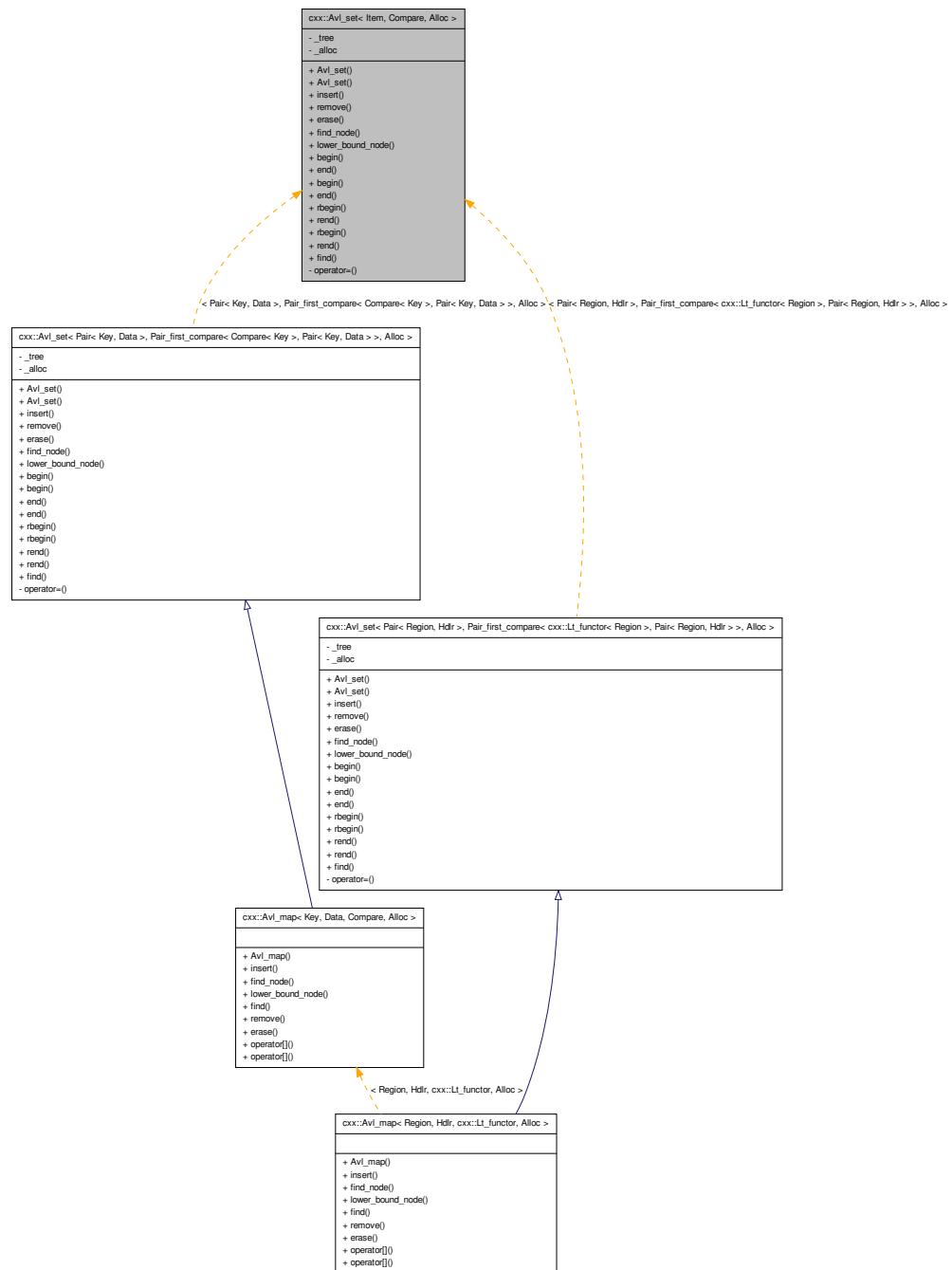
The documentation for this class was generated from the following file:

- [l4/cxx/avl_map](#)

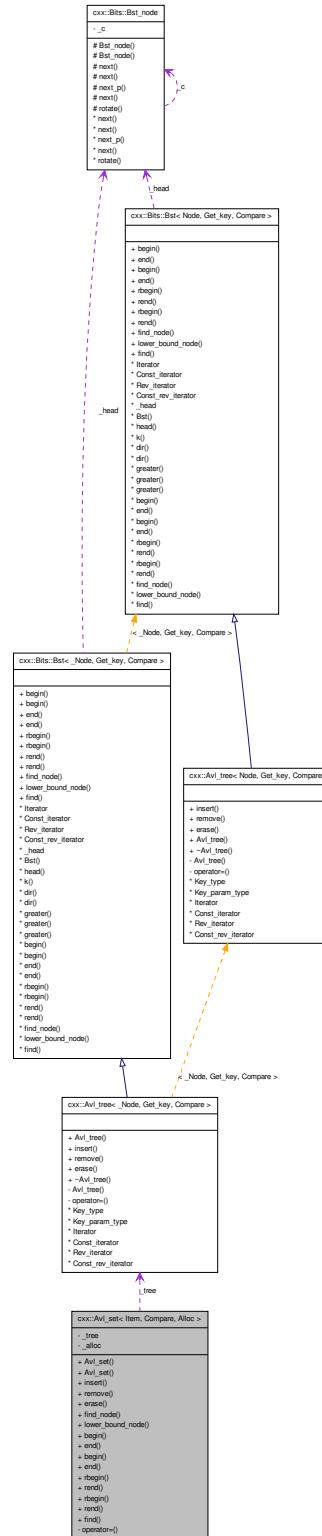
11.7 `cxx::Avl_set< Item, Compare, Alloc >` Class Template Reference

AVL Tree for simple comapreable items.

Inheritance diagram for cxx::Avl_set< Item, Compare, Alloc >:



Collaboration diagram for cxx::Avl_set< Item, Compare, Alloc >:



Data Structures

- class `Node`
A smart pointer to a tree item.

Public Types

- `typedef Item Item_type`
Type for the items contained in the tree.
- `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

- `Avl_set (Node_allocator const &alloc=Node_allocator())`
Create a AVL-tree based set.
- `Avl_set (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 (Item_type const &item)`
Remove an item from the set.
- `Node find_node (Item_type const &item) const`
Lookup a node equal to item.
- `Node lower_bound_node (Item_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.

11.7.1 Detailed Description

```
template<typename Item, class Compare = Lt_functor<Item>, template< typename A > class Alloc = New_allocator> class cxx::Avl_set< Item, Compare, Alloc >
```

AVL Tree for simple comparable items. The AVL tree can store any kind of items where a partial order is defined. The default relation is defined by the '<' operator.

Parameters

Item The type of the items to be stored in the tree.

Compare The relation to define the partial order, default is to use operator '<'.

Alloc The allocator to use for the nodes of the AVL tree.

Definition at line [106](#) of file [avl_set](#).

11.7.2 Constructor & Destructor Documentation

```
11.7.2.1 template<typename Item, class Compare = Lt_functor<Item>, template< typename A > class Alloc = New_allocator> cxx::Avl_set< Item, Compare, Alloc >::Avl_set ( Node_allocator const & alloc = Node_allocator () ) [inline, explicit]
```

Create a AVL-tree based set.

Parameters*comp* Comparison functor.*alloc* Node allocator.

Create an empty set (AVL-tree based).

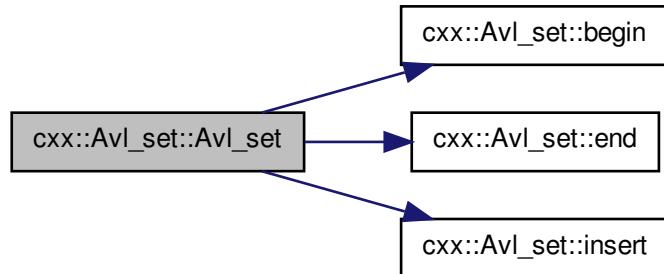
Definition at line 215 of file `avl_set`.

```
11.7.2.2 template<typename Item , class Compare , template< typename A > class Alloc>
cxx::Avl_set< Item, Compare, Alloc >::Avl_set ( Avl_set< Item, Compare, Alloc > const
& o ) [inline]
```

Create a copy of an AVL-tree based set.

Parameters*o* The set to copy.Definition at line 335 of file `avl_set`.References `cxx::Avl_set< Item, Compare, Alloc >::begin()`, `cxx::Avl_set< Item, Compare, Alloc >::end()`, and `cxx::Avl_set< Item, Compare, Alloc >::insert()`.

Here is the call graph for this function:

**11.7.3 Member Function Documentation**

```
11.7.3.1 template<typename Item, class Compare = Lt_functor<Item>, template< typename A > class Alloc = New_allocator> cxx::Pair<Iterator, int> cxx::Avl_set< Item, Compare,
Alloc >::insert ( Item_type const & item )
```

Insert an item into the set.

Parameters*item* The item to insert.

Returns

0 on success, -1 on out of memory, and -2 if the element already exists in the set.

Insert a new item into the set, each item can only be once in the set.

Referenced by [cxx::Avl_set< Item, Compare, Alloc >::Avl_set\(\)](#).

Here is the caller graph for this function:



11.7.3.2 template<typename Item, class Compare = Lt_functor<Item>, template< typename A > class Alloc = New_allocator> int cxx::Avl_set< Item, Compare, Alloc >::remove (Item_type const & item) [inline]

Remove an item from the set.

Parameters

item The item to remove.

Returns

0 on success, -3 if the item does not exist, and -4 on internal error.

Definition at line [242](#) of file [avl_set](#).

11.7.3.3 template<typename Item, class Compare = Lt_functor<Item>, template< typename A > class Alloc = New_allocator> Node cxx::Avl_set< Item, Compare, Alloc >::find_node (Item_type const & item) const [inline]

Lookup a node equal to *item*.

Parameters

item The value to search for.

Returns

A smart pointer to the element found, if no element found the pointer is NULL.

Definition at line [267](#) of file [avl_set](#).

11.7.3.4 template<typename Item, class Compare = Lt_functor<Item>, template< typename A > class Alloc = New_allocator> Node cxx::Avl_set< Item, Compare, Alloc >::lower_bound_node (Item_type const & key) const [inline]

Find the first node greater or equal to *key*.

Parameters

key the key to look for.

Returns

The first node greater or equal to *key*.

Definition at line 275 of file [avl_set](#).

11.7.3.5 template<typename Item, class Compare = Lt_functor<Item>, template< typename A > class Alloc = New_allocator> Const_iterator cxx::Avl_set< Item, Compare, Alloc >::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 283 of file [avl_set](#).

Referenced by [cxx::Avl_set< Item, Compare, Alloc >::Avl_set\(\)](#).

Here is the caller graph for this function:



11.7.3.6 template<typename Item, class Compare = Lt_functor<Item>, template< typename A > class Alloc = New_allocator> Const_iterator cxx::Avl_set< Item, Compare, Alloc >::end () const [inline]

Get the end marker for the constant forward iterator.

Returns

The end marker for the constant forward iterator.

Definition at line 288 of file [avl_set](#).

Referenced by [cxx::Avl_set< Item, Compare, Alloc >::Avl_set\(\)](#).

Here is the caller graph for this function:



11.7.3.7 template<typename Item, class Compare = Lt_functor<Item>, template< typename A > class Alloc = New_allocator> Iterator cxx::Avl_set< Item, Compare, Alloc >::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 294 of file [avl_set](#).

11.7.3.8 template<typename Item, class Compare = Lt_functor<Item>, template< typename A > class Alloc = New_allocator> Iterator cxx::Avl_set< Item, Compare, Alloc >::end () [inline]

Get the end marker for the mutable forward iterator.

Returns

The end marker for mutable forward iterator.

Definition at line 299 of file [avl_set](#).

11.7.3.9 template<typename Item, class Compare = Lt_functor<Item>, template< typename A > class Alloc = New_allocator> Const_rev_iterator cxx::Avl_set< Item, Compare, Alloc >::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 305 of file [avl_set](#).

```
11.7.3.10 template<typename Item, class Compare = Lt_functor<Item>, template< typename A
> class Alloc = New_allocator> Const_rev_iterator cxx::Avl_set< Item, Compare, Alloc
>::rend ( ) const [inline]
```

Get the end marker for the constant backward iterator.

Returns

The end marker for the constant backward iterator.

Definition at line 310 of file [avl_set](#).

```
11.7.3.11 template<typename Item, class Compare = Lt_functor<Item>, template< typename A
> class Alloc = New_allocator> Rev_iterator cxx::Avl_set< Item, Compare, Alloc
>::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 316 of file [avl_set](#).

```
11.7.3.12 template<typename Item, class Compare = Lt_functor<Item>, template< typename A
> class Alloc = New_allocator> Rev_iterator cxx::Avl_set< Item, Compare, Alloc
>::rend ( ) [inline]
```

Get the end marker for the mutable backward iterator.

Returns

The end marker for mutable backward iterator.

Definition at line 321 of file [avl_set](#).

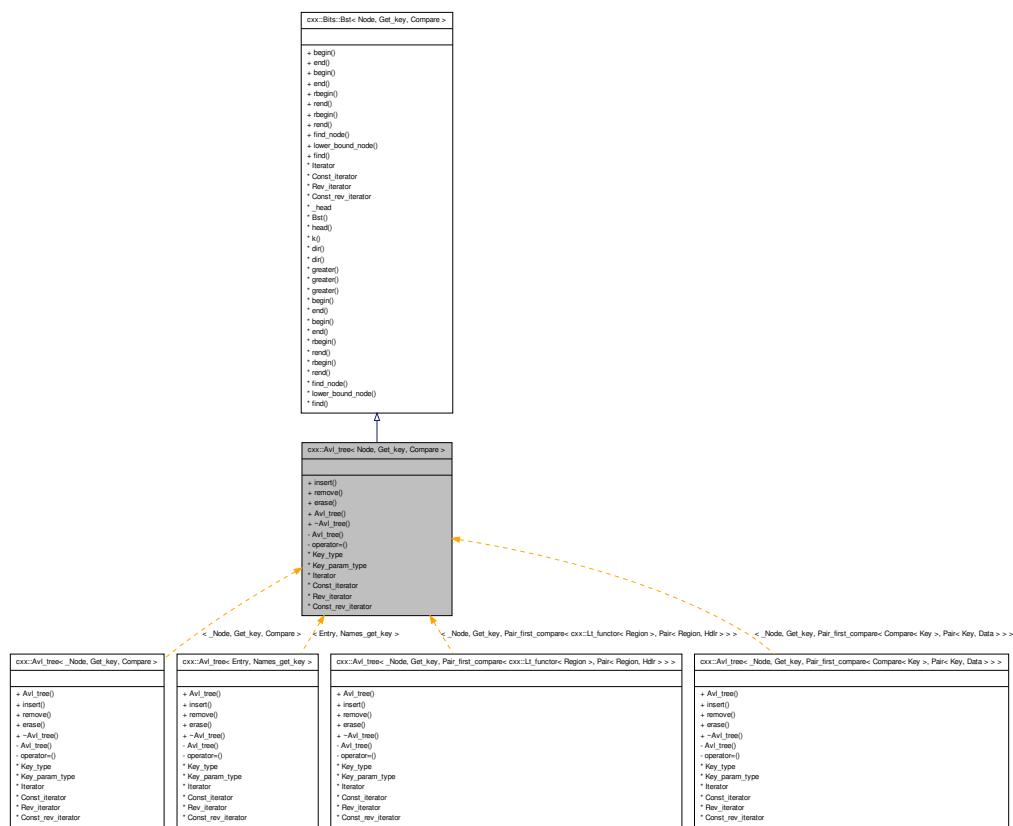
The documentation for this class was generated from the following file:

- 14/cxx/avl_set

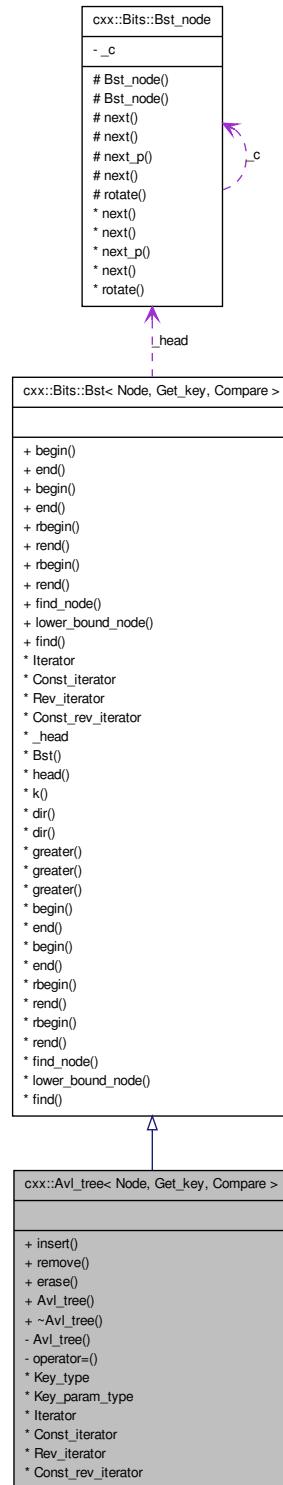
11.8 `cxx::Avl_tree< Node, Get_key, Compare >` Class Template Reference

A generic 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::Key_type Key_type**
The type of key values used to generate the total order of the elements.
- **typedef Bst::Key_param_type Key_param_type**
The type for key parameters.
- **typedef Bst::Iterator Iterator**
Grab iterator types from Bst.
- **typedef Bst::Const_iterator Const_iterator**
Constant forward iterator for the set.
- **typedef Bst::Rev_iterator Rev_iterator**
Backward iterator for the set.
- **typedef Bst::Const_rev_iterator Const_rev_iterator**
Constant backward iterator for the set.

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 ()**
Create an empty AVL tree.
- **~Avl_tree ()**
Destroy, and free the set.

11.8.1 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.

Parameters

Node the data type of the nodes (must inherit from `Avl_tree_node`).

`Get_key` the meta function to get the key value from a node. The implementation uses `Get_key::key_of(ptr_to_node)`.

`Compare` binary relation to establish a total order for the nodes of the tree. `Compare()(l, r)` must return true if the key *l* is smaller than the key *r*.

Examples:

[tmpfs/lib/src/fs.cc](#).

Definition at line 102 of file [avl_tree](#).

11.8.2 Member Typedef Documentation

11.8.2.1 `template<typename Node, typename Get_key, typename Compare = Lt_functor<typename Get_key::Key_type>> typedef Bst::Iterator cxx::Avl_tree< Node, Get_key, Compare >::Iterator`

Grab iterator types from `Bst`.

Forward iterator for the set.

Reimplemented from [cxx::Bits::Bst< Node, Get_key, Compare >](#).

Definition at line 133 of file [avl_tree](#).

11.8.3 Member Function Documentation

11.8.3.1 `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

`new_node` a pointer to the new node. This node must not already be in an AVL tree.

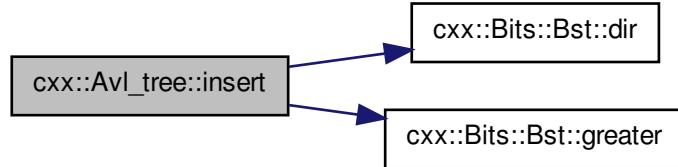
Returns

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 that first point to this node and second is 'false'.

Definition at line 225 of file [avl_tree](#).

References [cxx::Bits::Bst< Node, Get_key, Compare >::_head](#), [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:



11.8.3.2 `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

key The node to remove.

Returns

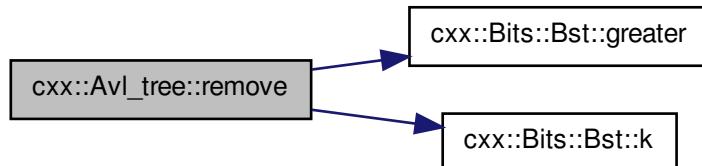
The pointer to the removed node on success, or NULL -3 if no node with the *key* exists.

Definition at line 287 of file [avl_tree](#).

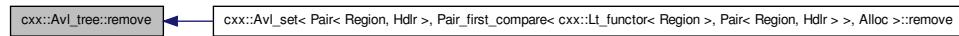
References `cxx::Bits::Bst< Node, Get_key, Compare >::_head`, `cxx::Bits::Bst< Node, Get_key, Compare >::greater()`, `cxx::Bits::Bst< Node, Get_key, Compare >::k()`, `cxx::Bits::Direction::L`, and `cxx::Bits::Direction::N`.

Referenced by `cxx::Avl_set< Pair< Region, Hdlr >, Pair_first_compare< cxx::Lt_functor< Region >, Pair< Region, Hdlr > >, Alloc >::remove()`.

Here is the call graph for this function:



Here is the caller graph for this function:



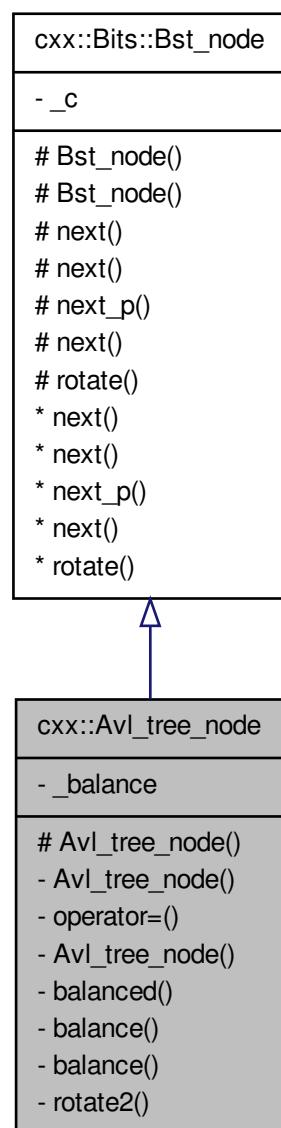
The documentation for this class was generated from the following file:

- l4/cxx/avl_tree

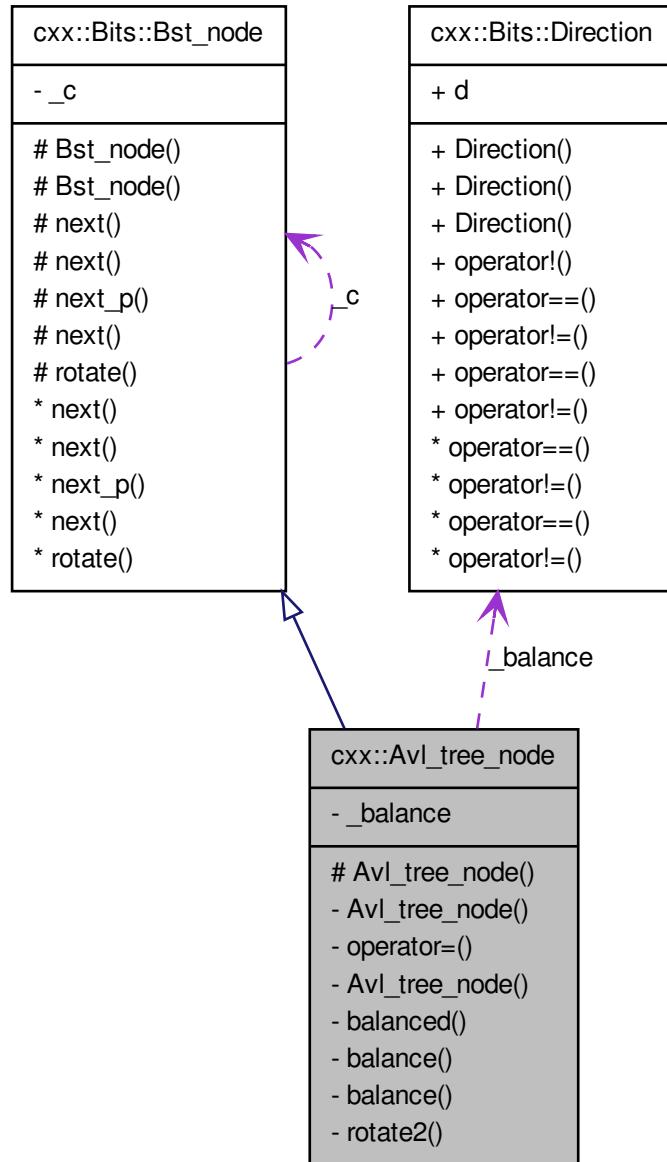
11.9 cxx::Avl_tree_node Class Reference

Node of an AVL tree.

Inheritance diagram for cxx::Avl_tree_node:



Collaboration diagram for cxx::Avl_tree_node:



Protected Member Functions

- [Avl_tree_node \(\)](#)

Create an uninitialized node, this is what you shoulkd do.

11.9.1 Detailed Description

Node of an AVL tree.

Definition at line 38 of file [avl_tree](#).

The documentation for this class was generated from the following file:

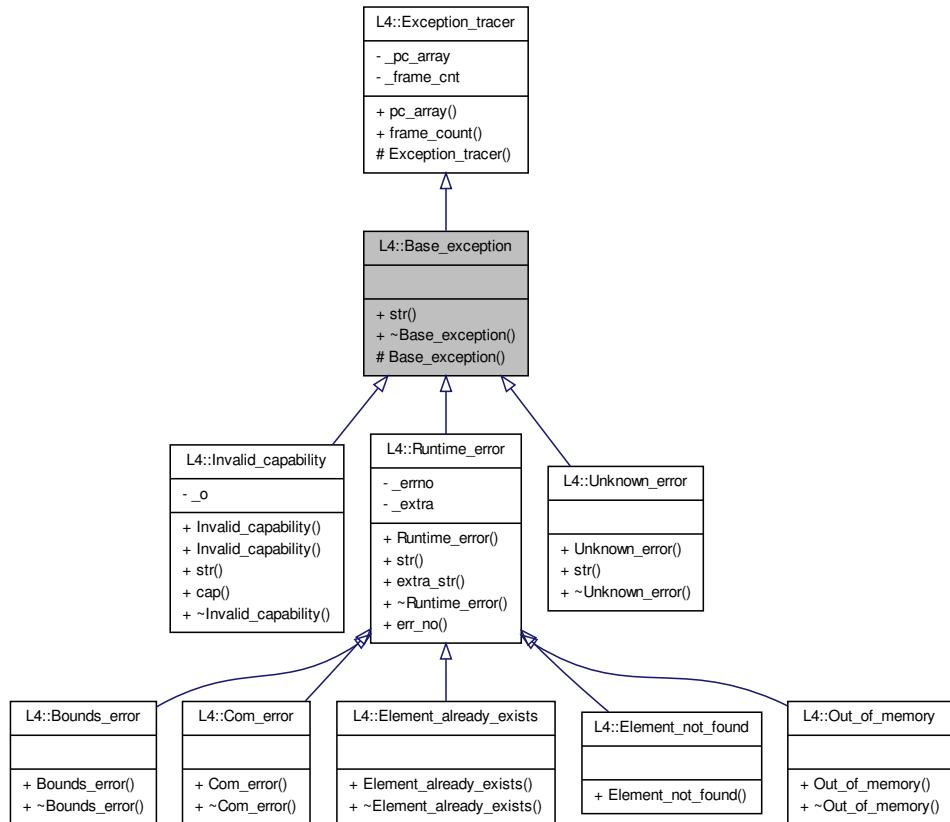
- [l4/cxx/avl_tree](#)

11.10 L4::Base_exception Class Reference

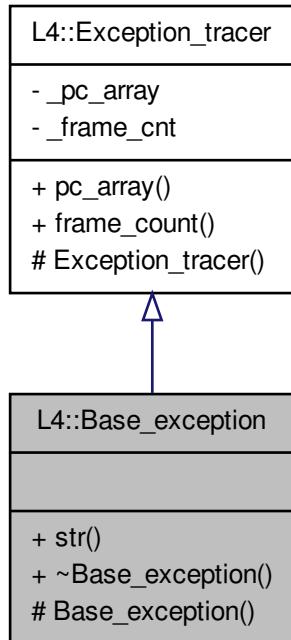
Base class for all exceptions, thrown by the [L4Re](#) framework.

```
#include <l4/cxx/exceptions>
```

Inheritance diagram for L4::Base_exception:



Collaboration diagram for L4::Base_exception:



Public Member Functions

- virtual char const * **str** () const =0 throw ()

Should return a human readable string for the exception.
- virtual **~Base_exception** () throw ()

Destruction.

Protected Member Functions

- **Base_exception** () throw ()

Create a base exception.

11.10.1 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 113 of file [exceptions](#).

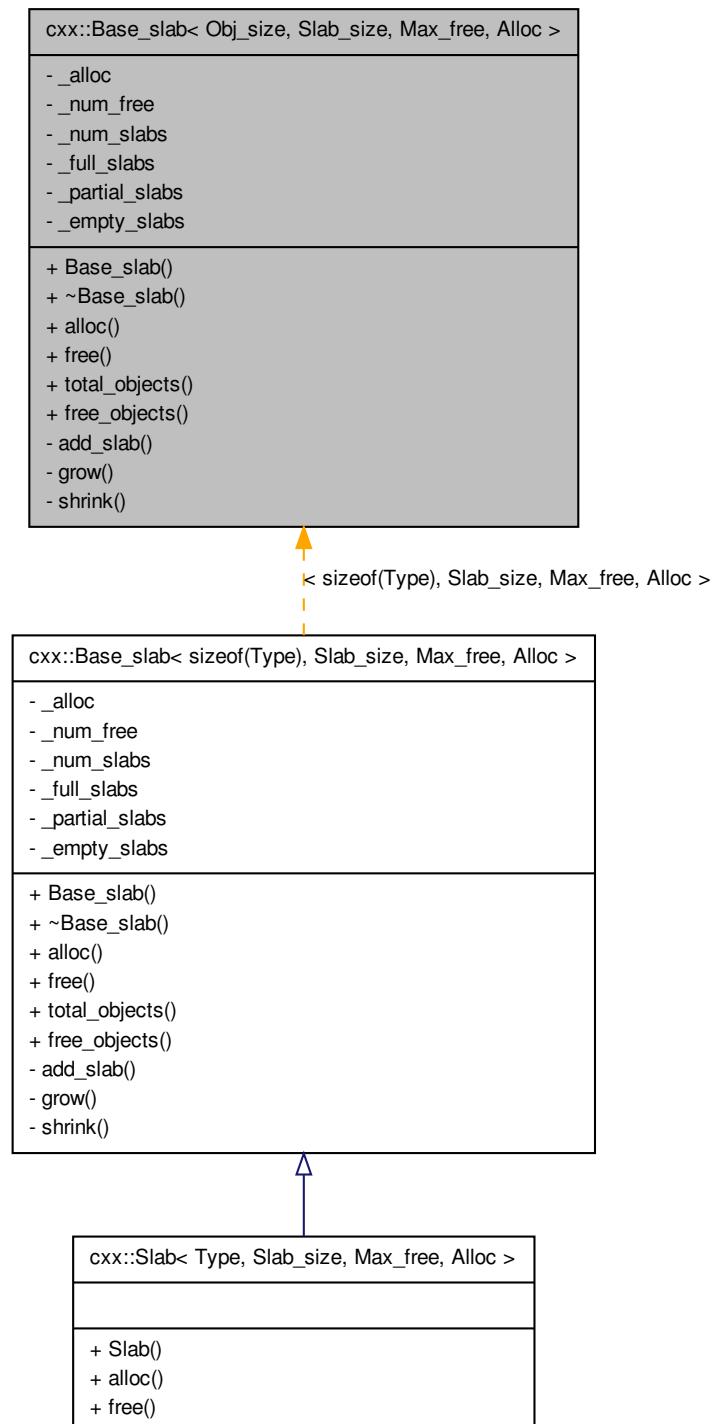
The documentation for this class was generated from the following file:

- [l4/cxx/exceptions](#)

11.11 `cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >` Class Template Reference

Basic slab allocator.

Inheritance diagram for cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >:



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 allocator for the slab caches.

Public Member Functions

- unsigned **total_objects** () const throw ()

Get the total number of objects managed by the slab allocator.
- unsigned **free_objects** () const throw ()

Get the total number of objects managed by the slab allocator.

11.11.1 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.

Parameters

Obj_size The size of the objects managed by the allocator (in bytes).

Slab_size The size of a slab cache (in bytes).

Max_free The maximum number of free slab caches. When this limit is reached slab caches are freed.

Alloc The allocator that is used to allocate the slab caches.

Definition at line 40 of file [slab_alloc](#).

11.11.2 Member Enumeration Documentation

11.11.2.1 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 cache.

objects_per_slab objects per slab cache.

max_free_slabs maximum number of free slab caches.

Definition at line 63 of file [slab_alloc](#).

11.11.3 Member Function Documentation

11.11.3.1 `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 throw () [inline]`

Get the total number of objects managed by the slab allocator.

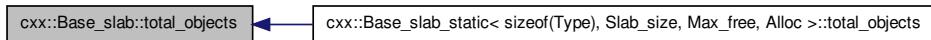
Returns

The number of objects managed by the allocator (including the free objects).

Definition at line 263 of file `slab_alloc`.

Referenced by `cxx::Base_slab_static< sizeof(Type), Slab_size, Max_free, Alloc >::total_objects()`.

Here is the caller graph for this function:



11.11.3.2 `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 throw () [inline]`

Get the total number of objects managed by the slab allocator.

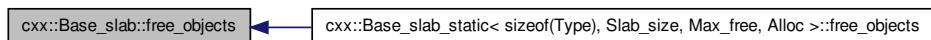
Returns

The number of objects managed by the allocator (including the free objects).

Definition at line 271 of file `slab_alloc`.

Referenced by `cxx::Base_slab_static< sizeof(Type), Slab_size, Max_free, Alloc >::free_objects()`.

Here is the caller graph for this function:



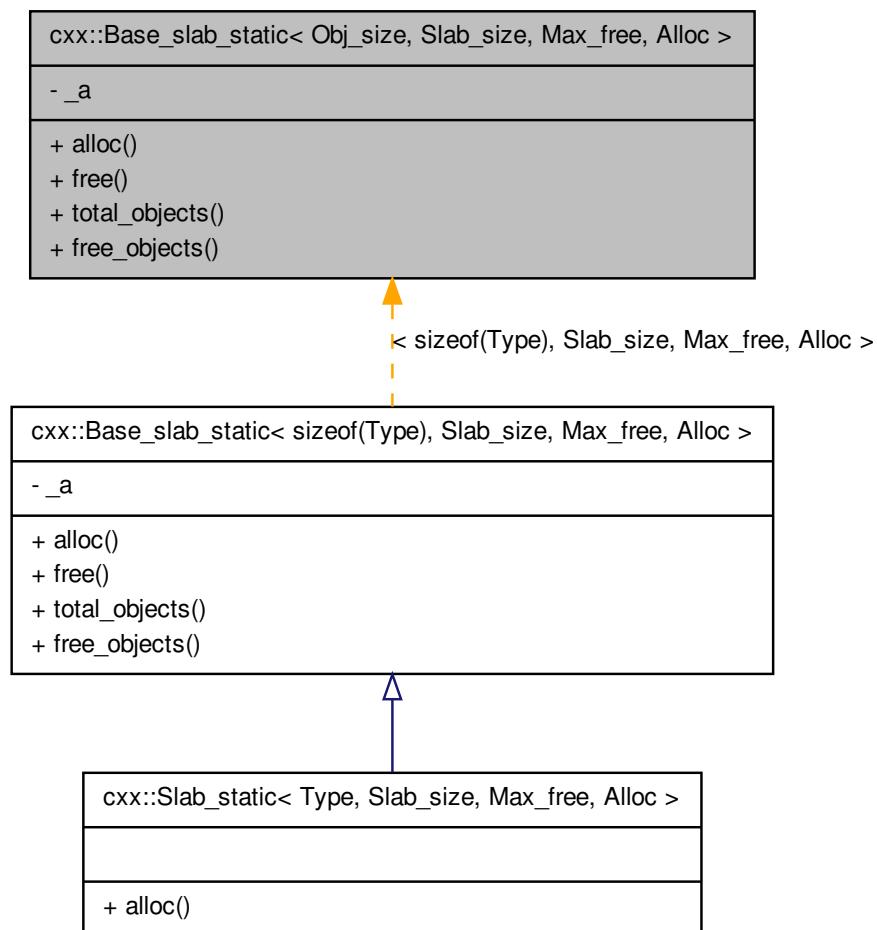
The documentation for this class was generated from the following file:

- `l4/cxx/slab_alloc`

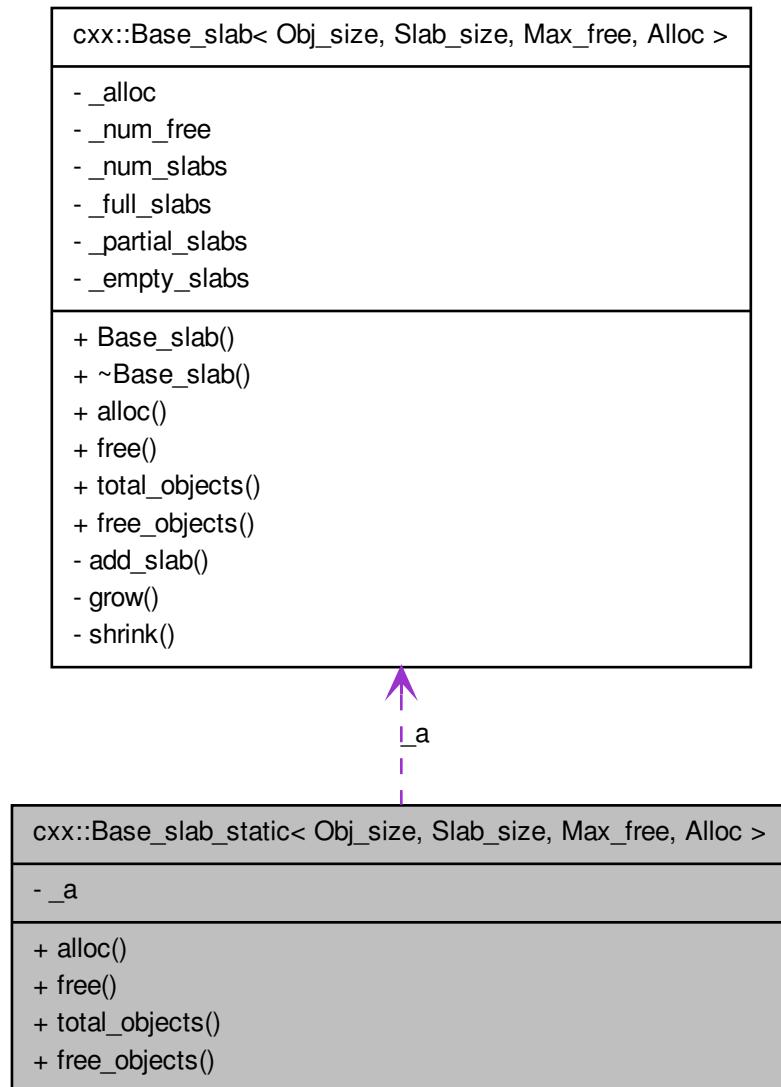
11.12 `cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >` Class Template Reference

Merged slab allocator (allocators for objects of the same size are merged together).

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 () throw ()`

Allocate an object.

- `void free (void *p) throw ()`

Free the given object (p).

- `unsigned total_objects () const throw ()`

Get the total number of objects managed by the slab allocator.

- `unsigned free_objects () const throw ()`

Get the number of free objects in the slab allocator.

11.12.1 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).

Parameters

Obj_size The size of an object managed by the slab allocator.

Slab_size The size of a slab cache.

Max_free The maximum number of free slab caches.

Alloc The allocator for the slab caches.

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 348 of file [slab_alloc](#).

11.12.2 Member Enumeration Documentation

11.12.2.1 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 cache.

objects_per_slab number of objects per slab cache.

max_free_slabs maximum number of free slab caches.

Definition at line 355 of file [slab_alloc](#).

11.12.3 Member Function Documentation

11.12.3.1 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 () throw () [inline]

Allocate an object.

Reimplemented in [cxx::Slab_static< Type, Slab_size, Max_free, Alloc >](#).

Definition at line 365 of file [slab_alloc](#).

11.12.3.2 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) throw () [inline]

Free the given object (*p*).

Parameters

p The pointer to the object to free.

Precondition

p must be a pointer to an object allocated by this allocator.

Definition at line 371 of file [slab_alloc](#).

11.12.3.3 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 throw () [inline]

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 380 of file [slab_alloc](#).

11.12.3.4 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 throw () [inline]

Get the number of free objects in the slab allocator.

Returns

The number of free objects in all free and partially used slab caches managed by this allocator.

Note

The value is the merged value for all equal parameterized [Base_slab_static](#) instances.

Definition at line 389 of file [slab_alloc](#).

The documentation for this class was generated from the following file:

- 14/cxx/slab_alloc

11.13 L4::Basic_registry Class Reference

This registry returns the corresponding server object based on the label of an [Ipc_gate](#).

Inherited by L4Re::Util::Object_registry.

Public Types

- **typedef Server_object Value**
Get the server object for a [Ipc_gate](#) label.

Static Public Member Functions

- **static int dispatch (l4_umword_t label, L4::Ipc::Iostream &ios)**
The dispatch function called by the server loop.

11.13.1 Detailed Description

This registry returns the corresponding server object based on the label of an [Ipc_gate](#).

Definition at line 310 of file [ipc_server](#).

11.13.2 Member Typedef Documentation

11.13.2.1 **typedef Server_object L4::Basic_registry::Value**

Get the server object for a [Ipc_gate](#) label.

Parameters

label The label usually stored in an [Ipc_gate](#).

Returns

A pointer to the [Server_object](#) identified the given label.

Definition at line 318 of file [ipc_server](#).

11.13.3 Member Function Documentation

11.13.3.1 static int L4::Basic_registry::dispatch (l4_umword_t *label*, L4::Ipc::Iostream & *ios*) [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

label The label used to find the object including the rights bits of the invoked capability.

ios The [Ipc::Iostream](#) for the request and the reply.

Returns

The return code from the object's dispatch function or -L4_ENOENT if the object does not exist.

Definition at line 334 of file [ipc_server](#).

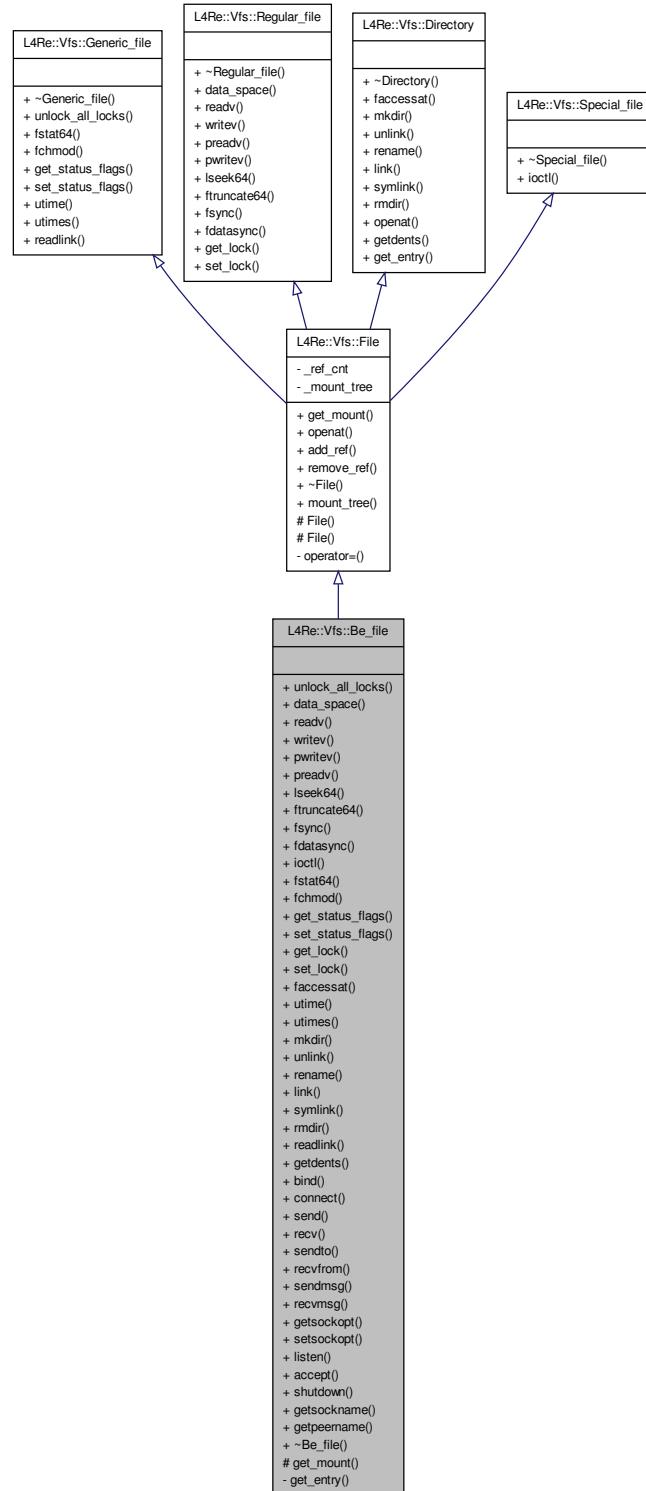
The documentation for this class was generated from the following file:

- l4/cxx/ipc_server

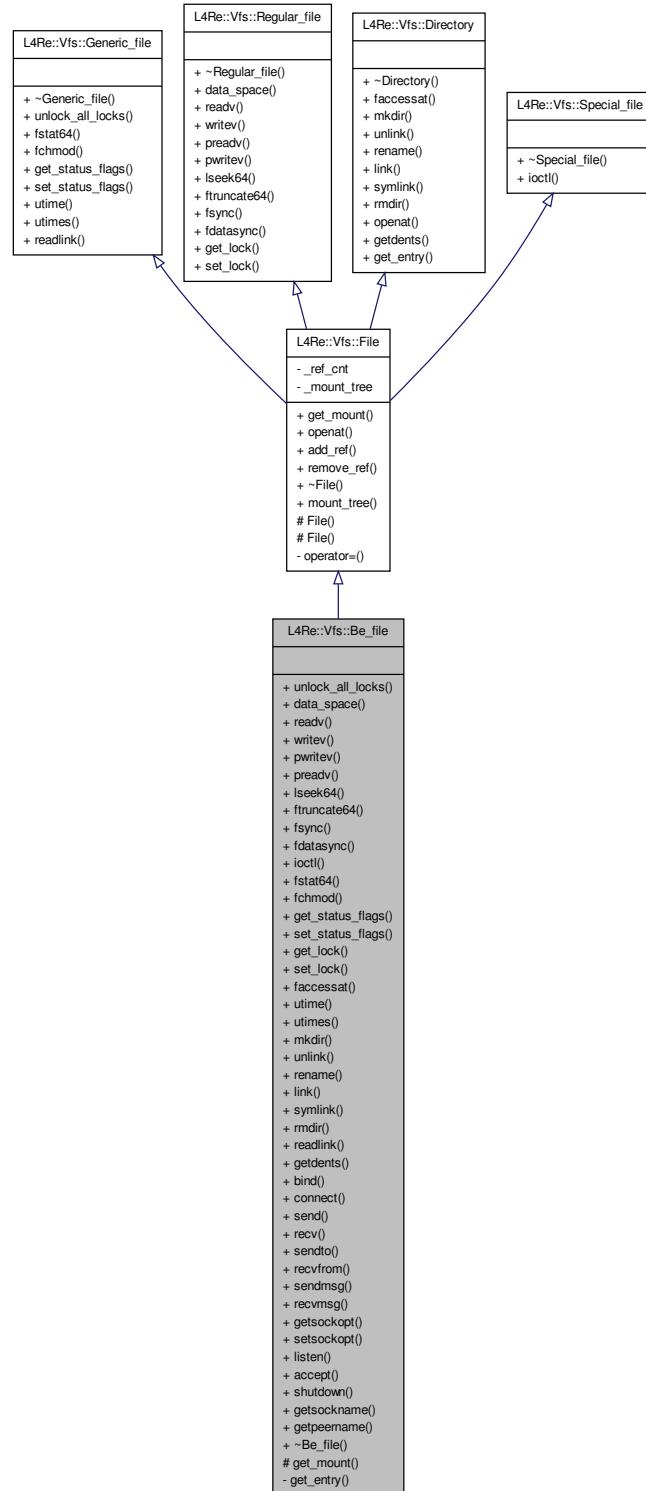
11.14 L4Re::Vfs::Be_file Class Reference

Boiler plate class for implementing an open file for [L4Re::Vfs](#).

Inheritance diagram for L4Re::Vfs::Be_file:



Collaboration diagram for L4Re::Vfs::Be_file:



Public Member Functions

- `ssize_t readv (const struct iovec *, int) throw ()`
Default backend for POSIX read and readv functions.
- `ssize_t writev (const struct iovec *, int) throw ()`
Default backend for POSIX write and writev functions.
- `ssize_t pwritev (const struct iovec *, int, off64_t) throw ()`
Default backend for POSIX pwrite and pwritev functions.
- `ssize_t preadv (const struct iovec *, int, off64_t) throw ()`
Default backend for POSIX pread and preadv functions.
- `off64_t lseek64 (off64_t, int) throw ()`
Default backend for POSIX seek and lseek functions.
- `int ftruncate64 (off64_t) throw ()`
Default backend for the POSIX truncate, ftruncate and similar functions.
- `int fsync () const throw ()`
Default backend for POSIX fsync.
- `int fdatasync () const throw ()`
Default backend for POSIX fdatasync.
- `int ioctl (unsigned long, va_list) throw ()`
Default backend for POSIX ioctl.
- `int fchmod (mode_t) throw ()`
Default backend for POSIX chmod and fchmod.
- `int get_status_flags () const throw ()`
Default backend for POSIX fcntl subfunctions.
- `int set_status_flags (long) throw ()`
Default backend for POSIX fcntl subfunctions.
- `int get_lock (struct flock64 *) throw ()`
Default backend for POSIX fcntl subfunctions.
- `int set_lock (struct flock64 *, bool) throw ()`
Default backend for POSIX fcntl subfunctions.
- `int faccessat (const char *, int, int) throw ()`
Default backend for POSIX access and faccessat functions.
- `int utime (const struct utimbuf *) throw ()`
Default backend for POSIX utime.

- int [utimes](#) (const struct timeval[2]) throw ()
Default backend for POSIX utimes.
- int [mkdir](#) (const char *, mode_t) throw ()
Default backend for POSIX mkdir and mkdirat.
- int [unlink](#) (const char *) throw ()
Default backend for POSIX unlink, unlinkat.
- int [rename](#) (const char *, const char *) throw ()
Default backend for POSIX rename, renameat.
- int [link](#) (const char *, const char *) throw ()
Default backend for POSIX link, linkat.
- int [symlink](#) (const char *, const char *) throw ()
Default backend for POSIX symlink, symlinkat.
- int [rmdir](#) (const char *) throw ()
Default backend for POSIX rmdir, rmdirat.
- ssize_t [readlink](#) (char *, size_t)
Default backend for POSIX readlink, readlinkat.

11.14.1 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 38 of file [backend](#).

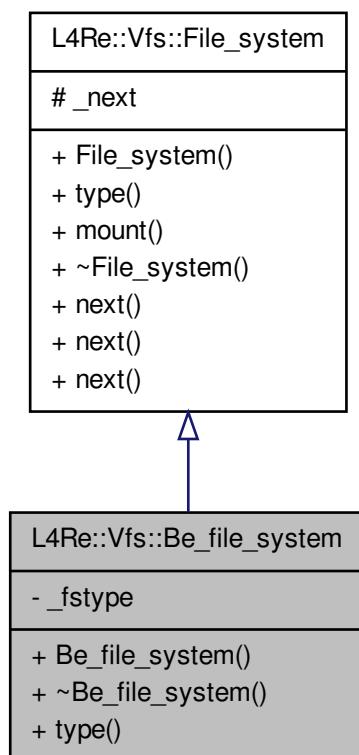
The documentation for this class was generated from the following file:

- l4/l4re_vfs/backend

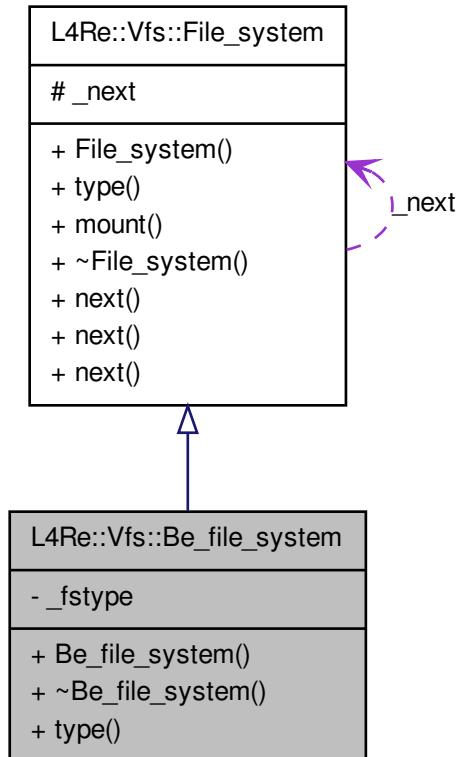
11.15 L4Re::Vfs::Be_file_system Class Reference

Boilerplate class for implementing a [L4Re::Vfs::File_system](#).

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) throw ()
Create a file-system object for the given fstype.
- **~Be_file_system** () throw ()
Destroy a file-system object.
- char const * **type** () const throw ()
Return the file-system type.

11.15.1 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 [289](#) of file [backend](#).

11.15.2 Constructor & Destructor Documentation

11.15.2.1 [L4Re::Vfs::Be_file_system::Be_file_system \(char const * *fstype* \) throw \(\) \[inline, explicit\]](#)

Create a file-system object for the given *fstype*.

Parameters

fstype The type that [type\(\)](#) shall return.

This constructor takes care of registering the file system in the registry of [L4Re::Vfs::vfs_ops](#).

Examples:

[tmpfs/lib/src/fs.cc](#).

Definition at line [303](#) of file [backend](#).

11.15.2.2 [L4Re::Vfs::Be_file_system::~Be_file_system \(\) throw \(\) \[inline\]](#)

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 [315](#) of file [backend](#).

11.15.3 Member Function Documentation

11.15.3.1 [char const* L4Re::Vfs::Be_file_system::type \(\) const throw \(\) \[inline, virtual\]](#)

Return the file-system type.

Returns the file-system type given as *fstype* in the constructor.

Implements [L4Re::Vfs::File_system](#).

Definition at line [325](#) of file [backend](#).

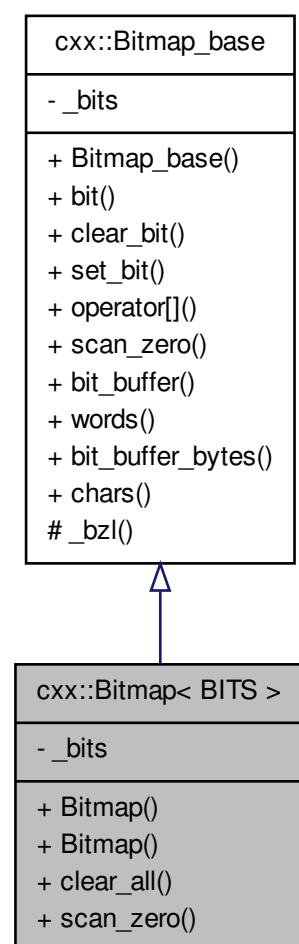
The documentation for this class was generated from the following file:

- [l4/l4re_vfs/backend](#)

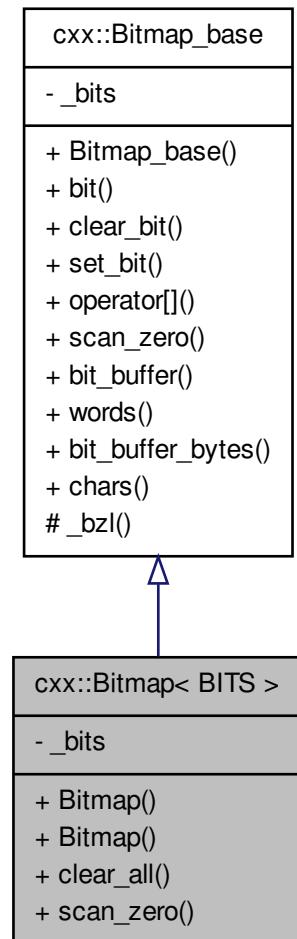
11.16 [cxx::Bitmap< BITS > Class Template Reference](#)

A static bit map.

Inheritance diagram for cxx::Bitmap< BITS >:



Collaboration diagram for cxx::Bitmap< BITS >:



Public Member Functions

- `Bitmap()` throw ()

Create a bitmap with BITS bits.

- `void clear_all()`

Scan for the first zero bit.

11.16.1 Detailed Description

`template<int BITS> class cxx::Bitmap< BITS >`

A static bit map.

Parameters

BITS the number of bits that shall be in the bitmap.

Definition at line 120 of file [bitmap](#).

11.16.2 Constructor & Destructor Documentation

11.16.2.1 `template<int BITS> cxx::Bitmap< BITS >::Bitmap() throw() [inline]`

Create a bitmap with *BITS* bits.

Definition at line 127 of file [bitmap](#).

11.16.3 Member Function Documentation

11.16.3.1 `template<int BITS> void cxx::Bitmap< BITS >::clear_all() [inline]`

Scan for the first zero bit.

Parameters

start_bit the bit where the scanning shall begin.

Compared to `Bitmap_base::scan_zero()`, the upper bound is set to *BITS*.

Definition at line 137 of file [bitmap](#).

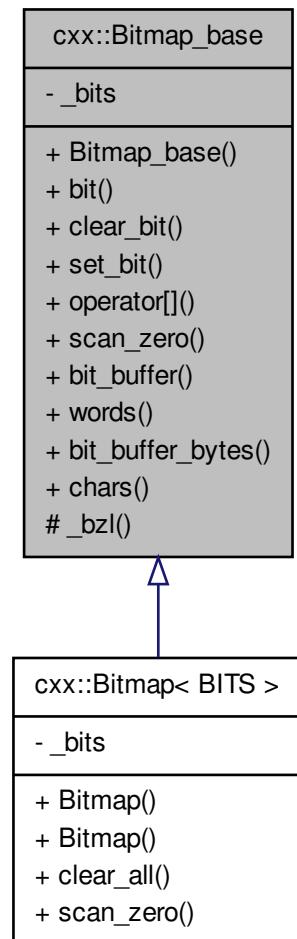
The documentation for this class was generated from the following file:

- 14/cxx(bitmap)

11.17 cxx::Bitmap_base Class Reference

Basic bitmap abstraction.

Inheritance diagram for cxx::Bitmap_base:



Data Structures

- class [Char](#)
Helper abstraction for a byte contained in the bitmap.
- class [Word](#)
Helper abstraction for a word contained in the bitmap.

Public Member Functions

- void [bit](#) (long bit, bool on) throw ()

*Set the value of bit *bit* to on.*

- void [clear_bit](#) (long *bit*) throw ()
*Clear bit *bit*.*
- void [set_bit](#) (long *bit*) throw ()
*Set bit *bit*.*
- unsigned long [operator\[\]](#) (long *bit*) const throw ()
Get the truth value of a bit.
- long [scan_zero](#) (long *max_bit*, long *start_bit*=0) const throw ()
Scans for the first zero bit.

Static Public Member Functions

- static long [words](#) (long *bits*) throw ()
Get the number of Words that are used for the bitmap.
- static long [chars](#) (long *bits*) throw ()
Get the number of chars that are used for the bitmap.

11.17.1 Detailed Description

Basic bitmap abstraction. This abstraction keeps a pointer to a memory area that is used as bitmap.
Definition at line 30 of file [bitmap](#).

11.17.2 Member Function Documentation

11.17.2.1 static long cxx::Bitmap_base::words (long *bits*) throw () [inline, static]

Get the number of *Words* that are used for the bitmap.
Definition at line 44 of file [bitmap](#).

11.17.2.2 static long cxx::Bitmap_base::chars (long *bits*) throw () [inline, static]

Get the number of chars that are used for the bitmap.
Definition at line 61 of file [bitmap](#).

11.17.2.3 void cxx::Bitmap_base::bit (long *bit*, bool *on*) throw () [inline]

Set the value of bit *bit* to *on*.

Parameters

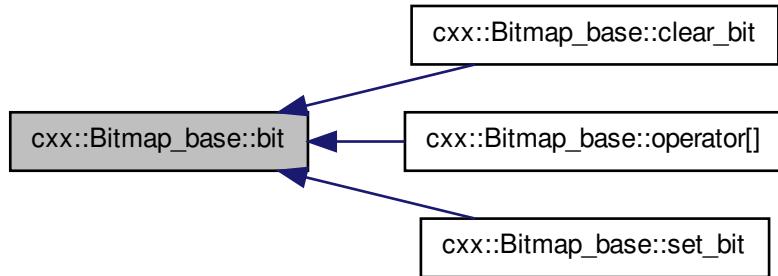
bit the number of the bit

on the boolean value that shall be assigned to the bit.

Definition at line 146 of file [bitmap](#).

Referenced by [clear_bit\(\)](#), [operator\[\]\(\)](#), and [set_bit\(\)](#).

Here is the caller graph for this function:



11.17.2.4 void cxx::Bitmap_base::clear_bit (long *bit*) throw () [inline]

Clear bit *bit*.

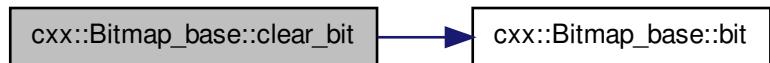
Parameters

bit the number of the bit to clear.

Definition at line 155 of file [bitmap](#).

References [bit\(\)](#).

Here is the call graph for this function:



11.17.2.5 void cxx::Bitmap_base::set_bit (long *bit*) throw () [inline]

Set bit *bit*.

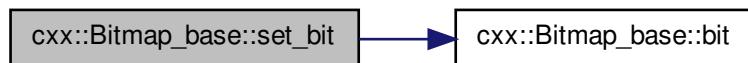
Parameters

bit the number of the bit to set,

Definition at line 164 of file [bitmap](#).

References [bit\(\)](#).

Here is the call graph for this function:



11.17.2.6 `unsigned long cxx::Bitmap_base::operator[](long bit) const throw() [inline]`

Get the truth value of a bit.

Parameters

bit the number of the bit to read.

Definition at line 173 of file [bitmap](#).

References [bit\(\)](#).

Here is the call graph for this function:



11.17.2.7 `long cxx::Bitmap_base::scan_zero(long max_bit, long start_bit = 0) const throw() [inline]`

Scans for the first zero bit.

Parameters

max_bit the upper bound for the scanning operation.

start_bit the number of the first bit to look at.

Definition at line 194 of file [bitmap](#).

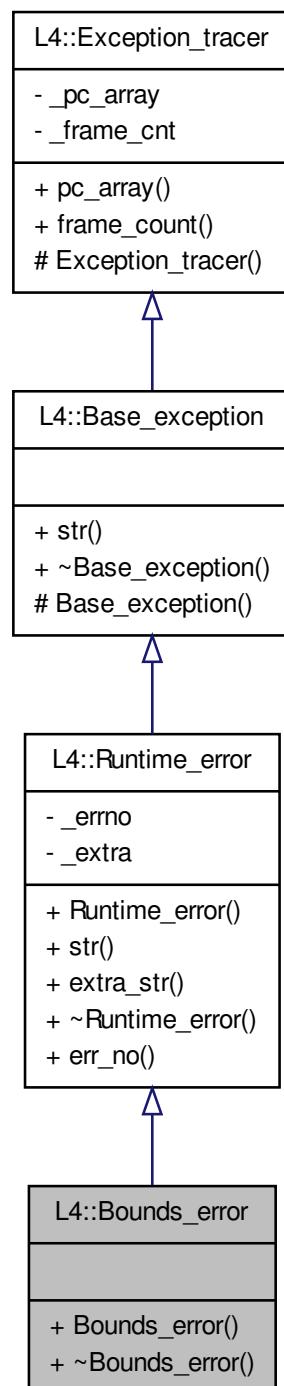
The documentation for this class was generated from the following file:

- [l4/cxx\(bitmap\)](#)

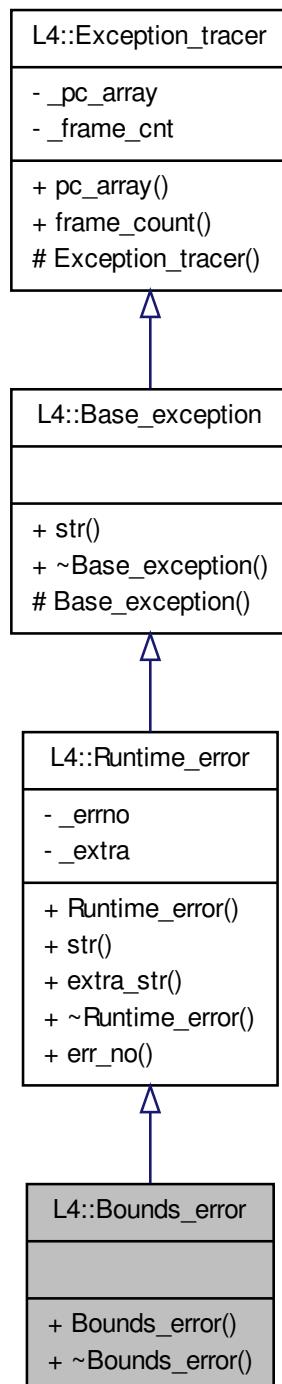
11.18 L4::Bounds_error Class Reference

Access out of bounds.

Inheritance diagram for L4::Bounds_error:



Collaboration diagram for L4::Bounds_error:



11.18.1 Detailed Description

Access out of bounds.

Definition at line 269 of file [exceptions](#).

The documentation for this class was generated from the following file:

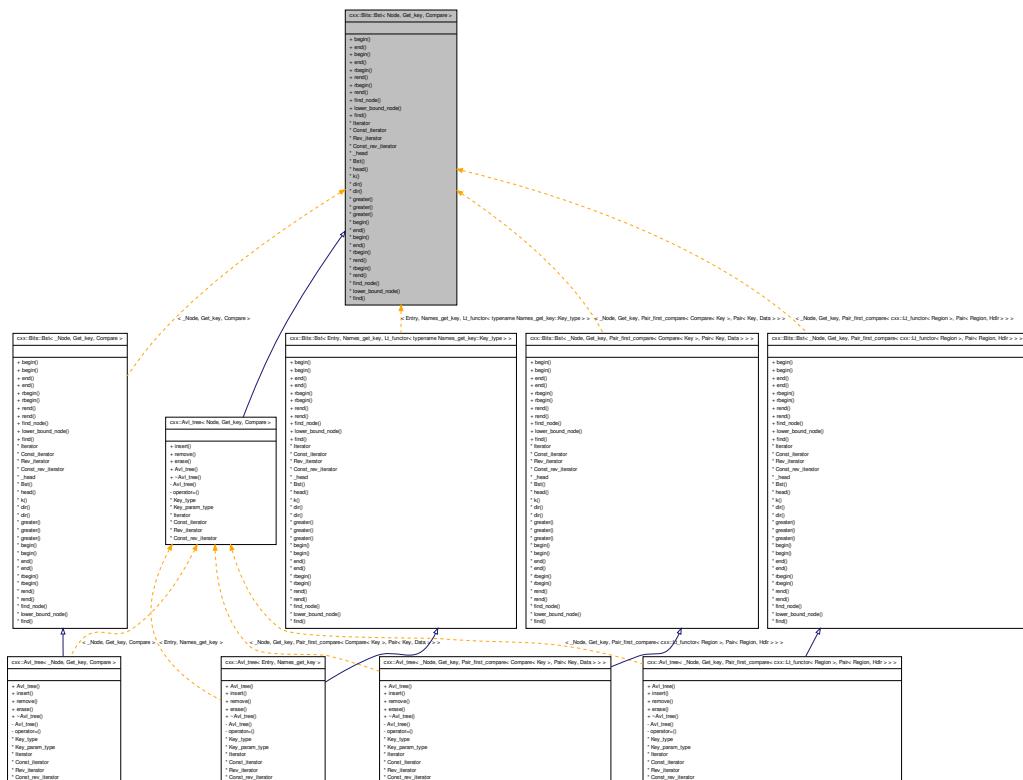
- 14/cxx/exceptions

11.19 `cxx::Bits::Bst< Node, Get_key, Compare >` Class Template Reference

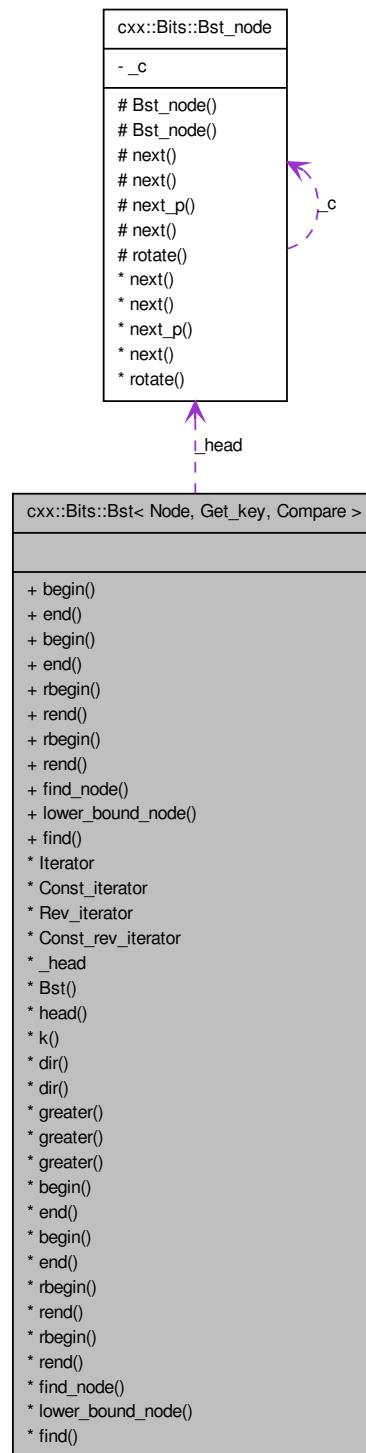
Basic binary search tree (BST).

```
#include <bst.h>
```

Inheritance diagram for cxx::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.

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.

11.19.1 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 40 of file [bst.h](#).

11.19.2 Member Function Documentation

11.19.2.1 template<typename Node, typename Get_key, typename Compare> static 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

l is the key to look for.

r is the key at the current position.

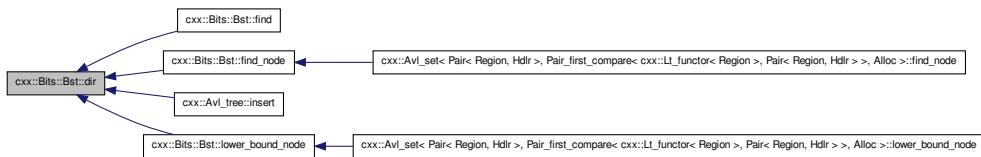
Returns

Direction::L for left, Direction::R for right, and Direction::N if *l* is equal to *r*.

Definition at line 117 of file [bst.h](#).

Referenced by [cxx::Bits::Bst< Node, Get_key, Compare >::find\(\)](#), [cxx::Bits::Bst< Node, Get_key, Compare >::find_node\(\)](#), [cxx::Avl_tree< Node, Get_key, Compare >::insert\(\)](#), and [cxx::Bits::Bst< Node, Get_key, Compare >::lower_bound_node\(\)](#).

Here is the caller graph for this function:



11.19.2.2 template<typename Node, typename Get_key, typename Compare> static 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

l is the key to look for.

r is the node at the current position.

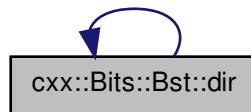
Returns

Direction::L for left, Direction::R for right, and Direction::N if *l* is equal to *r*.

Definition at line 133 of file [bst.h](#).

Referenced by [cxx::Bits::Bst< _Node, Get_key, Pair_first_compare< Compare< Key >, Pair< Key, Data > > >::dir\(\)](#).

Here is the caller graph for this function:



11.19.2.3 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.

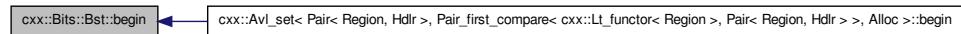
Returns

Constant forward iterator for the first element in the set.

Definition at line 159 of file [bst.h](#).

Referenced by [cxx::Avl_set< Pair< Region, Hdlr >, Pair_first_compare< cxx::Lt_functor< Region >, Pair< Region, Hdlr > >, Alloc >::begin\(\)](#).

Here is the caller graph for this function:



11.19.2.4 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.

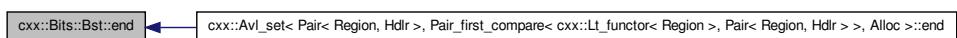
Returns

The end marker for the constant forward iterator.

Definition at line 164 of file `bst.h`.

Referenced by `cxx::Avl_set< Pair< Region, Hdlr >, Pair_first_compare< cxx::Lt_functor< Region >, Pair< Region, Hdlr > >, Alloc >::end()`.

Here is the caller graph for this function:



11.19.2.5 `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 170 of file `bst.h`.

11.19.2.6 `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 175 of file `bst.h`.

11.19.2.7 `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.

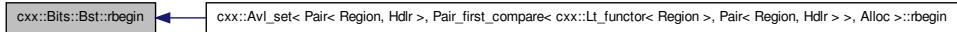
Returns

The constant backward iterator for the last element in the set.

Definition at line 181 of file `bst.h`.

Referenced by `cxx::Avl_set< Pair< Region, Hdlr >, Pair_first_compare< cxx::Lt_functor< Region >, Pair< Region, Hdlr > >, Alloc >::rbegin()`.

Here is the caller graph for this function:



**11.19.2.8 template<typename Node, typename Get_key, typename Compare> Const_rev_iterator
cxx::Bst< Node, Get_key, Compare >::rend () const [inline]**

Get the end marker for the constant backward iterator.

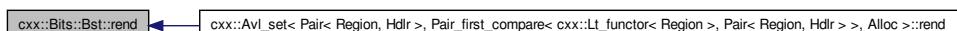
Returns

The end marker for the constant backward iterator.

Definition at line 186 of file [bst.h](#).

Referenced by [cxx::Avl_set< Pair< Region, Hdr >, Pair_first_compare< cxx::Lt_functor< Region >, Pair< Region, Hdr > >, Alloc >::rend\(\)](#).

Here is the caller graph for this function:



**11.19.2.9 template<typename Node, typename Get_key, typename Compare> Rev_iterator
cxx::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 192 of file [bst.h](#).

**11.19.2.10 template<typename Node, typename Get_key, typename Compare> Rev_iterator
cxx::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 197 of file [bst.h](#).

```
11.19.2.11 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

key The key value of the element to search.

Returns

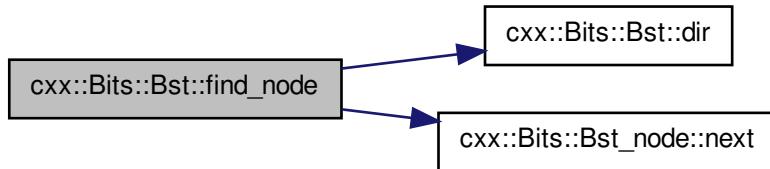
A pointer to the node with the given *key*, or NULL if *key* was not found.

Definition at line 236 of file [bst.h](#).

References [cxx::Bits::Bst< Node, Get_key, Compare >::_head](#), [cxx::Bits::Bst< Node, Get_key, Compare >::dir\(\)](#), [cxx::Bits::Direction::N](#), and [cxx::Bits::Bst_node::next\(\)](#).

Referenced by [cxx::Avl_set< Pair< Region, Hldr >, Pair_first_compare< cxx::Lt_functor< Region >, Pair< Region, Hldr > >, Alloc >::find_node\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```
11.19.2.12 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

key The key value of the element to search.

Returns

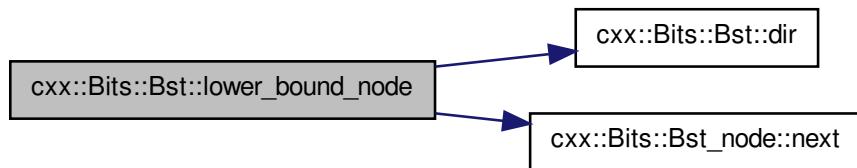
A pointer to the node with the given *key*, or NULL if *key* was not found.

Definition at line 252 of file [bst.h](#).

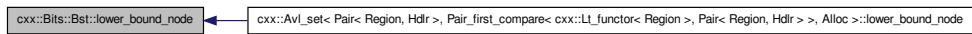
References [cxx::Bits::Bst< Node, Get_key, Compare >::_head](#), [cxx::Bits::Bst< Node, Get_key, Compare >::dir\(\)](#), [cxx::Bits::Direction::L](#), [cxx::Bits::Direction::N](#), and [cxx::Bits::Bst_node::next\(\)](#).

Referenced by [cxx::Avl_set< Pair< Region, Hdlr >, Pair_first_compare< cxx::Lt_functor< Region >, Pair< Region, Hdlr > >, Alloc >::lower_bound_node\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.19.2.13 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

key The key value of the element to search.

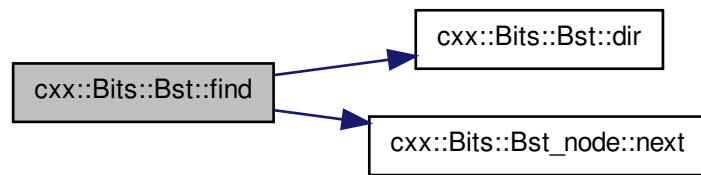
Returns

A valid iterator for the node with the given *key*, or an invalid iterator if *key* was not found.

Definition at line 272 of file [bst.h](#).

References [cxx::Bits::Bst< Node, Get_key, Compare >::_head](#), [cxx::Bits::Bst< Node, Get_key, Compare >::dir\(\)](#), [cxx::Bits::Direction::L](#), [cxx::Bits::Direction::N](#), and [cxx::Bits::Bst_node::next\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

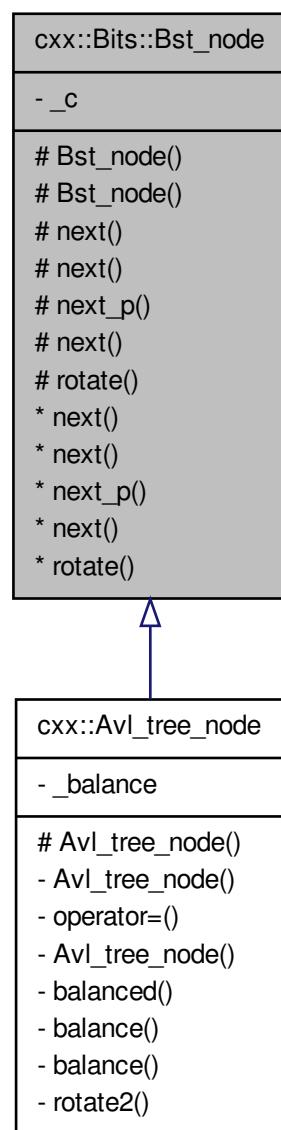
- 14/cxx/bits/bst.h

11.20 **cxx::Bits::Bst_node** Class Reference

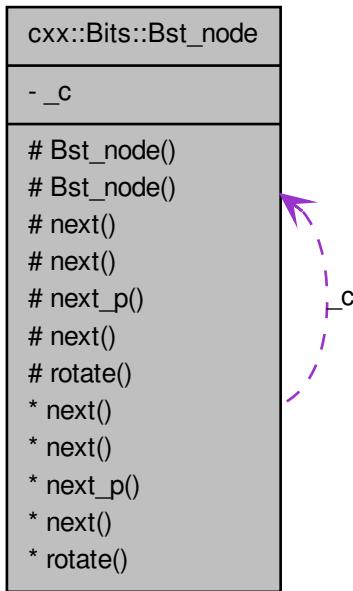
Basic type of a node in a binary search tree (BST).

```
#include <bst_base.h>
```

Inheritance diagram for cxx::Bits::Bst_node:



Collaboration diagram for cxx::Bits::Bst_node:



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.

11.20.1 Detailed Description

Basic type of a node in a binary search tree (BST).

Definition at line 77 of file [bst_base.h](#).

The documentation for this class was generated from the following file:

- l4/cxx/bits/bst_base.h

11.21 L4::Ipc::Buf_cp_in< T > Class Template Reference

Abstraction for extracting array from an [Ipc::Istream](#).

Public Member Functions

- **Buf_cp_in** (T *v, unsigned long &size)
Create a buffer for extracting an array from an [Ipc::Istream](#).

11.21.1 Detailed Description

template<typename T> class L4::Ipc::Buf_cp_in< T >

Abstraction for extracting array from an [Ipc::Istream](#). An instance of **Buf_cp_in** can be used to extract an array from an [Ipc::Istream](#). This is the counterpart to the [Buf_cp_out](#) abstraction. 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_cp_in\(\)](#), [Buf_in](#), [buf_in\(\)](#), [Buf_cp_out](#), and [buf_cp_out\(\)](#).

Definition at line 117 of file [ipc_stream](#).

11.21.2 Constructor & Destructor Documentation

11.21.2.1 **template<typename T> L4::Ipc::Buf_cp_in< T >::Buf_cp_in (T * v, unsigned long & size) [inline]**

Create a buffer for extracting an array from an [Ipc::Istream](#).

Parameters

- *v* The buffer for array (copy in).
- *size* Input: the number of elements the array can take at most
Output: the number of elements found in the stream.

Definition at line 126 of file [ipc_stream](#).

The documentation for this class was generated from the following file:

- l4/cxx/ipc_stream

11.22 L4::Ipc::Buf_cp_out< T > Class Template Reference

Abstraction for inserting an array into an [Ipc::Ostream](#).

Public Member Functions

- [Buf_cp_out](#) (T *v, unsigned long size)
Create a buffer object for the given array.
- unsigned long [size](#) () const
Get the number of elements in the array.
- T * [buf](#) () const
Get the pointer to the array.

11.22.1 Detailed Description

template<typename T> class L4::Ipc::Buf_cp_out< T >

Abstraction for inserting an array into an [Ipc::Ostream](#). An object of [Buf_cp_out](#) can be used to insert an array of arbitrary values, that can be inserted into an [Ipc::Ostream](#) individually. The array is therefore copied to the message buffer, in contrast to data handled with [Msg_out_buffer](#) or [Msg_io_buffer](#).

On insertion into the [Ipc::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.

You should use [buf_cp_out\(\)](#) to create instances of [Buf_cp_out](#).

The counterpart is either [Buf_cp_in](#) ([buf_cp_in\(\)](#)) or [Buf_in](#) ([buf_in\(\)](#)).

Definition at line 61 of file [ipc_stream](#).

11.22.2 Constructor & Destructor Documentation

11.22.2.1 template<typename T> L4::Ipc::Buf_cp_out< T >::Buf_cp_out (T * v, unsigned long size) [inline]

Create a buffer object for the given array.

Parameters

- v* The pointer to the array with *size* elements of type T.
- size* the number of elements in the array.

Definition at line 69 of file [ipc_stream](#).

11.22.3 Member Function Documentation

11.22.3.1 `template<typename T> unsigned long L4::Ipc::Buf_cp_out< T >::size () const [inline]`

Get the number of elements in the array.

Note

This function is usually used by the [Ipc::Ostream](#) itself.

Definition at line 75 of file [ipc_stream](#).

Referenced by [operator<<\(\)](#).

Here is the caller graph for this function:



11.22.3.2 `template<typename T> T* L4::Ipc::Buf_cp_out< T >::buf () const [inline]`

Get the pointer to the array.

Note

This function is usually used by the [Ipc::Ostream](#) itself.

Definition at line 81 of file [ipc_stream](#).

Referenced by [operator<<\(\)](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

- 14/cxx/ ipc_stream

11.23 L4::Ipc::Buf_in< T > Class Template Reference

Abstraction to extract an array from an [Ipc::Istream](#).

Public Member Functions

- [Buf_in](#) (T *&v, unsigned long &size)
Create an [Buf_in](#) to adjust a pointer to the array and the size of the array.

11.23.1 Detailed Description

template<typename T> class L4::Ipc::Buf_in< T >

Abstraction to extract an array from an [Ipc::Istream](#). This wrapper provides a possibility to extract an array from an [Ipc::Istream](#), 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 mechanism is comparable to that of [Msg_ptr](#), however it handles arrays inserted with [Buf_cp_out](#).

See [buf_in\(\)](#), [Buf_cp_out](#), [buf_cp_out\(\)](#), [Buf_cp_in](#), and [buf_cp_in\(\)](#).

Definition at line 211 of file [ipc_stream](#).

11.23.2 Constructor & Destructor Documentation

11.23.2.1 template<typename T> L4::Ipc::Buf_in< T >::Buf_in (T *& v, unsigned long & size) [inline]

Create an [Buf_in](#) to adjust a pointer to the array and the size of the array.

Parameters

- v The pointer to adjust to the first element of the array.

size The number of elements found in the stream.

Definition at line 220 of file [ipc_stream](#).

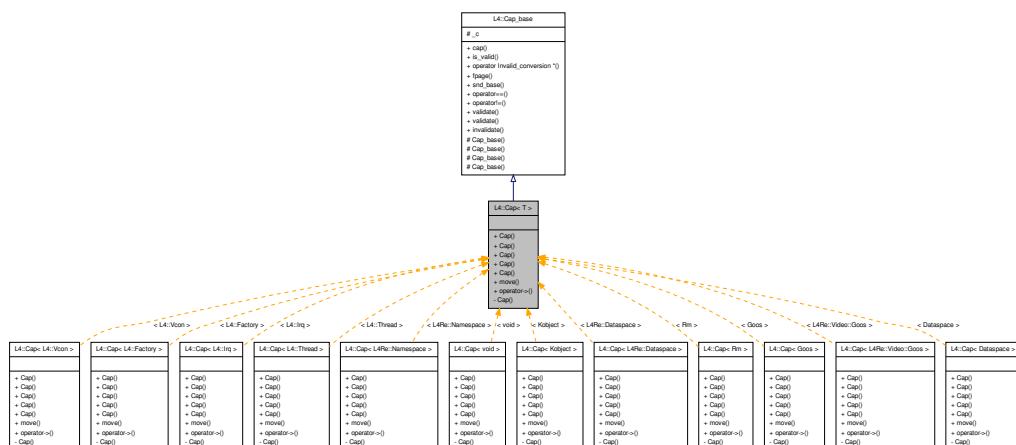
The documentation for this class was generated from the following file:

- 14/cxx/ipc_stream

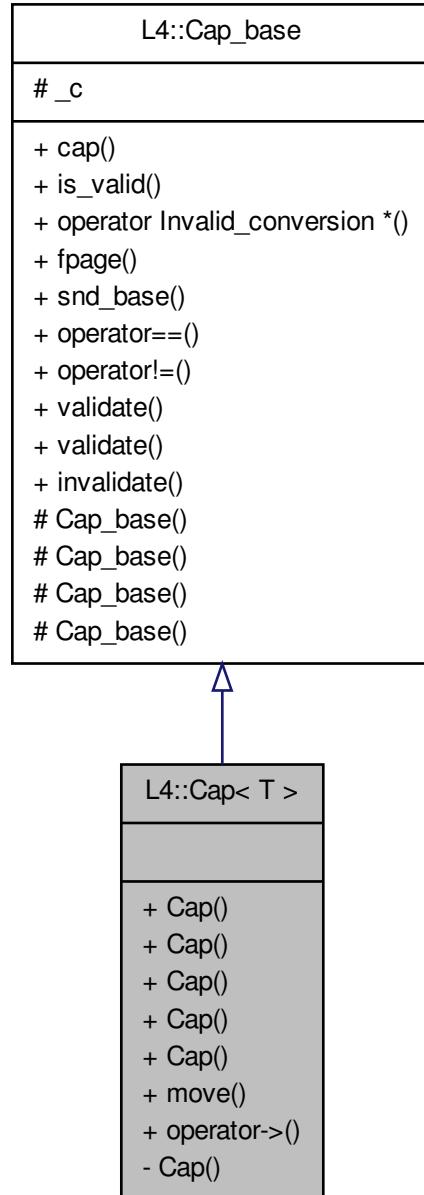
11.24 L4::Cap< T > Class Template Reference

Capability Selector a la C++.

Inheritance diagram for L4::Cap< T >:



Collaboration diagram for L4::Cap< T >:



Public Member Functions

- template<typename O >
Cap (`Cap< O > const &o`) throw ()

Create a copy from o, supporting implicit type casting.

- **Cap (Cap_type cap) throw ()**
Constructor to create an invalid capability selector.
- **Cap (l4_default_caps_t cap) throw ()**
Initialize capability with one of the default capability selectors.
- **Cap (l4_cap_idx_t idx=L4_INVALID_CAP) throw ()**
Initialize capability, defaults to the invalid capability selector.
- **Cap (No_init_type) throw ()**
Create an uninitialized cap selector.
- **Cap move (Cap const &src) const**
Move a capability to this cap slot.
- **T * operator-> () const throw ()**
Member access of a T.

Friends

- class [L4::Kobject](#)

11.24.1 Detailed Description

template<typename T> class L4::Cap< T >

Capability Selector a la C++.

Template Parameters

T the type of the object the capability points to

The C++ version of a capability looks just as a pointer, in fact it is a kind of a smart pointer for our kernel objects and the objects derived from the kernel objects ([L4::Kobject](#)).

Examples:

[examples/clntsrv/client.cc](#), [examples/libs/l4re/c++/shared_ds/ds_clnt.cc](#), and [examples/libs/l4re/streammap/client.cc](#).

Definition at line 199 of file [capability](#).

11.24.2 Constructor & Destructor Documentation

11.24.2.1 **template<typename T> template<typename O > L4::Cap< T >::Cap (Cap< O > const & o) throw () [inline]**

Create a copy from o, supporting implicit type casting.

Parameters

o is the source selector that shall be copied (and casted).

Definition at line 224 of file [capability](#).

**11.24.2.2 template<typename T> L4::Cap< T >::Cap (l4_default_caps_t *cap*) throw ()
[inline]**

Initialize capability with one of the default capability selectors.

Parameters

cap Capability selector.

Definition at line 236 of file [capability](#).

11.24.2.3 template<typename T> L4::Cap< T >::Cap (l4_cap_idx_t *idx* = L4_INVALID_CAP) throw () [inline, explicit]

Initialize capability, defaults to the invalid capability selector.

Parameters

idx Capability selector.

Definition at line 242 of file [capability](#).

11.24.3 Member Function Documentation

11.24.3.1 template<typename T> Cap L4::Cap< T >::move (Cap< T > const & *src*) const [inline]

Move a capability to this cap slot.

Parameters

src the source capability slot.

After this operation the source slot is no longer valid.

Definition at line 255 of file [capability](#).

The documentation for this class was generated from the following file:

- l4/sys/capability

11.25 L4Re::Cap_alloc Class Reference

Capability allocator interface.

Public Member Functions

- virtual `L4::Cap< void > alloc ()=0 throw ()`

Allocate a capability.

- template<typename T >
`L4::Cap< T > alloc () throw ()`

Allocate a capability.

- virtual void `free (L4::Cap< void > cap)=0 throw ()`

Free a capability.

- virtual `~Cap_alloc ()=0`

Destructor.

Static Public Member Functions

- template<typename CAP_ALLOC >
static `L4Re::Cap_alloc * get_cap_alloc (CAP_ALLOC &ca)`

Construct an instance of a capability allocator.

11.25.1 Detailed Description

Capability allocator interface.

Definition at line 39 of file `cap_alloc`.

11.25.2 Member Function Documentation

11.25.2.1 virtual `L4::Cap<void> L4Re::Cap_alloc::alloc () throw () [pure virtual]`

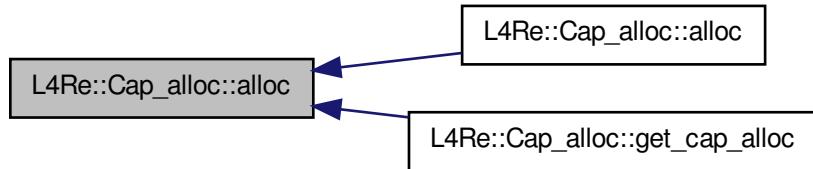
Allocate a capability.

Returns

Capability of type void

Referenced by `alloc()`, and `get_cap_alloc()`.

Here is the caller graph for this function:



11.25.2.2 template<typename T > L4::Cap<T> L4Re::Cap_alloc::alloc () throw () [inline]

Allocate a capability.

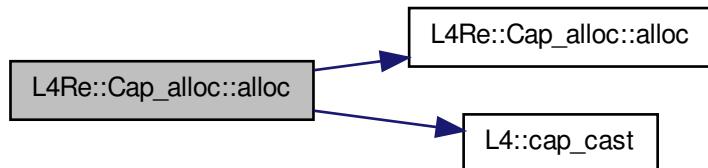
Returns

Capability of type T

Definition at line 61 of file [cap_alloc](#).

References [alloc\(\)](#), and [L4::cap_cast\(\)](#).

Here is the call graph for this function:



11.25.2.3 virtual void L4Re::Cap_alloc::free (L4::Cap< void > cap) throw () [pure virtual]

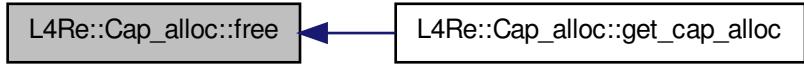
Free a capability.

Parameters

cap Capability to free.

Referenced by [get_cap_alloc\(\)](#).

Here is the caller graph for this function:



11.25.2.4 `template<typename CAP_ALLOC > static L4Re::Cap_alloc*
L4Re::Cap_alloc::get_cap_alloc(CAP_ALLOC & ca) [inline, static]`

Construct an instance of a capability allocator.

Parameters

ca Capaiblity allocator

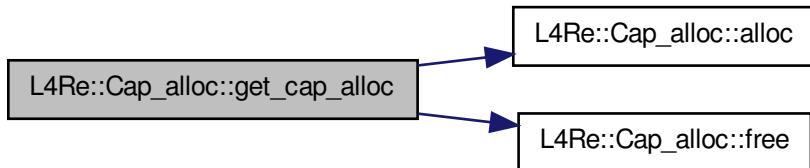
Returns

Instance of a capability allocator.

Definition at line 84 of file [cap_alloc](#).

References [alloc\(\)](#), and [free\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

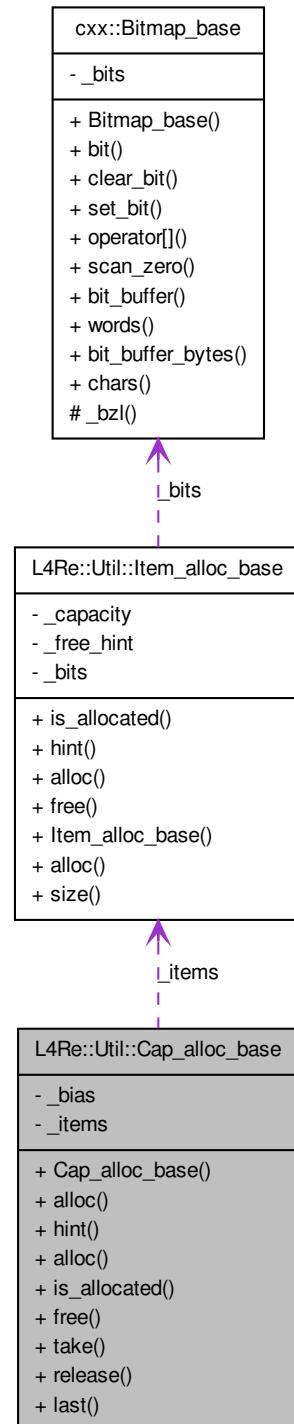
- [l4/re/cap_alloc](#)

11.26 L4Re::Util::Cap_alloc_base Class Reference

Capability allocator.

Inherited by L4Re::Util::Cap_alloc< Size >.

Collaboration diagram for L4Re::Util::Cap_alloc_base:



Public Member Functions

- template<typename T >
L4::Cap< T > alloc () throw ()
Allocate a capability slot.
 - template<typename T >
void **free (L4::Cap< T > const &cap, l4_cap_idx_t task=-1UL, l4_umword_t unmap_flags=L4_-FP_ALL_SPACES) throw ()**
Free a capability slot.

11.26.1 Detailed Description

Capability allocator.

Definition at line 35 of file [bitmap_cap_alloc.c](#).

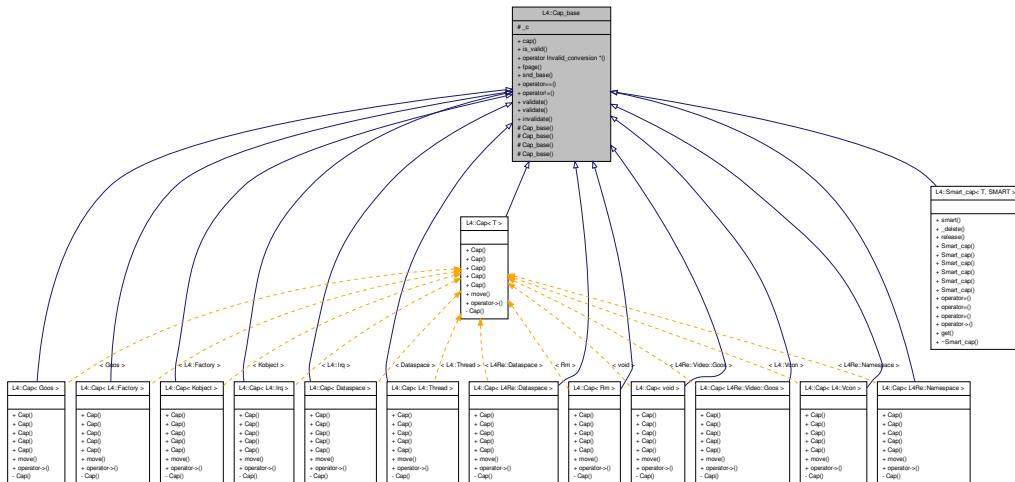
The documentation for this class was generated from the following file:

- 14/re/util(bitmap_cap_alloc)

11.27 L4::Cap_base Class Reference

Base class for all kinds of capabilities.

Inheritance diagram for L4::Cap_base:



Public Types

- enum **No_init_type** { **No_init** }
 - enum **Cap_type** { **Invalid** = L4_INVALID_CAP }

Invalid capability type.

Public Member Functions

- `l4_cap_idx_t cap () const throw ()`
Return capability selector.
- `bool is_valid () const throw ()`
Test whether capability selector is not the invalid capability selector.
- `l4_fpage_t fpage (unsigned rights=L4_FPAGE_RWX) const throw ()`
Returns flex-page of the capability selector.
- `l4_umword_t snd_base (unsigned grant=0, l4_cap_idx_t base=L4_INVALID_CAP) const throw ()`
Returns send base.
- `bool operator==(Cap_base const &o) const throw ()`
Test if two capability selectors are equal.
- `bool operator!=(Cap_base const &o) const throw ()`
Test if two capability selectors are not equal.
- `l4_mshtag_t validate (l4_utcb_t *u=l4_utcb()) const throw ()`
Check whether a capability selector points to a valid capability.
- `l4_mshtag_t validate (Cap< Task > task, l4_utcb_t *u=l4_utcb()) const throw ()`
Check whether a capability selector points to a valid capability.
- `void invalidate () throw ()`
Set this selector to the invalid capability (L4_INVALID_CAP).

Protected Member Functions

- `Cap_base (l4_cap_idx_t c) throw ()`
Generate a capability from its C representation.
- `Cap_base (Cap_type cap) throw ()`
Constructor to create an invalid capability selector.
- `Cap_base (l4_default_caps_t cap) throw ()`
Initialize capability with one of the default capability selectors.
- `Cap_base () throw ()`
Create an uninitialized instance.

Protected Attributes

- `l4_cap_idx_t _c`
The C representation of a capability selector.

11.27.1 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 [65](#) of file [capability](#).

11.27.2 Member Enumeration Documentation

11.27.2.1 enum L4::Cap_base::No_init_type

Enumerator:

No_init Special value for constructing uninitialized [Cap](#) objects.

Definition at line [71](#) of file [capability](#).

11.27.2.2 enum L4::Cap_base::Cap_type

Invalid capability type.

Enumerator:

Invalid Invalid capability selector.

Definition at line [82](#) of file [capability](#).

11.27.3 Constructor & Destructor Documentation

11.27.3.1 L4::Cap_base::Cap_base (l4_cap_idx_t *c*) throw () [inline, explicit, protected]

Generate a capability from its C representation.

Parameters

c the C capability selector

Definition at line [167](#) of file [capability](#).

11.27.3.2 L4::Cap_base::Cap_base (*l4_default_caps_t cap*) throw () [inline, explicit, protected]

Initialize capability with one of the default capability selectors.

Parameters

cap Capability selector.

Definition at line 177 of file [capability](#).

11.27.4 Member Function Documentation

11.27.4.1 l4_cap_idx_t L4::Cap_base::cap () const throw () [inline]

Return capability selector.

Returns

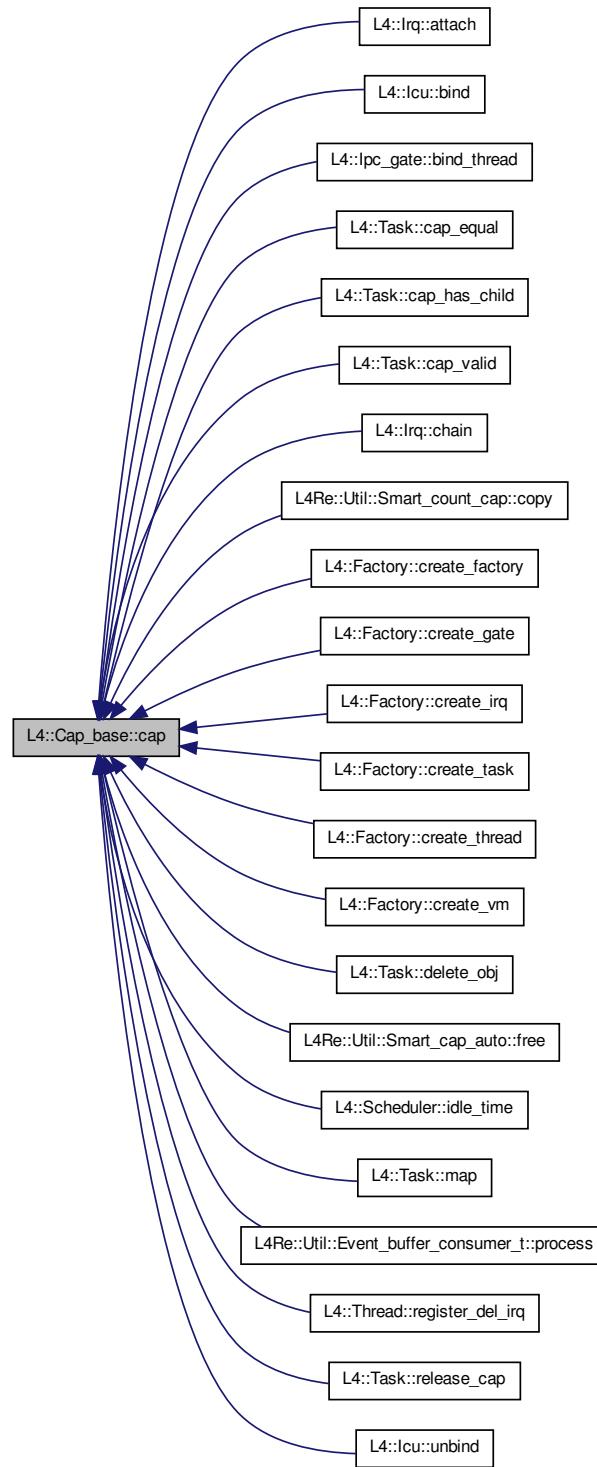
Capability selector.

Definition at line 91 of file [capability](#).

References [_c](#).

Referenced by [L4::Irq::attach\(\)](#), [L4::Icu::bind\(\)](#), [L4::Ipc_gate::bind_thread\(\)](#), [L4::Task::cap_equal\(\)](#), [L4::Task::cap_has_child\(\)](#), [L4::Task::cap_valid\(\)](#), [L4::Irq::chain\(\)](#), [L4Re::Util::Smart_count_cap< Unmap_flags >::copy\(\)](#), [L4::Factory::create_factory\(\)](#), [L4::Factory::create_gate\(\)](#), [L4::Factory::create_irq\(\)](#), [L4::Factory::create_task\(\)](#), [L4::Factory::create_thread\(\)](#), [L4::Factory::create_vml\(\)](#), [L4::Task::delete_obj\(\)](#), [L4Re::Util::Smart_cap_auto< Unmap_flags >::free\(\)](#), [L4::Scheduler::idle_time\(\)](#), [L4::Task::map\(\)](#), [L4Re::Util::Event_buffer_consumer_t< PAYLOAD >::process\(\)](#), [L4::Thread::register_del_irq\(\)](#), [L4::Task::release_cap\(\)](#), and [L4::Icu::unbind\(\)](#).

Here is the caller graph for this function:



11.27.4.2 `bool L4::Cap_base::is_valid () const throw () [inline]`

Test whether capability selector is not the invalid capability selector.

Returns

True if capability is not invalid, false if invalid

Examples:

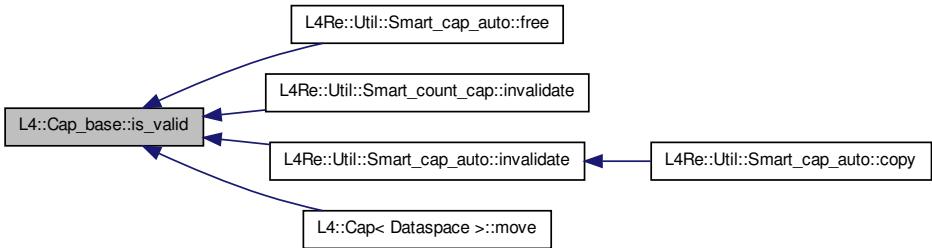
[examples/libs/l4re/c++/mem_alloc/ma+rm.cc](#), and [examples/libs/l4re/c++/shared_ds/ds_clnt.cc](#).

Definition at line 99 of file [capability](#).

References [_c](#).

Referenced by [L4Re::Util::Smart_cap_auto< Unmap_flags >::free\(\)](#), [L4Re::Util::Smart_count_cap< Unmap_flags >::invalidate\(\)](#), [L4Re::Util::Smart_cap_auto< Unmap_flags >::invalidate\(\)](#), and [L4::Cap< Dataspace >::move\(\)](#).

Here is the caller graph for this function:



11.27.4.3 `l4_fpage_t L4::Cap_base::fpage (unsigned rights = L4_FPAGE_RWX) const throw () [inline]`

Returns flex-page of the capability selector.

Parameters

rights Rights, defaults to 'rwx'

Returns

flex-page

Definition at line 109 of file [capability](#).

References [_c](#), and [l4_obj_fpage\(\)](#).

Referenced by [L4::Cap< Dataspace >::move\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.27.4.4 `l4_umword_t L4::Cap_base::snd_base(unsigned grant = 0, l4_cap_idx_t base = L4_INVALID_CAP) const throw() [inline]`

Returns send base.

Parameters

grant True object should be granted.
base Base capability selector

Returns

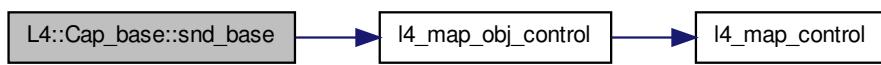
Map object.

Definition at line 118 of file [capability](#).

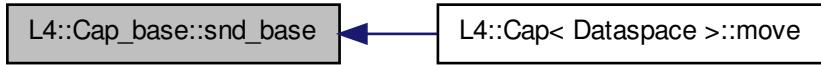
References [_c](#), [L4_INVALID_CAP](#), and [l4_map_obj_control\(\)](#).

Referenced by [L4::Cap< Dataspace >::move\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.27.4.5 `l4_msntag_t L4::Cap_base::validate (l4_utcb_t * u = l4_utcb()) const throw () [inline]`

Check whether a capability selector points to a valid capability.

Parameters

u UTCB of the caller

Returns

label = 0 valid, label > 0 invalid

Definition at line 513 of file [capability](#).

References [L4_BASE_TASK_CAP](#).

11.27.4.6 `l4_msntag_t L4::Cap_base::validate (Cap< Task > task, l4_utcb_t * u = l4_utcb()) const throw () [inline]`

Check whether a capability selector points to a valid capability.

Parameters

u UTCB of the caller

task Task to check the capability in

Returns

label = 0 valid, label > 0 invalid

Definition at line 509 of file [capability](#).

11.27.5 Field Documentation

11.27.5.1 `l4_cap_idx_t L4::Cap_base::_c [protected]`

The C representation of a capability selector.

Definition at line 186 of file [capability](#).

Referenced by `cap()`, `fpage()`, `invalidate()`, `is_valid()`, `operator!=()`, `L4::Smart_cap< T, SMART >::operator->()`, `L4::Cap< Dataspace >::operator->()`, `operator==()`, and `rnd_base()`.

The documentation for this class was generated from the following file:

- `l4/sys/capability`

11.28 cxx::Bitmap_base::Char< BITS > Class Template Reference

Helper abstraction for a byte contained in the bitmap.

11.28.1 Detailed Description

`template<long BITS> class cxx::Bitmap_base::Char< BITS >`

Helper abstraction for a byte contained in the bitmap.

Definition at line 66 of file `bitmap`.

The documentation for this class was generated from the following file:

- `l4/cxx(bitmap`

11.29 L4Re::Video::Color_component Class Reference

A 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 STREAM >
STREAM & **dump** (STREAM &s) const

Dump information on the view information to a stream.

11.29.1 Detailed Description

A color component.

Definition at line 32 of file [colors](#).

11.29.2 Constructor & Destructor Documentation

11.29.2.1 L4Re::Video::Color_component::Color_component (`unsigned char bits, unsigned char shift`) [\[inline\]](#)

Constructor.

Parameters

`bits` Number of bits used by the component

`shift` Position in bits of the component in the pixel

Definition at line 47 of file [colors](#).

11.29.3 Member Function Documentation

11.29.3.1 `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 54 of file [colors](#).

11.29.3.2 `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 60 of file [colors](#).

11.29.3.3 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 66 of file [colors](#).

11.29.3.4 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 74 of file [colors](#).

11.29.3.5 long unsigned L4Re::Video::Color_component::set (int *v*) const [inline]

Transform 16bit normalized value to the component in the color space.

Parameters

v Value return Converted value.

Definition at line 85 of file [colors](#).

11.29.3.6 template<typename STREAM > STREAM& L4Re::Video::Color_component::dump (STREAM & *s*) const [inline]

Dump information on the view information to a stream.

Parameters

s Stream

Returns

The stream

Definition at line 94 of file [colors](#).

The documentation for this class was generated from the following file:

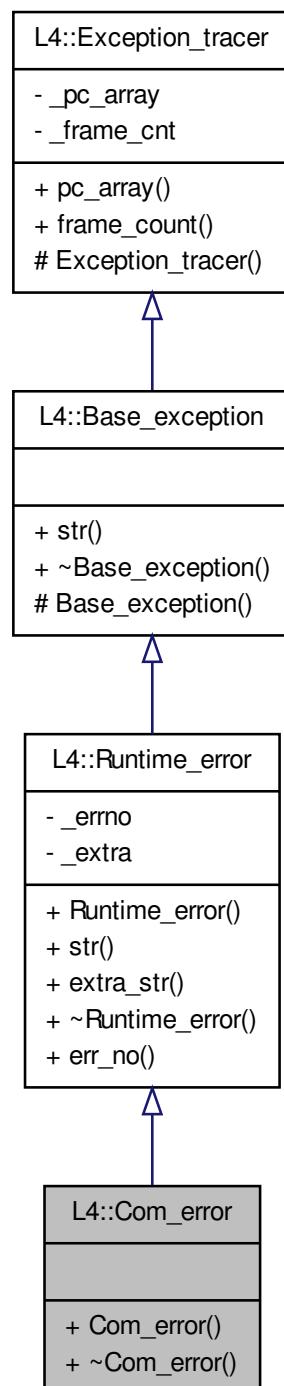
- l4/re/video/colors

11.30 L4::Com_error Class Reference

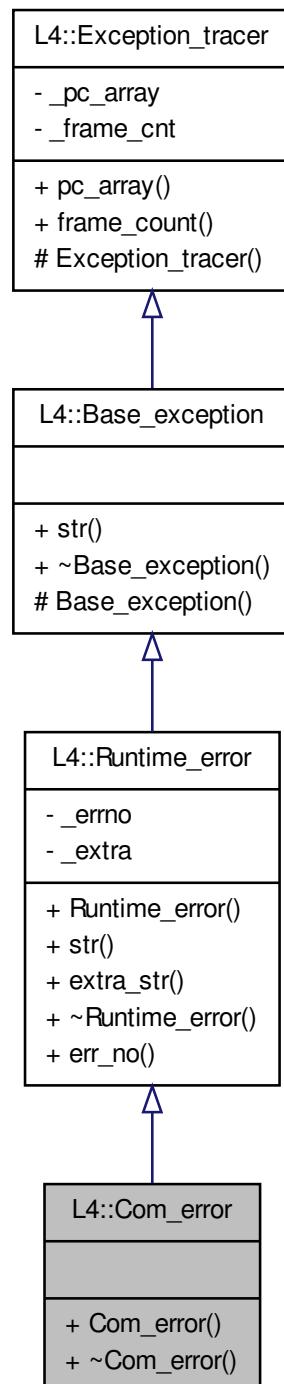
Error conditions during IPC.

```
#include <l4/cxx/exceptions>
```

Inheritance diagram for L4::Com_error:



Collaboration diagram for L4::Com_error:



Public Member Functions

- [Com_error \(long err\) throw \(\)](#)

Create a Com_error for the givel L4 IPC error code.

11.30.1 Detailed Description

Error conditions during IPC. This exception encapsulates all IPC error conditions of L4 IPC.

Definition at line [254](#) of file [exceptions](#).

11.30.2 Constructor & Destructor Documentation

11.30.2.1 L4::Com_error::Com_error (long err) throw () [inline, explicit]

Create a Com_error for the givel L4 IPC error code.

Parameters

err The L4 IPC error code (l4_ipc... return value).

Definition at line [261](#) of file [exceptions](#).

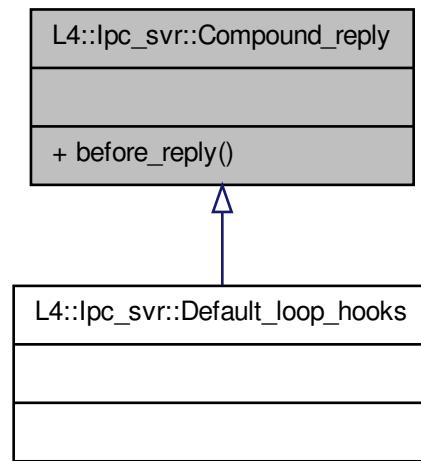
The documentation for this class was generated from the following file:

- l4/cxx/exceptions

11.31 L4::Ipc_svr::Compound_reply Struct Reference

Mix in for LOOP_HOOKS to always use compound reply and wait.

Inheritance diagram for L4::Ipc_svr::Compound_reply:



11.31.1 Detailed Description

Mix in for LOOP_HOOKS to always use compound reply and wait.

Definition at line [65](#) of file [ipc_server](#).

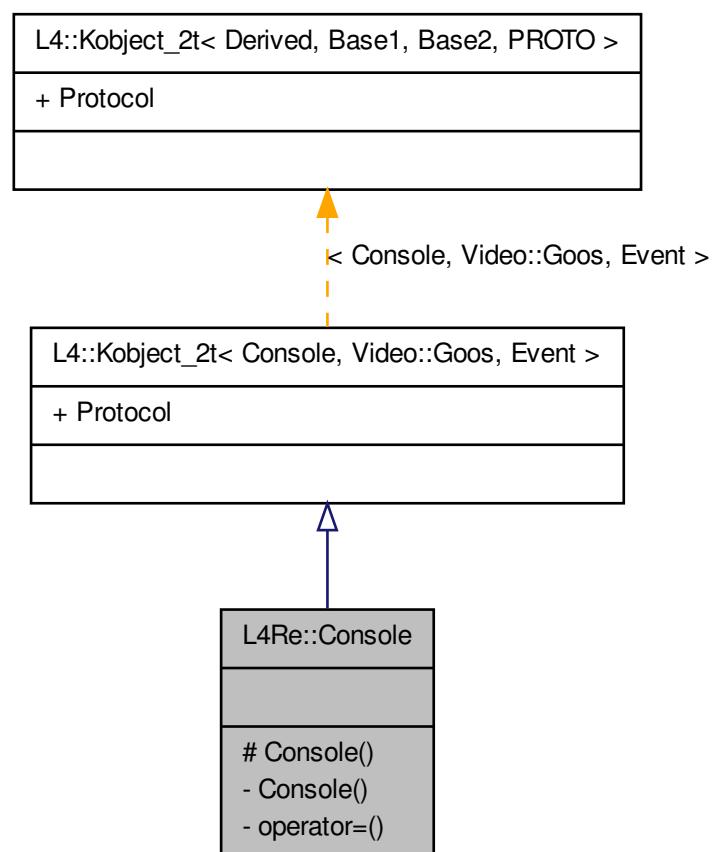
The documentation for this struct was generated from the following file:

- [14/cxx/ipc_server](#)

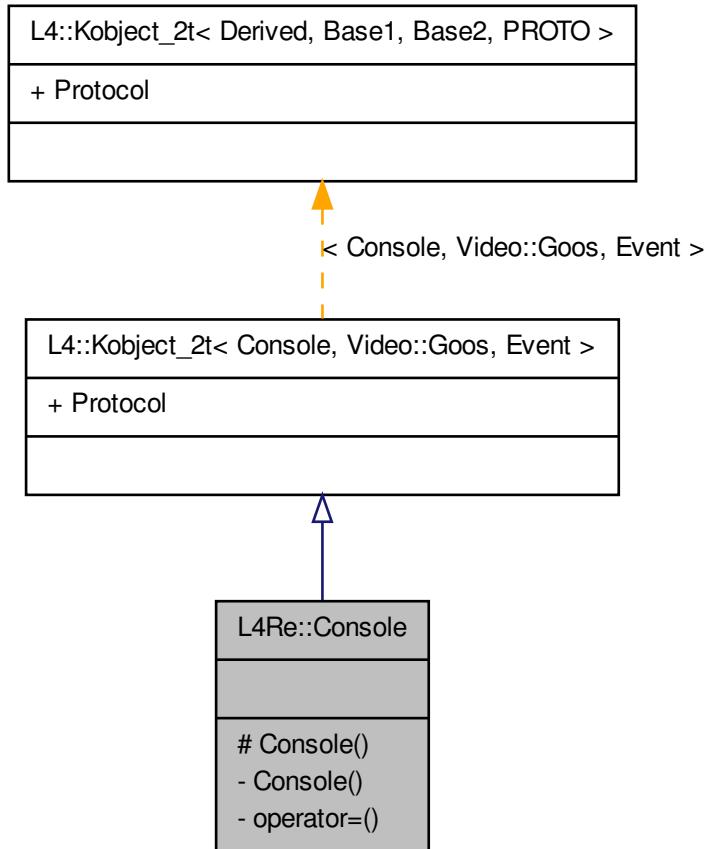
11.32 L4Re::Console Class Reference

[Console class](#).

Inheritance diagram for L4Re::Console:



Collaboration diagram for L4Re::Console:



11.32.1 Detailed Description

Console class.

Definition at line 38 of file [console](#).

The documentation for this class was generated from the following file:

- [14/re/console](#)

11.33 L4Re::Util::Counting_cap_alloc< COUNTERTYPE > Class Template Reference

Reference-counting cap allocator.

11.33.1 Detailed Description

```
template<typename COUNTERTYPE = L4Re::Util::Counter<unsigned char>> class
L4Re::Util::Counting_cap_alloc<COUNTERTYPE >
```

Reference-counting cap allocator.

Definition at line 52 of file [counting_cap_alloc](#).

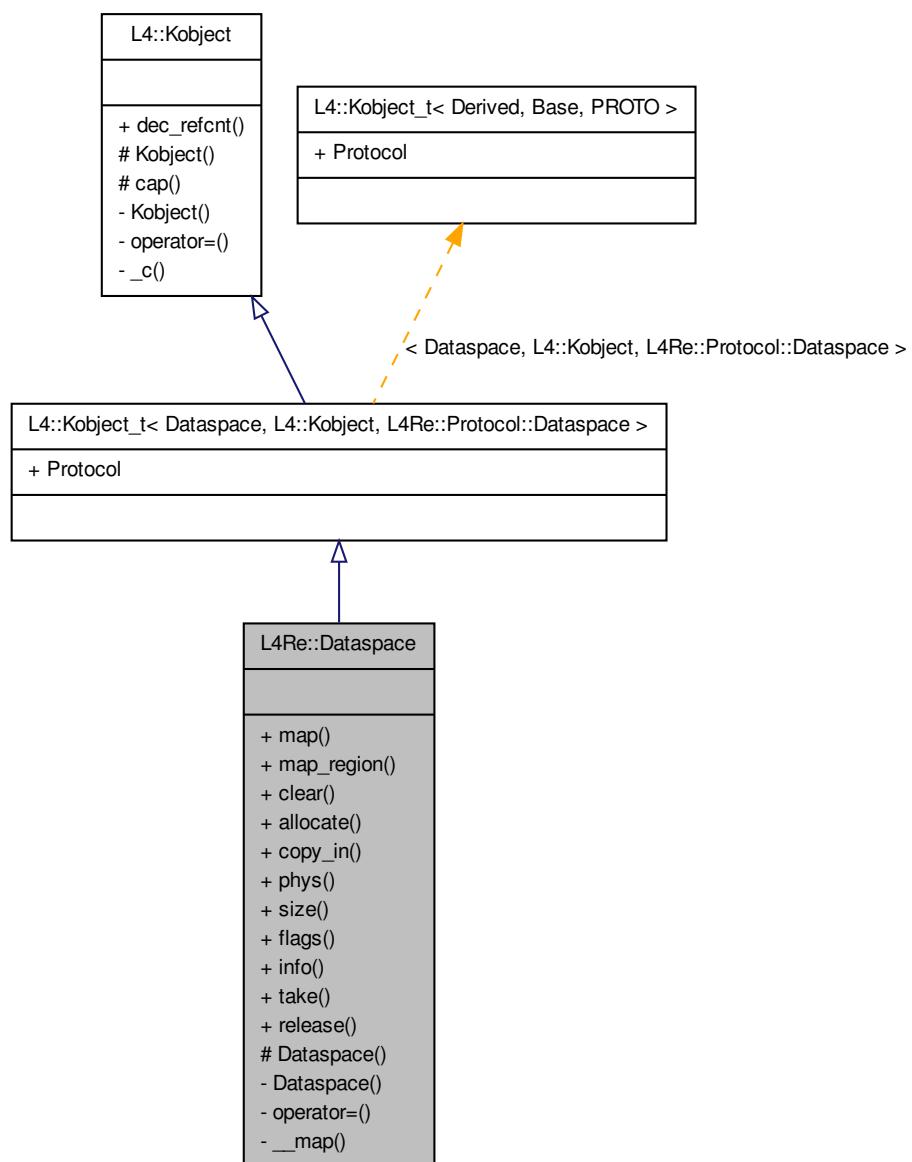
The documentation for this class was generated from the following file:

- [l4/re/util/counting_cap_alloc](#)

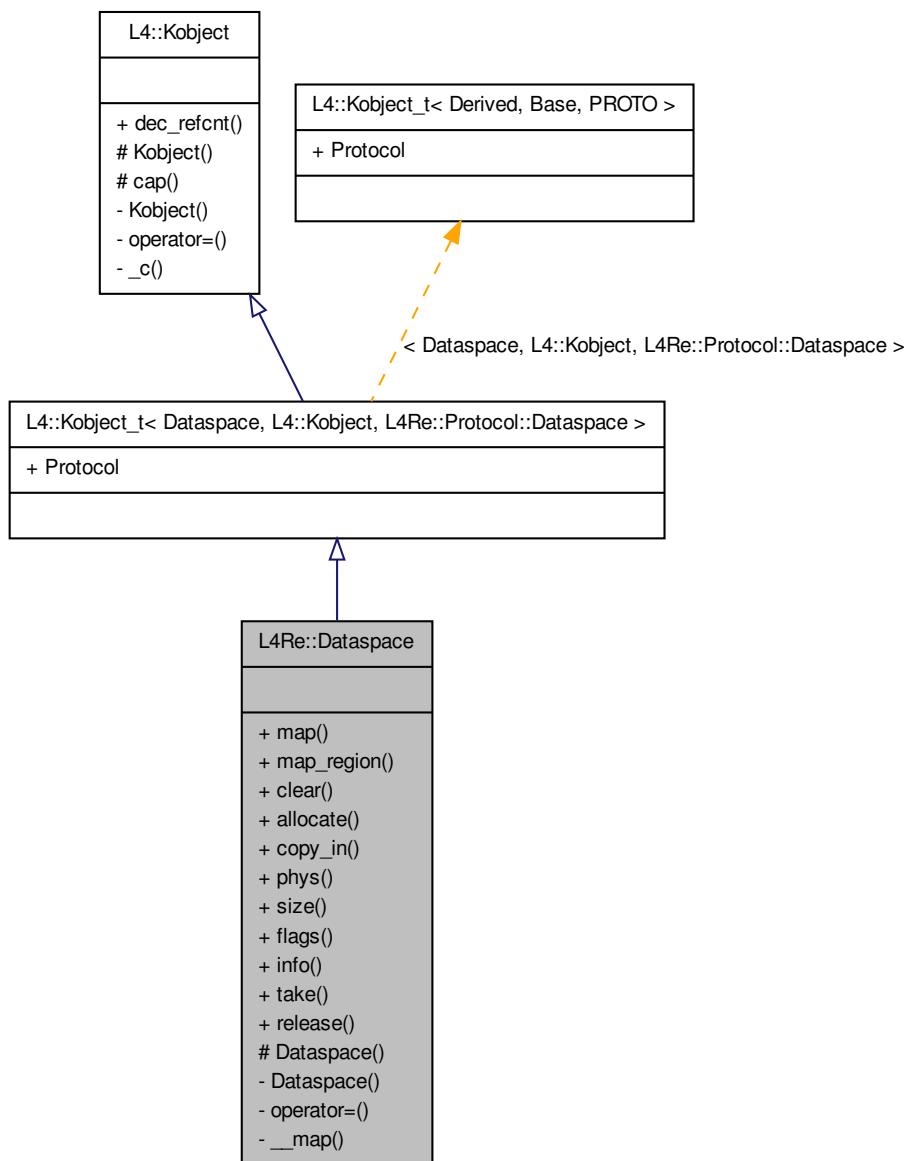
11.34 L4Re::Dataspace Class Reference

This class represents a data space.

Inheritance diagram for L4Re::Dataspace:



Collaboration diagram for L4Re::Dataspace:



Data Structures

- struct [Stats](#)

Information about the data space.

Public Types

- enum `Map_flags` { `Map_ro` = 0, `Map_rw` = 1 }

Flags for map operations.

Public Member Functions

- long `map` (`l4_addr_t` offset, unsigned long flags, `l4_addr_t` local_addr, `l4_addr_t` min_addr, `l4_addr_t` max_addr) const throw ()

Request a flex-page mapping from the data space.

- long `map_region` (`l4_addr_t` offset, unsigned long flags, `l4_addr_t` min_addr, `l4_addr_t` max_addr) const throw ()

Map a part of a data space completely.

- long `clear` (`l4_addr_t` offset, unsigned long size) const throw ()

Clear parts of a data space.

- long `allocate` (`l4_addr_t` offset, `l4_size_t` size) throw ()

Allocate a range in the dataspace.

- long `copy_in` (`l4_addr_t` dst_offs, `L4::Cap< Dataspace >` src, `l4_addr_t` src_offs, unsigned long size) const throw ()

Copy data space contents.

- long `phys` (`l4_addr_t` offset, `l4_addr_t` &phys_addr, `l4_size_t` &phys_size) const throw ()

Get the physical addresses of a data space.

- long `size` () const throw ()

Get size of a data space.

- long `flags` () const throw ()

Get flags of the data space.

- int `info` (`Stats` *stats) const throw ()

Get information on the data space.

11.34.1 Detailed Description

This class represents a data space. For more details, see [Data-Space API](#).

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_ds/ds_srv.cc`.

Definition at line 67 of file `dataspace`.

11.34.2 Member Enumeration Documentation

11.34.2.1 enum L4Re::Dataspace::Map_flags

Flags for map operations.

Enumerator:

Map_ro Request read-only mapping.

Map_rw Request writable mapping.

Definition at line 77 of file [dataspace](#).

11.34.3 Member Function Documentation

11.34.3.1 long L4Re::Dataspace::map (*l4_addr_t offset*, *unsigned long flags*, *l4_addr_t local_addr*, *l4_addr_t min_addr*, *l4_addr_t max_addr*) const throw()

Request a flex-page mapping from the data space.

Parameters

offset Offset to start within data space

flags map flags, see [Map_flags](#).

local_addr Local address to map to.

min_addr Defines start of receive window.

max_addr Defines end of receive window.

Returns

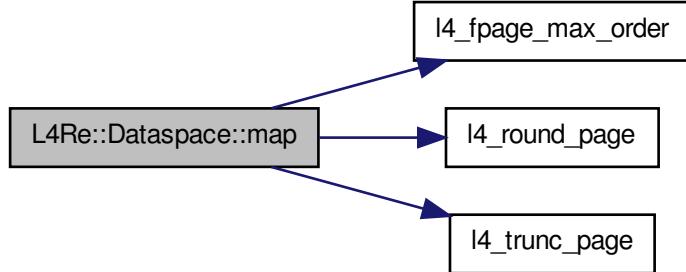
0 on success, <0 on error

- [-L4_ERANGE](#)
- [-L4_EPERM](#)
- IPC errors

Definition at line 94 of file [dataspace_impl.h](#).

References [l4_fpage_max_order\(\)](#), [L4_LOG2_PAGESIZE](#), [l4_round_page\(\)](#), and [l4_trunc_page\(\)](#).

Here is the call graph for this function:



11.34.3.2 long L4Re::Dataspace::map_region (*l4_addr_t offset*, *unsigned long flags*, *l4_addr_t min_addr*, *l4_addr_t max_addr*) const throw ()

Map a part of a data space completely.

Parameters

offset Offset to start within data space

flags map flags, see [Map_flags](#).

min_addr Defines start of receive window.

max_addr Defines end of receive window.

Returns

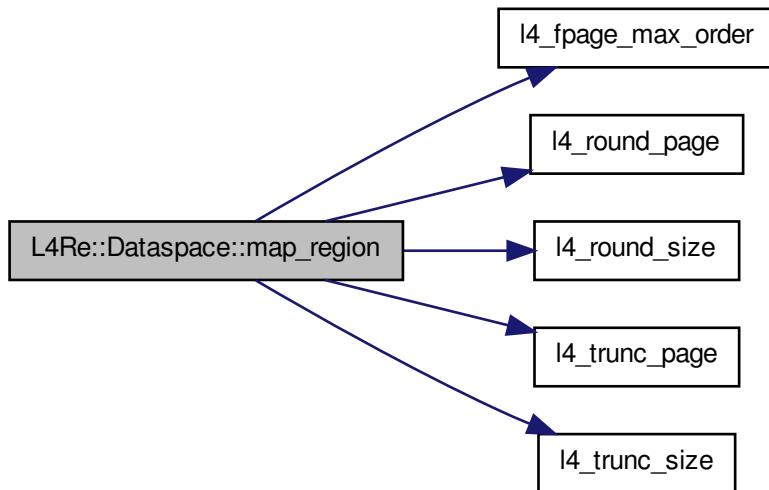
0 on success, <0 on error

- [-L4_ERANGE](#)
- [-L4_EPERM](#)
- IPC errors

Definition at line 57 of file [dataspace_impl.h](#).

References [EXPECT_FALSE](#), [l4_fpage_max_order\(\)](#), [l4_round_page\(\)](#), [l4_round_size\(\)](#), [l4_trunc_page\(\)](#), and [l4_trunc_size\(\)](#).

Here is the call graph for this function:



11.34.3.3 long L4Re::Dataspace::clear (*l4_addr_t offset*, *unsigned long size*) const throw ()

Clear parts of a data space.

Parameters

offset Offset within data space.

size Size to clear (in bytes).

Returns

>0 on sucess, <0 on error.

- [-L4_EACCESS](#)
- IPC errors

Clears memory. Depending on the type of memory the memory could also be deallocated and replaced by shared zero-page.

11.34.3.4 long L4Re::Dataspace::allocate (*l4_addr_t offset*, *l4_size_t size*) throw ()

Allocate a range in the dataspace.

Parameters

offset Offset in the dataspace, in bytes.

size Size of the range, in bytes.

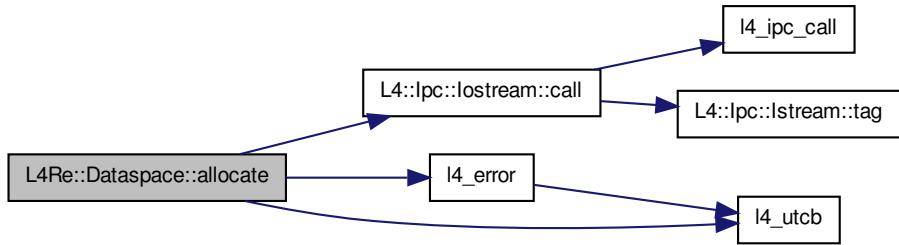
Returns

0 on success, <0 on error

Definition at line 177 of file [dataspace_impl.h](#).

References [L4::Ipc::Iostream::call\(\)](#), [L4Re::Protocol::Dataspace](#), [l4_error\(\)](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



11.34.3.5 long L4Re::Dataspace::copy_in (l4_addr_t dst_offs, L4::Cap< Dataspace > src, l4_addr_t src_offs, unsigned long size) const throw ()

Copy data space contents.

Parameters

dst_offs Offset in destination data space.

src Source data space.

src_offs Offset in the source data space.

size Size to copy (in bytes).

Returns

0 on success, <0 on error

- [-L4_EACCESS](#)
- [-L4_EINVAL](#)
- IPC errors

The copy operation may use copy-on-write mechanisms. The operation may also fail if both data spaces are not from the same data space manager or the data space managers do not cooperate.

11.34.3.6 `long L4Re::Dataspace::phys (l4_addr_t offset, l4_addr_t & phys_addr, l4_size_t & phys_size) const throw ()`

Get the physical addresses of a data space.

Parameters

offset Offset in data space

Return values

phys_addr Physical address.

phys_size Size of largest physically contiguous region in the data space (in bytes).

Returns

0 on success, <0 on error

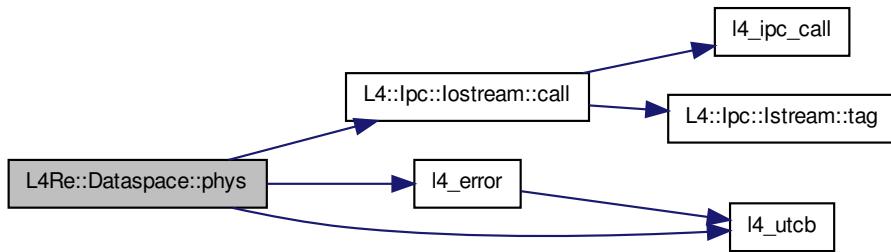
- -L4_EINVAL
- IPC errors

Get the physical address(es) of a data space. This call will only succeed on pinned memory data spaces.

Definition at line 164 of file `dataspace_impl.h`.

References `L4::Ipc::Iostream::call()`, `L4Re::Protocol::Dataspace`, `EXPECT_FALSE`, `l4_error()`, and `l4_utcb()`.

Here is the call graph for this function:



11.34.3.7 `long L4Re::Dataspace::size () const throw ()`

Get size of a data space.

Returns

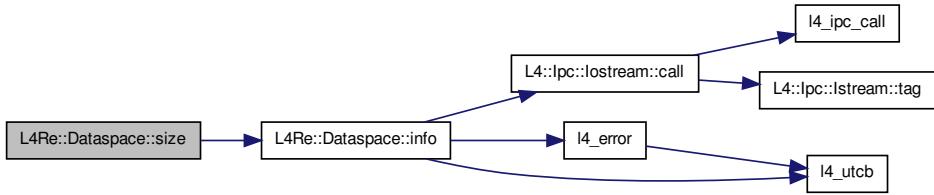
Size of the data space (in bytes), <0 on errors

- IPC errors

Definition at line 135 of file [dataspace_impl.h](#).

References [info\(\)](#), and [L4Re::Dataspace::Stats::size](#).

Here is the call graph for this function:



11.34.3.8 long L4Re::Dataspace::flags () const throw ()

Get flags of the data space.

Returns

Flags of the data space, <0 on errors

- IPC errors

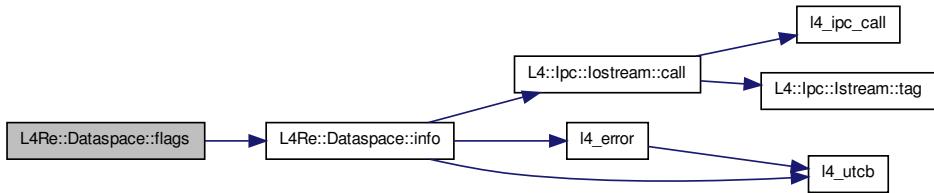
See also

[L4Re::Dataspace::Map_flags](#)

Definition at line 145 of file [dataspace_impl.h](#).

References [L4Re::Dataspace::Stats::flags](#), and [info\(\)](#).

Here is the call graph for this function:



11.34.3.9 int L4Re::Dataspace::info (Stats * stats) const throw ()

Get information on the data space.

Return values

info Data space information,

See also

[L4Re::Dataspace::Stats](#)

Returns

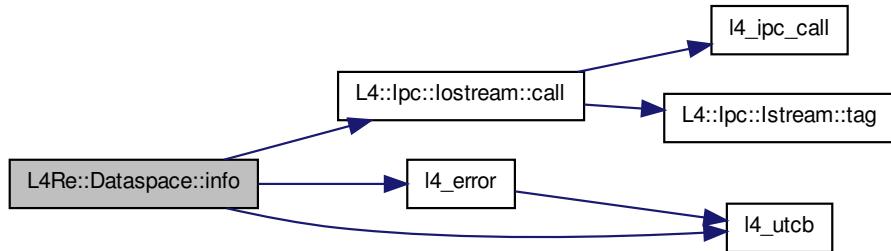
0 on success, < 0 on errors

Definition at line 122 of file [dataspace_impl.h](#).

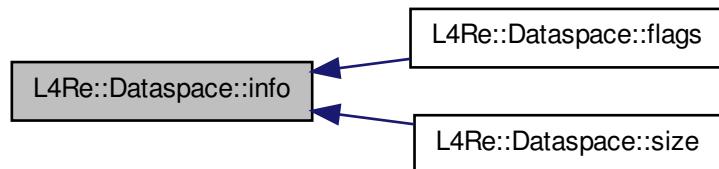
References [L4::Ipc::Iostream::call\(\)](#), [L4Re::Protocol::Dataspace](#), [EXPECT_FALSE](#), [l4_error\(\)](#), and [l4_utcb\(\)](#).

Referenced by [flags\(\)](#), and [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 files:

- [l4/re/dataspace](#)
- [l4/re/impl/dataspace_impl.h](#)

11.35 L4Re::Util::Dataspace_svr Class Reference

Dataspace server class.

Public Member Functions

- `int map (l4_addr_t offset, l4_addr_t local_addr, unsigned long flags, l4_addr_t min_addr, l4_addr_t max_addr, L4::Ipc::Snd_fpage &memory)`
Map a region of the dataspace.
- `virtual int map_hook (l4_addr_t offs, unsigned long flags, l4_addr_t min, l4_addr_t max)`
A hook that is called as the first operation in each map request.
- `virtual int phys (l4_addr_t offset, l4_addr_t &phys_addr, l4_size_t &phys_size) throw ()`
Return physical address for a virtual address.
- `virtual void take () throw ()`
Take a reference to this dataspace.
- `virtual unsigned long release () throw ()`
Release a reference to this dataspace.
- `virtual unsigned long copy (unsigned long dst_offs, l4_umword_t src_id, unsigned long src_offs, unsigned long size) throw ()`
Copy from src dataspace to this destination dataspace.
- `virtual long clear (unsigned long offs, unsigned long size) const throw ()`
Clear a region in the dataspace.
- `virtual unsigned long page_shift () const throw ()`
Define the size of the flexpage to map.

11.35.1 Detailed Description

Dataspace server class. The default implementation of the interface provides a continuously mapped dataspace.

Definition at line 48 of file [dataspace_svr](#).

11.35.2 Member Function Documentation

11.35.2.1 `int L4Re::Util::Dataspace_svr::map (l4_addr_t offset, l4_addr_t local_addr, unsigned long flags, l4_addr_t min_addr, l4_addr_t max_addr, L4::Ipc::Snd_fpage & memory)`

Map a region of the dataspace.

Parameters

`offset` Offset to start within data space

local_addr Local address to map to.
flags map flags, see Map_flags.
min_addr Defines start of receive window.
max_addr Defines end of receive window.

Return values

memory Send fpage to map

Returns

0 on success, <0 on error

11.35.2.2 virtual int L4Re::Util::Dataspace_svr::map_hook (**l4_addr_t offs**, **unsigned long flags**, **l4_addr_t min**, **l4_addr_t max**) [inline, virtual]

A hook that is called as the first operation in each map request.

Parameters

offs Offs param to map
flags Flags param to map
min Min param to map
max Max param to map

Returns

< 0 on error and the map request will be aborted with that error >= 0: ok

See also

[map](#)

Definition at line 97 of file [dataspace_svr](#).

11.35.2.3 virtual int L4Re::Util::Dataspace_svr::phys (**l4_addr_t offset**, **l4_addr_t & phys_addr**, **l4_size_t & phys_size**) throw () [virtual]

Return physical address for a virtual address.

Parameters

offset Offset into the dataspace

Return values

phys_addr Physical address
phys_size Size of continuous physical region

Returns

Zero on success, else failure

11.35.2.4 virtual void L4Re::Util::Dataspace_svr::take () throw () [inline, virtual]

Take a reference to this dataspace.

Default does nothing.

Definition at line 120 of file [dataspace_svr](#).

11.35.2.5 virtual unsigned long L4Re::Util::Dataspace_svr::release () throw () [inline, virtual]

Release a reference to this dataspace.

Returns

Number of references to the dataspace

Default does nothing and returns always zero.

Definition at line 130 of file [dataspace_svr](#).

11.35.2.6 virtual unsigned long L4Re::Util::Dataspace_svr::copy (unsigned long *dst_offs*, l4_umword_t *src_id*, unsigned long *src_offs*, unsigned long *size*) throw () [inline, virtual]

Copy from src dataspace to this destination dataspace.

Parameters

dst_offs Offset into the destination dataspace

src_id Local id of the source dataspace

src_offs Offset into the source dataspace

size Number of bytes to copy

Returns

Number of bytes copied

Definition at line 143 of file [dataspace_svr](#).

11.35.2.7 virtual long L4Re::Util::Dataspace_svr::clear (unsigned long *offs*, unsigned long *size*) const throw () [virtual]

Clear a region in the dataspace.

Parameters

offs Start of the region

size Size of the region

**11.35.2.8 virtual unsigned long L4Re::Util::Dataspace_svr::page_shift () const throw ()
[inline, virtual]**

Define the size of the flexpage to map.

Returns

flexpage size

Definition at line 160 of file [dataspace_svr](#).

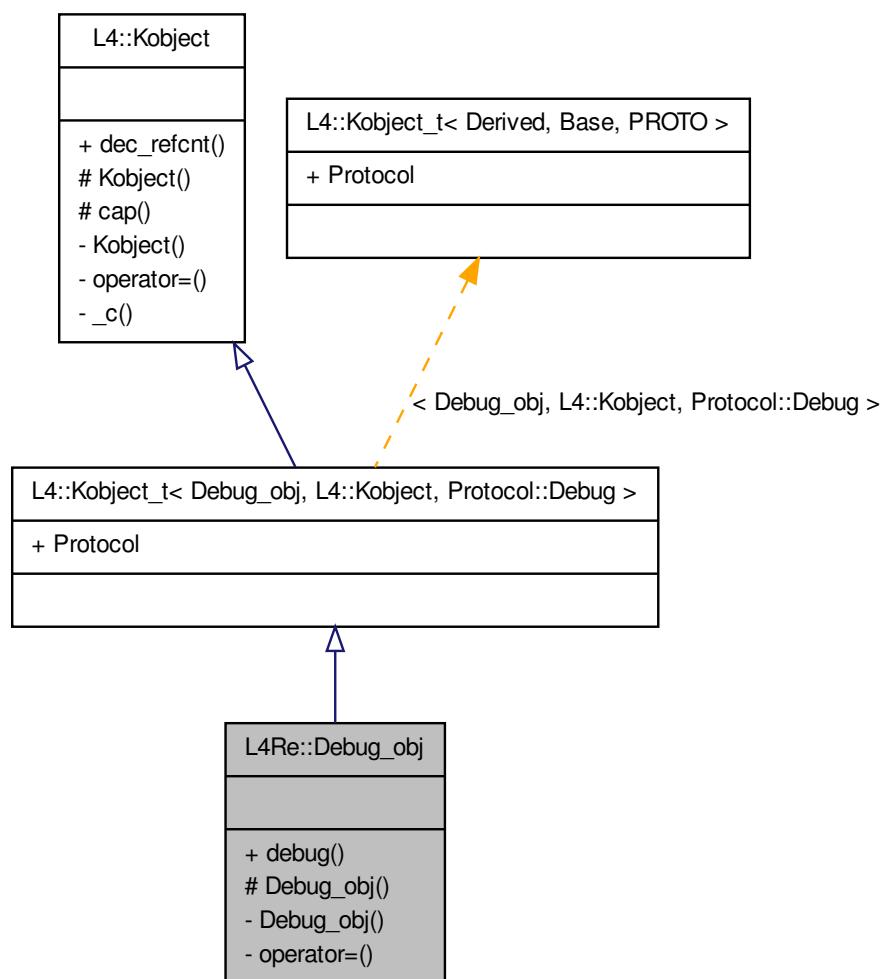
The documentation for this class was generated from the following file:

- 14/re/util/dataspace_svr

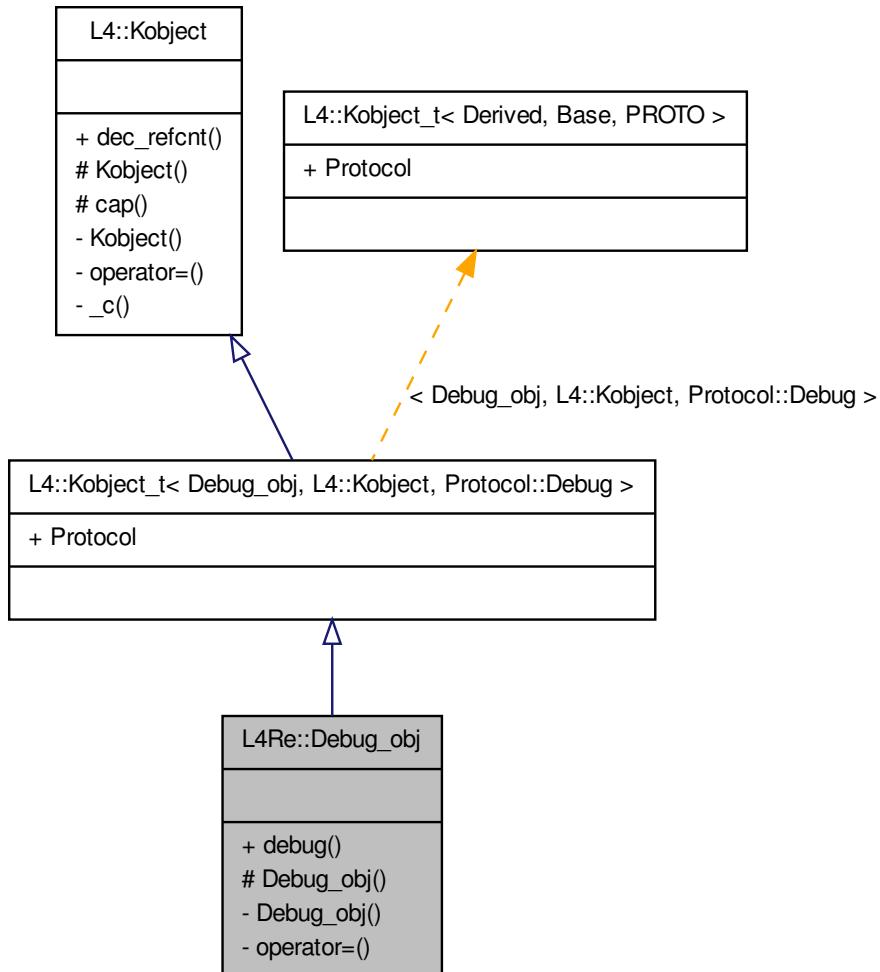
11.36 L4Re::Debug_obj Class Reference

Debug interface.

Inheritance diagram for L4Re::Debug_obj:



Collaboration diagram for L4Re::Debug_obj:



Public Member Functions

- int `debug` (unsigned long function) const throw ()

Debug call.

11.36.1 Detailed Description

Debug interface.

See also

[Debugging API](#) .

Definition at line 50 of file [debug](#).

11.36.2 Member Function Documentation

11.36.2.1 int L4Re::Debug_obj::debug (*unsigned long function*) const throw ()

Debug call.

Parameters

function Function to call.

Returns

- L4_EOK

- IPC errors

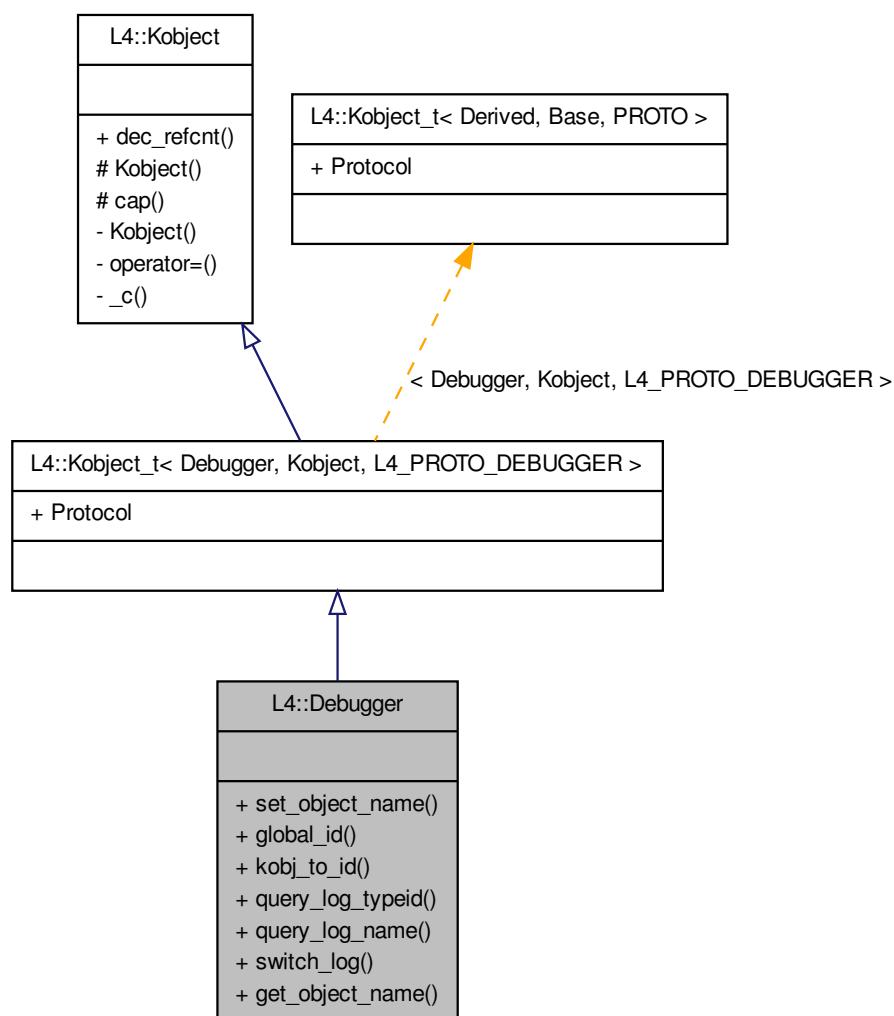
The documentation for this class was generated from the following file:

- [l4/re/debug](#)

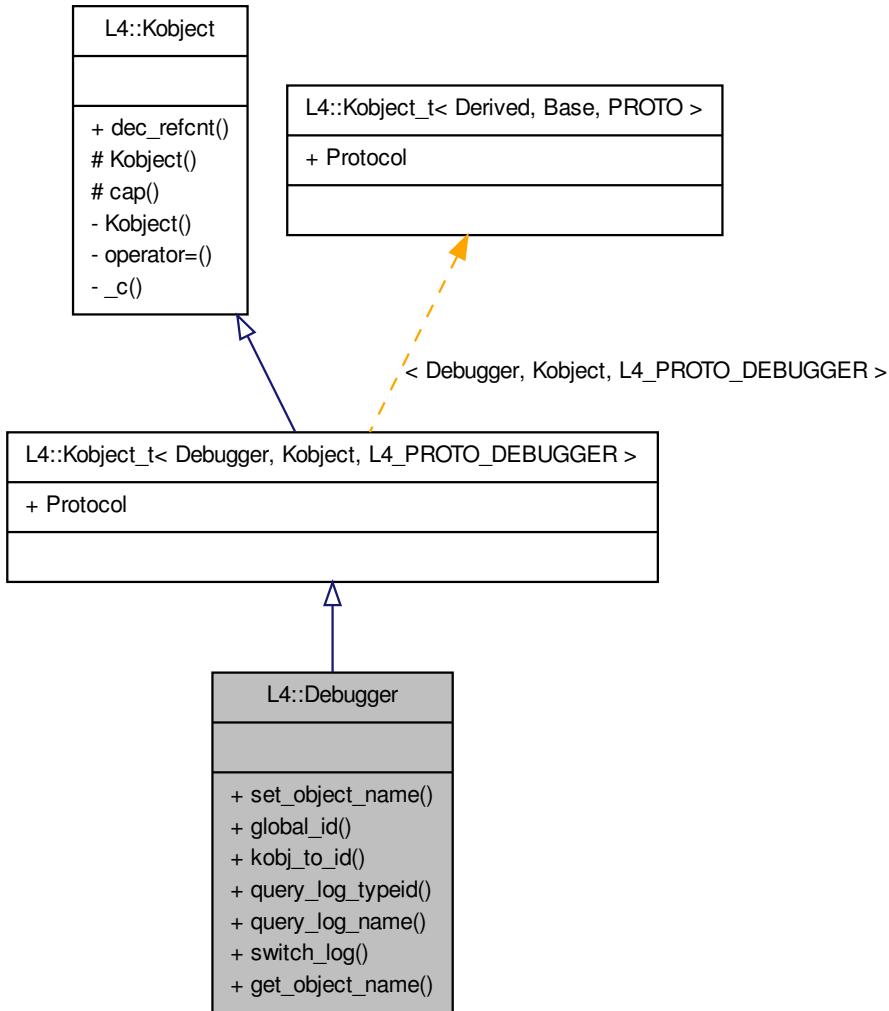
11.37 L4::Debugger Class Reference

[Debugger](#) interface.

Inheritance diagram for L4::Debugger:



Collaboration diagram for L4::Debugger:



Public Member Functions

- `l4_mshtag_t set_object_name (const char *name, l4_utcb_t *utcb=l4_utcb()) throw ()`
- `unsigned long global_id (l4_utcb_t *utcb=l4_utcb()) throw ()`
- `unsigned long kobj_to_id (l4_addr_t kobjp, l4_utcb_t *utcb=l4_utcb()) throw ()`
- `int query_log_typeid (const char *name, unsigned idx, l4_utcb_t *utcb=l4_utcb()) throw ()`
- `int query_log_name (unsigned idx, char *name, unsigned namelen, char *shortname, unsigned shortnamelen, l4_utcb_t *utcb=l4_utcb()) throw ()`
- `l4_mshtag_t switch_log (const char *name, unsigned on_off, l4_utcb_t *utcb=l4_utcb()) throw ()`
- `l4_mshtag_t get_object_name (unsigned id, char *name, unsigned size, l4_utcb_t *utcb=l4_utcb()) throw ()`

11.37.1 Detailed Description

`Debugger` interface. #include <l4/sys/debugger>

Definition at line 39 of file `debugger`.

11.37.2 Member Function Documentation

11.37.2.1 `l4_mshtag_t L4::Debugger::set_object_name (const char * name, l4_utcb_t * utcb = 14_utcb()) throw () [inline]`

The string name of kernel object.

Parameters

cap Capability

name Name

This is a debugging facility, the call might be invalid.

Note

the *cap* argument is the implicit *this* pointer.

Definition at line 45 of file `debugger`.

11.37.2.2 `unsigned long L4::Debugger::global_id (l4_utcb_t * utcb = 14_utcb()) throw () [inline]`

Get the globally unique ID of the object behind a capability.

Parameters

cap Capability

Returns

~0UL on non-valid capability, ID otherwise

This is a debugging facility, the call might be invalid.

Note

the *cap* argument is the implicit *this* pointer.

Definition at line 53 of file `debugger`.

11.37.2.3 `unsigned long L4::Debugger::kobj_to_id (l4_addr_t kobjp, l4_utcb_t * utcb = 14_utcb()) throw () [inline]`

Get the globally unique ID of the object behind the kobject pointer.

Parameters

cap Capability

kobjp Kobject pointer

Returns

~0UL on non-valid capability or invalid kobject pointer, ID otherwise

This is a debugging facility, the call might be invalid.

Note

the *cap* argument is the implicit *this* pointer.

Definition at line 60 of file [debugger](#).

11.37.2.4 `int L4::Debugger::query_log_typeid (const char * name, unsigned idx, l4_utcb_t * utcb = l4_utcb()) throw () [inline]`

Note

the *cap* argument is the implicit *this* pointer.

Definition at line 68 of file [debugger](#).

11.37.2.5 `int L4::Debugger::query_log_name (unsigned idx, char * name, unsigned namelen, char * shortname, unsigned shortnamelen, l4_utcb_t * utcb = l4_utcb()) throw () [inline]`

Note

the *cap* argument is the implicit *this* pointer.

Definition at line 76 of file [debugger](#).

11.37.2.6 `l4_mshtag_t L4::Debugger::switch_log (const char * name, unsigned on_off, l4_utcb_t * utcb = l4_utcb()) throw () [inline]`

Note

the *cap* argument is the implicit *this* pointer.

Definition at line 89 of file [debugger](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.37.2.7 l4_mshtag_t L4::Debugger::get_object_name (`unsigned id, char * name, unsigned size, l4_utcb_t * utcb = l4_utcb()`) throw () [inline]

Note

the `cap` argument is the implicit *this* pointer.

Definition at line 97 of file [debugger](#).

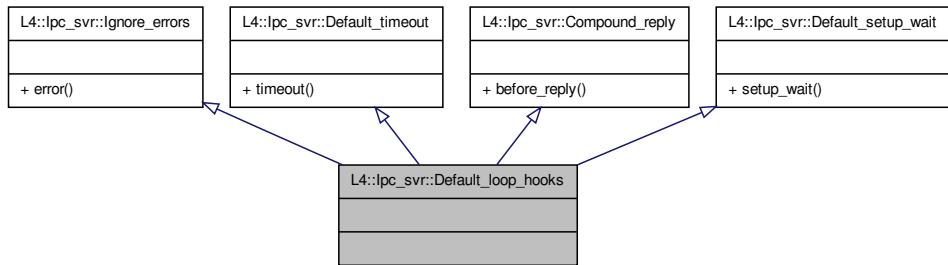
The documentation for this class was generated from the following file:

- l4/sys/debugger

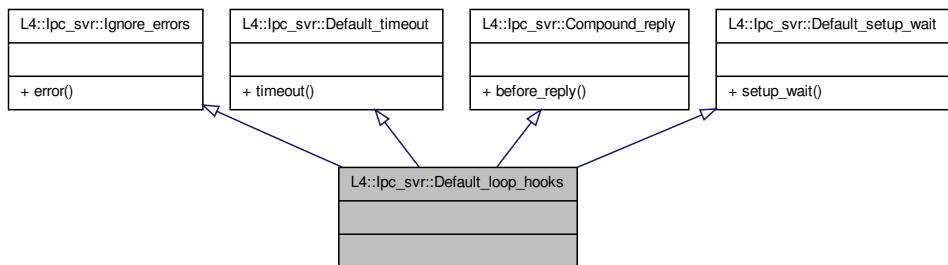
11.38 L4::Ipc_svr::Default_loop_hooks Struct Reference

Default LOOP_HOOKS.

Inheritance diagram for L4::Ipc_svr::Default_loop_hooks:



Collaboration diagram for L4::Ipc_svr::Default_loop_hooks:



11.38.1 Detailed Description

Default LOOP_HOOKS. Combination of [Ignore_errors](#), [Default_timeout](#), [Compound_reply](#), and [Default_setup_wait](#).

Definition at line 83 of file [ipc_server](#).

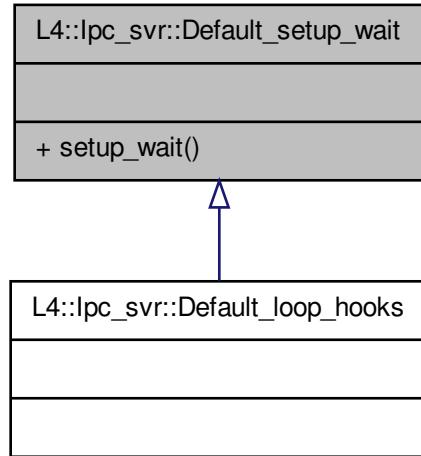
The documentation for this struct was generated from the following file:

- [l4/cxx/ipc_server](#)

11.39 L4::Ipc_svr::Default_setup_wait Struct Reference

Mix in for LOOP_HOOKS for setup_wait no op.

Inheritance diagram for L4::Ipc_svr::Default_setup_wait:



11.39.1 Detailed Description

Mix in for LOOP_HOOKS for setup_wait no op.

Definition at line 74 of file [ipc_server](#).

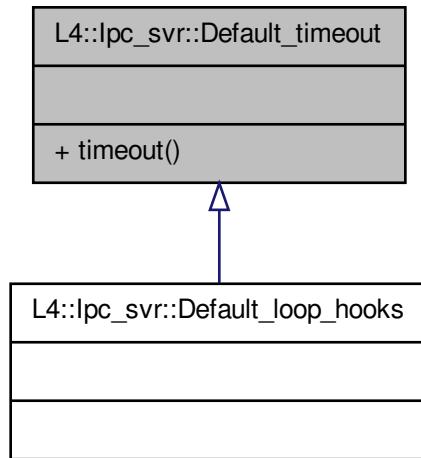
The documentation for this struct was generated from the following file:

- [l4/cxx/ipc_server](#)

11.40 L4::Ipc_svr::Default_timeout Struct Reference

Mix in for LOOP_HOOKS to use a 0 send and a infinite receive timeout.

Inheritance diagram for L4::Ipc_svr::Default_timeout:



11.40.1 Detailed Description

Mix in for LOOP_HOOKS to use a 0 send and a infinite receive timeout.

Definition at line 59 of file [ipc_server](#).

The documentation for this struct was generated from the following file:

- [l4/cxx/ipc_server](#)

11.41 cxx::Bits::Direction Struct Reference

The direction to go in a binary search tree.

```
#include <bst_base.h>
```

Public Types

- enum `Direction_e` { `L` = 0, `R` = 1, `N` = 2 }

The literal direction values.

Public Member Functions

- `Direction()`

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`
- `bool operator!=(Direction_e o) const`
- `bool operator==(Direction o) const`
- `bool operator!=(Direction o) const`

11.41.1 Detailed Description

The direction to go in a binary search tree.

Definition at line 39 of file `bst_base.h`.

11.41.2 Member Enumeration Documentation

11.41.2.1 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 42 of file `bst_base.h`.

11.41.3 Member Function Documentation

11.41.3.1 **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 63 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:

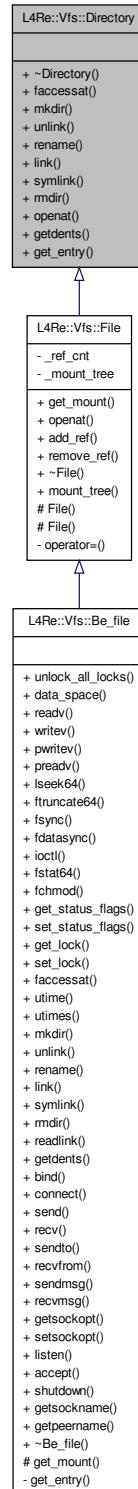
- l4/cxx/bits/bst_base.h

11.42 L4Re::Vfs::Directory Class Reference

Interface for a POSIX file that is a directory.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::Directory:



Public Member Functions

- virtual int **faccessat** (const char *path, int mode, int flags)=0 throw ()

Check access permissions on the given file.
- virtual int **mkdir** (const char *path, mode_t mode)=0 throw ()

Create a new subdirectory.
- virtual int **unlink** (const char *path)=0 throw ()

Unlink the given file from that directory.
- virtual int **rename** (const char *src_path, const char *dst_path)=0 throw ()

Rename the given file.
- virtual int **link** (const char *src_path, const char *dst_path)=0 throw ()

Create a hard link (second name) for the given file.
- virtual int **symlink** (const char *src_path, const char *dst_path)=0 throw ()

Create a symbolic link for the given file.
- virtual int **rmdir** (const char *)=0 throw ()

Delete an empty directory.

11.42.1 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 use always the combined [L4Re::Vfs::File](#) interface.

Definition at line 141 of file [vfs.h](#).

11.42.2 Member Function Documentation

11.42.2.1 virtual int L4Re::Vfs::Directory::faccessat (const char * *path*, int *mode*, int *flags*) throw () [pure virtual]

Check access permissions on the given file.

Backend function for POSIX access and faccessat functions.

Parameters

path The path relative to this directory. Note: *path* is relative to this directory and may contain subdirectories.

mode The access mode to check.

flags The flags as in POSIX faccessat (AT_EACCESS, AT_SYMLINK_NOFOLLOW).

Returns

0 on success, or <0 on error.

11.42.2.2 virtual int L4Re::Vfs::Directory::mkdir (const char * *path*, mode_t *mode*) throw () [pure virtual]

Create a new subdirectory.

Backend for POSIX mkdir and mkdirat function calls.

Parameters

path The name of the subdirectory to create. Note: *path* is relative to this directory and may contain subdirectories.

mode 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.

11.42.2.3 virtual int L4Re::Vfs::Directory::unlink (const char * *path*) throw () [pure virtual]

Unlink the given file from that directory.

Backend for the POSIX unlink and unlinkat functions.

Parameters

path The name to the file to unlink. Note: *path* is relative to this directory and may contain subdirectories.

Returns

0 on success, or <0 on error.

11.42.2.4 virtual int L4Re::Vfs::Directory::rename (const char * *src_path*, const char * *dst_path*) throw () [pure virtual]

Rename the given file.

Backend for the POSIX rename, renameat functions.

Parameters

src_path The old name to the file to rename. Note: *src_path* is relative to this directory and may contain subdirectories.

dst_path The new name for the file. Note: *dst_path* is relative to this directory and may contain subdirectories.

Returns

0 on success, or <0 on error.

11.42.2.5 virtual int L4Re::Vfs::Directory::link (const char * *src_path*, const char * *dst_path*) throw () [pure virtual]

Create a hard link (second name) for the given file.

Backend for the POSIX link and linkat functions.

Parameters

src_path The old name to the file. Note: *src_path* is relative to this directory and may contain subdirectories.

dst_path The new (second) name for the file. Note: *dst_path* is relative to this directory and may contain subdirectories.

Returns

0 on success, or <0 on error.

11.42.2.6 virtual int L4Re::Vfs::Directory::symlink (const char * *src_path*, const char * *dst_path*) throw () [pure virtual]

Create a symbolic link for the given file.

Backend for the POSIX symlink and symlinkat functions.

Parameters

src_path The old name to the file. Note: *src_path* shall be an absolute path.

dst_path The name for symlink. Note: *dst_path* is relative to this directory and may contain subdirectories.

Returns

0 on success, or <0 on error.

11.42.2.7 virtual int L4Re::Vfs::Directory::rmdir (const char *) throw () [pure virtual]

Delete an empty directory.

Backend for POSIX rmdir, rmdirat functions.

Parameters

path The name of the directory to remove. Note: *path* is relative to this directory and may contain subdirectories.

Returns

0 on success, or <0 on error.

The documentation for this class was generated from the following file:

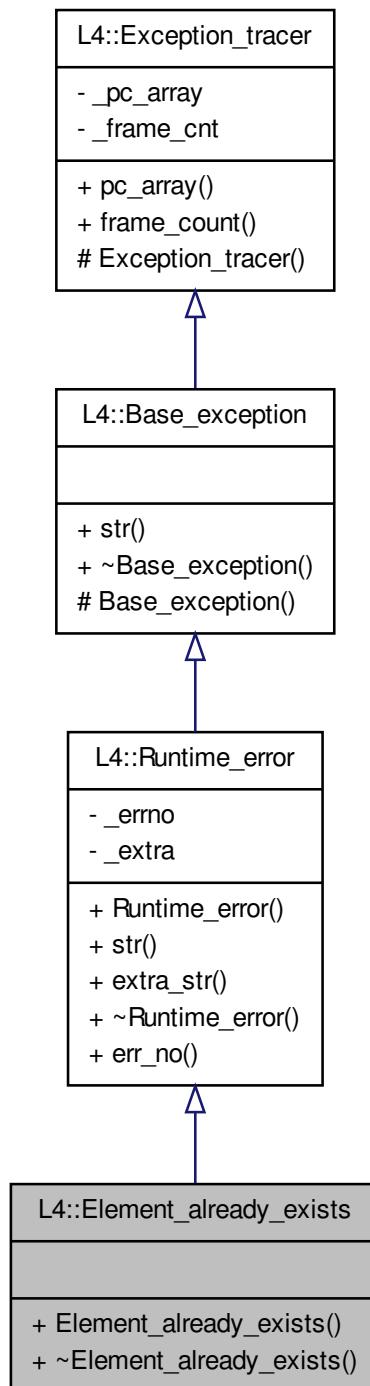
- l4/l4re_vfs/vfs.h

11.43 L4::Element_already_exists Class Reference

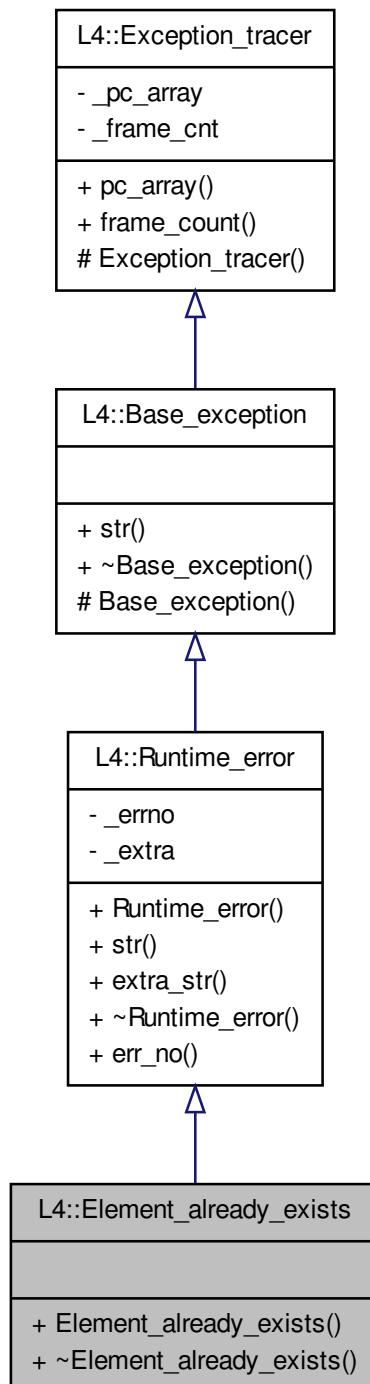
Exception for duplicate element insertions.

```
#include <l4/cxx/exceptions>
```

Inheritance diagram for L4::Element_already_exists:



Collaboration diagram for L4::Element_already_exists:



11.43.1 Detailed Description

Exception for duplicate element insertions.

Definition at line [182](#) of file [exceptions](#).

The documentation for this class was generated from the following file:

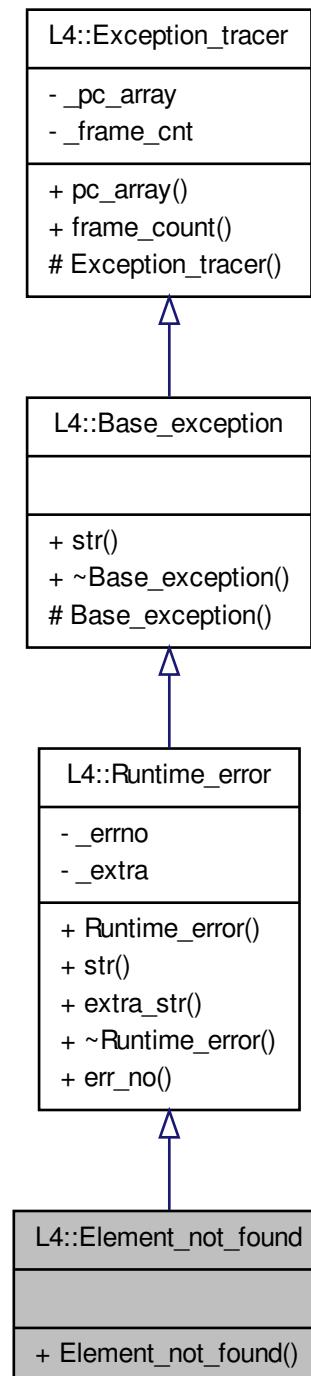
- [l4/cxx/exceptions](#)

11.44 L4::Element_not_found Class Reference

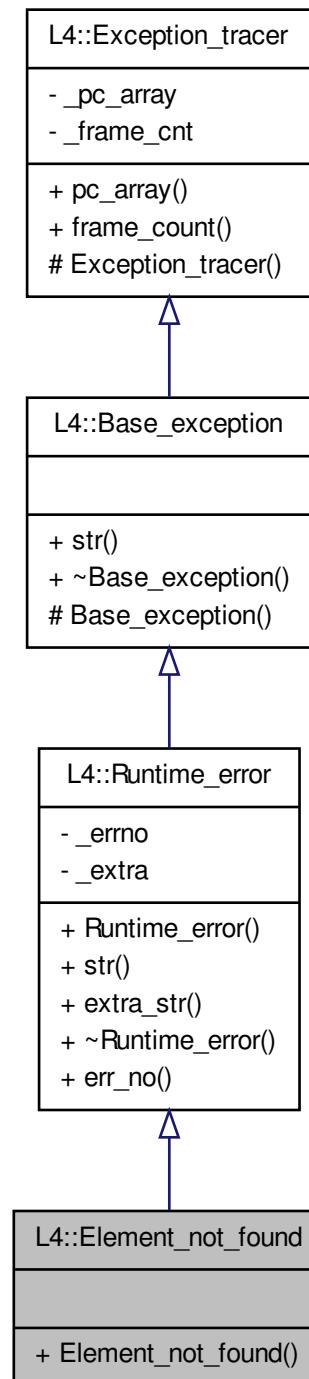
Exception for a failed lookup (element not found).

```
#include <l4/cxx/exceptions>
```

Inheritance diagram for L4::Element_not_found:



Collaboration diagram for L4::Element_not_found:



11.44.1 Detailed Description

Exception for a failed lookup (element not found).

Definition at line 211 of file [exceptions](#).

The documentation for this class was generated from the following file:

- l4/cxx/exceptions

11.45 Elf32_Dyn Struct Reference

ELF32 dynamic entry.

```
#include <elf.h>
```

Data Fields

- [Elf32_Sword d_tag](#)
see DT_values
- [Elf32_Word d_val](#)
integer values with various interpret.
- [Elf32_Addr d_ptr](#)
program virtual addresses

11.45.1 Detailed Description

ELF32 dynamic entry.

Definition at line 459 of file [elf.h](#).

11.45.2 Field Documentation

11.45.2.1 Elf32_Word Elf32_Dyn::d_val

integer values with various interpret.

Definition at line 462 of file [elf.h](#).

The documentation for this struct was generated from the following file:

- l4/util/elf.h

11.46 Elf32_Ehdr Struct Reference

ELF32 header.

```
#include <elf.h>
```

Data Fields

- `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 eip
- `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`
of entries in prog.
- `Elf32_Half e_shentsize`
size of section header entry
- `Elf32_Half e_shnum`
of entries in sect.
- `Elf32_Half e_shstrndx`
sect.head.tab.idx of strtab

11.46.1 Detailed Description

ELF32 header.

Definition at line 118 of file `elf.h`.

11.46.2 Field Documentation

11.46.2.1 Elf32_Half Elf32_Ehdr::e_phnum

of entries in prog.

head. tab.

Definition at line 129 of file [elf.h](#).

11.46.2.2 Elf32_Half Elf32_Ehdr::e_shnum

of entries in sect.

head. tab.

Definition at line 131 of file [elf.h](#).

The documentation for this struct was generated from the following file:

- 14/util/elf.h

11.47 Elf32_Phdr Struct Reference

ELF32 program header.

```
#include <elf.h>
```

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

11.47.1 Detailed Description

ELF32 program header.

Definition at line 378 of file [elf.h](#).

The documentation for this struct was generated from the following file:

- [l4/util/elf.h](#)

11.48 Elf32_Shdr Struct Reference

ELF32 section header - figure 1-9, page 1-9.

```
#include <elf.h>
```

Data Fields

- [Elf32_Word sh_name](#)
name of sect (idx into shstrtab)
- [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

11.48.1 Detailed Description

ELF32 section header - figure 1-9, page 1-9.

Definition at line 302 of file [elf.h](#).

The documentation for this struct was generated from the following file:

- l4/util/elf.h

11.49 Elf32_Sym Struct Reference

ELF32 symbol table entry.

```
#include <elf.h>
```

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

11.49.1 Detailed Description

ELF32 symbol table entry.

Definition at line 759 of file [elf.h](#).

The documentation for this struct was generated from the following file:

- l4/util/elf.h

11.50 Elf64_Dyn Struct Reference

ELF64 dynamic entry.

```
#include <elf.h>
```

Data Fields

- `Elf64_Sxword d_tag`
see DT_values
- `Elf64_Xword d_val`
integer values with various interpret.
- `Elf64_Addr d_ptr`
program virtual addresses

11.50.1 Detailed Description

ELF64 dynamic entry.

Definition at line 468 of file [elf.h](#).

11.50.2 Field Documentation

11.50.2.1 Elf64_Xword Elf64_Dyn::d_val

integer values with various interpret.

Definition at line 471 of file [elf.h](#).

The documentation for this struct was generated from the following file:

- l4/util/elf.h

11.51 Elf64_Ehdr Struct Reference

ELF64 header.

```
#include <elf.h>
```

Data Fields

- `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 eip

- `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`
of entries in prog.
- `Elf64_Half e_shentsize`
size of section header entry
- `Elf64_Half e_shnum`
of entries in sect.
- `Elf64_Half e_shstrndx`
sect.head.tab.idx of strtab

11.51.1 Detailed Description

ELF64 header.

Definition at line 138 of file `elf.h`.

11.51.2 Field Documentation

11.51.2.1 Elf64_Half Elf64_Ehdr::e_phnum

of entries in prog.

head. tab.

Definition at line 149 of file `elf.h`.

11.51.2.2 Elf64_Half Elf64_Ehdr::e_shnum

of entries in sect.

head. tab.

Definition at line 151 of file `elf.h`.

The documentation for this struct was generated from the following file:

- l4/util/elf.h

11.52 Elf64_Phdr Struct Reference

ELF64 program header.

```
#include <elf.h>
```

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

11.52.1 Detailed Description

ELF64 program header.

Definition at line 390 of file [elf.h](#).

The documentation for this struct was generated from the following file:

- l4/util/elf.h

11.53 Elf64_Shdr Struct Reference

ELF64 section header.

```
#include <elf.h>
```

Data Fields

- `Elf64_Word sh_name`
`name of sect (idx into shstrtab)`
- `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`

11.53.1 Detailed Description

ELF64 section header.

Definition at line 316 of file `elf.h`.

The documentation for this struct was generated from the following file:

- 14/util/elf.h

11.54 Elf64_Sym Struct Reference

ELF64 symbol table entry.

```
#include <elf.h>
```

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

11.54.1 Detailed Description

ELF64 symbol table entry.

Definition at line [769](#) of file [elf.h](#).

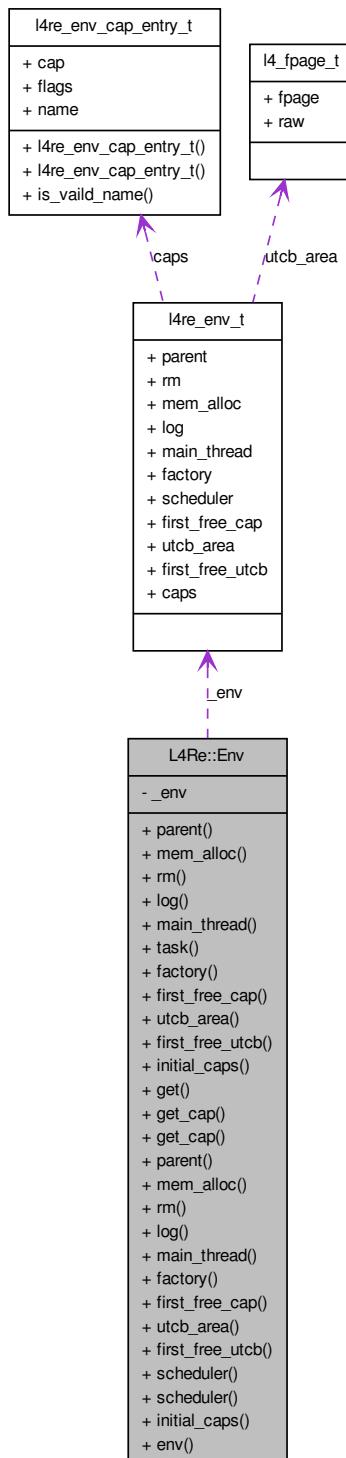
The documentation for this struct was generated from the following file:

- [l4/util/elf.h](#)

11.55 L4Re::Env Class Reference

Initial Environment (C++ version).

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 throw ()`
Object-capability to the parent.
- `L4::Cap< Mem_alloc > mem_alloc () const throw ()`
Object-capability to the memory allocator.
- `L4::Cap< Rm > rm () const throw ()`
Object-capability to the region map.
- `L4::Cap< Log > log () const throw ()`
Object-capability to the logging service.
- `L4::Cap< L4::Thread > main_thread () const throw ()`
Object-capability of the first user thread.
- `L4::Cap< L4::Task > task () const throw ()`
Object-capability of the user task.
- `L4::Cap< L4::Factory > factory () const throw ()`
Object-capability to the factory object available to the task.
- `l4_cap_idx_t first_free_cap () const throw ()`
First available capability selector.
- `l4_fpage_t utcb_area () const throw ()`
UTCB area of the task.
- `l4_addr_t first_free_utcb () const throw ()`
First free UTCB.
- `Cap_entry const * initial_caps () const throw ()`
Get a pointer to the first entry in the initial objects array.
- `Cap_entry const * get (char const *name, unsigned l) const throw ()`
Get the Cap_entry for the object named name.
- `template<typename T > L4::Cap< T > get_cap (char const *name, unsigned l) const throw ()`
Get the capability selector for the object named name.
- `template<typename T > L4::Cap< T > get_cap (char const *name) const throw ()`

Get the capability selector for the object named name.

- void `parent (L4::Cap< Parent > const &c) throw ()`
Set parent object-capability.
- void `mem_alloc (L4::Cap< Mem_alloc > const &c) throw ()`
Set memory allocator object-capability.
- void `rm (L4::Cap< Rm > const &c) throw ()`
Set region map object-capability.
- void `log (L4::Cap< Log > const &c) throw ()`
Set log object-capability.
- void `main_thread (L4::Cap< L4::Thread > const &c) throw ()`
Set object-capability of first user thread.
- void `factory (L4::Cap< L4::Factory > const &c) throw ()`
Set factory object-capability.
- void `first_free_cap (l4_cap_idx_t c) throw ()`
Set first available capability selector.
- void `utcbs_area (l4_fpage_t utcbs) throw ()`
Set UTCB area of the task.
- void `first_free_utcb (l4_addr_t u) throw ()`
Set first free UTCB.
- `L4::Cap< L4::Scheduler > scheduler () const throw ()`
Get the scheduler capability for the task.
- void `scheduler (L4::Cap< L4::Scheduler > const &c) throw ()`
Set the scheduler capability.
- void `initial_caps (Cap_entry *first) throw ()`
Set the pointer to the first Cap_entry in the initial objects array.

Static Public Member Functions

- static `Env const * env () throw ()`
Returns the initial environment for the current task.

11.55.1 Detailed Description

Initial Environment (C++ version). This class provides an initial set of capabilities as well as information the first free UTCB and used capability slots.

See also

[Initial environment](#)

Definition at line [85](#) of file [env](#).

11.55.2 Member Function Documentation

11.55.2.1 static Env const* L4Re::Env::env() throw() [inline, static]

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/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 [103](#) of file [env](#).

11.55.2.2 L4::Cap<Parent> L4Re::Env::parent() const throw() [inline]

Object-capability to the parent.

Returns

`Parent` object-capability

Definition at line [110](#) of file [env](#).

11.55.2.3 L4::Cap<Mem_alloc> L4Re::Env::mem_alloc() const throw() [inline]

Object-capability to the memory allocator.

Returns

Memory allocator object-capability

Definition at line [116](#) of file [env](#).

11.55.2.4 L4::Cap<Rm> L4Re::Env::rm() const throw() [inline]

Object-capability to the region map.

Returns

Region map object-capability

Definition at line 122 of file [env](#).

11.55.2.5 L4::Cap<Log> L4Re::Env::log() const throw() [inline]

Object-capability to the logging service.

Returns

[Log](#) object-capability

Definition at line 128 of file [env](#).

11.55.2.6 L4::Cap<L4::Thread> L4Re::Env::main_thread() const throw() [inline]

Object-capability of the first user thread.

Returns

Object-capability of the first user thread.

Definition at line 134 of file [env](#).

11.55.2.7 L4::Cap<L4::Task> L4Re::Env::task() const throw() [inline]

Object-capability of the user task.

Returns

Object-capability of the user task.

Definition at line 140 of file [env](#).

11.55.2.8 L4::Cap<L4::Factory> L4Re::Env::factory() const throw() [inline]

Object-capability to the factory object available to the task.

Returns

Factory object-capability

Definition at line 146 of file [env](#).

11.55.2.9 l4_cap_idx_t L4Re::Env::first_free_cap () const throw () [inline]

First available capability selector.

Returns

First capability selector.

First capability selector available for use for in the application.

Definition at line 154 of file [env](#).

11.55.2.10 l4_fpage_t L4Re::Env::utcb_area () const throw () [inline]

UTCB area of the task.

Returns

UTCB area

Definition at line 160 of file [env](#).

11.55.2.11 l4_addr_t L4Re::Env::first_free_utcb () const throw () [inline]

First free UTCB.

Returns

object-capability

First free UTCB within the UTCB area available for the application to use.

Definition at line 169 of file [env](#).

11.55.2.12 Cap_entry const* L4Re::Env::initial_caps () const throw () [inline]

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 176 of file [env](#).

11.55.2.13 Cap_entry const* L4Re::Env::get (char const * *name*, unsigned *l*) const throw () [inline]

Get the Cap_entry for the object named *name*.

Parameters

name is the name of the object.

l is the length of the name, thus *name* might not be zero terminated.

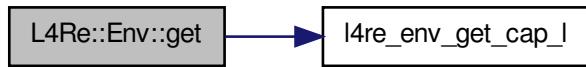
Returns

A pointer to the Cap_entry for the object named *name*, or NULL if no such object was found.

Definition at line 187 of file [env](#).

References [l4re_env_get_cap_l\(\)](#).

Here is the call graph for this function:



11.55.2.14 template<typename T > L4::Cap<T> L4Re::Env::get_cap (char const * name, unsigned l) const throw () [inline]

Get the capability selector for the object named *name*.

Parameters

name is the name of the object.

l is the length of the name, thus *name* 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.

Definition at line 199 of file [env](#).

11.55.2.15 template<typename T > L4::Cap<T> L4Re::Env::get_cap (char const * name) const throw () [inline]

Get the capability selector for the object named *name*.

Parameters

name 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 214 of file [env](#).

11.55.2.16 void L4Re::Env::parent (L4::Cap< Parent > const & *c*) throw () [inline]

Set parent object-capability.

Parameters

c Parent object-capability

Definition at line 221 of file [env](#).

11.55.2.17 void L4Re::Env::mem_alloc (L4::Cap< Mem_alloc > const & *c*) throw () [inline]

Set memory allocator object-capability.

Parameters

c Memory allocator object-capability

Definition at line 227 of file [env](#).

11.55.2.18 void L4Re::Env::rm (L4::Cap< Rm > const & *c*) throw () [inline]

Set region map object-capability.

Parameters

c Region map object-capability

Definition at line 233 of file [env](#).

11.55.2.19 void L4Re::Env::log (L4::Cap< Log > const & *c*) throw () [inline]

Set log object-capability.

Parameters

c Log object-capability

Definition at line 239 of file [env](#).

11.55.2.20 void L4Re::Env::main_thread (L4::Cap< L4::Thread > const & *c*) throw () [inline]

Set object-capability of first user thread.

Parameters

c First thread's object-capability

Definition at line 245 of file [env](#).

11.55.2.21 void L4Re::Env::factory (L4::Cap< L4::Factory > const & *c*) throw () [inline]

Set factory object-capability.

Parameters

c Factory object-capability

Definition at line 251 of file [env](#).

11.55.2.22 void L4Re::Env::first_free_cap (l4_cap_idx_t *c*) throw () [inline]

Set first available capability selector.

Parameters

c First capability selector available to the application.

Definition at line 257 of file [env](#).

11.55.2.23 void L4Re::Env::utcb_area (l4_fpage_t *utcbs*) throw () [inline]

Set UTCB area of the task.

Parameters

utcbs UTCB area

Definition at line 263 of file [env](#).

11.55.2.24 void L4Re::Env::first_free_utcb (l4_addr_t *u*) throw () [inline]

Set first free UTCB.

Parameters

u First UTCB available for the application to use.

Definition at line 269 of file [env](#).

11.55.2.25 L4::Cap<L4::Scheduler> L4Re::Env::scheduler () const throw () [inline]

Get the scheduler capability for the task.

Returns

The capability selector for the default scheduler used for this task.

Definition at line 277 of file [env](#).

11.55.2.26 void L4Re::Env::scheduler (L4::Cap< L4::Scheduler > const & *c*) throw () [inline]

Set the scheduler capability.

Parameters

c is the capability to be set as scheduler.

Definition at line 284 of file [env](#).

11.55.2.27 void L4Re::Env::initial_caps (Cap_entry * *first*) throw () [inline]

Set the pointer to the first Cap_entry in the initial objects array.

Parameters

first is the first element in the array.

Definition at line 292 of file [env](#).

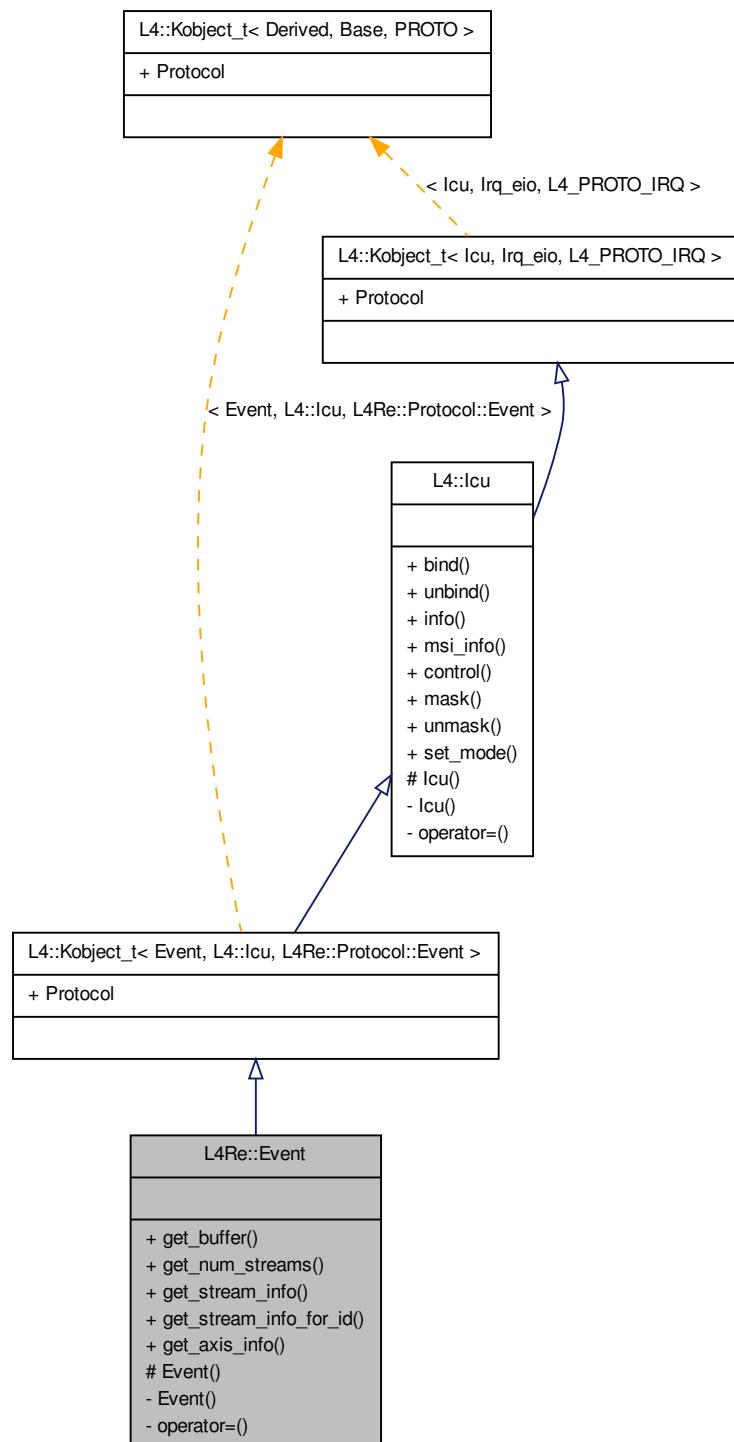
The documentation for this class was generated from the following file:

- l4/re/env

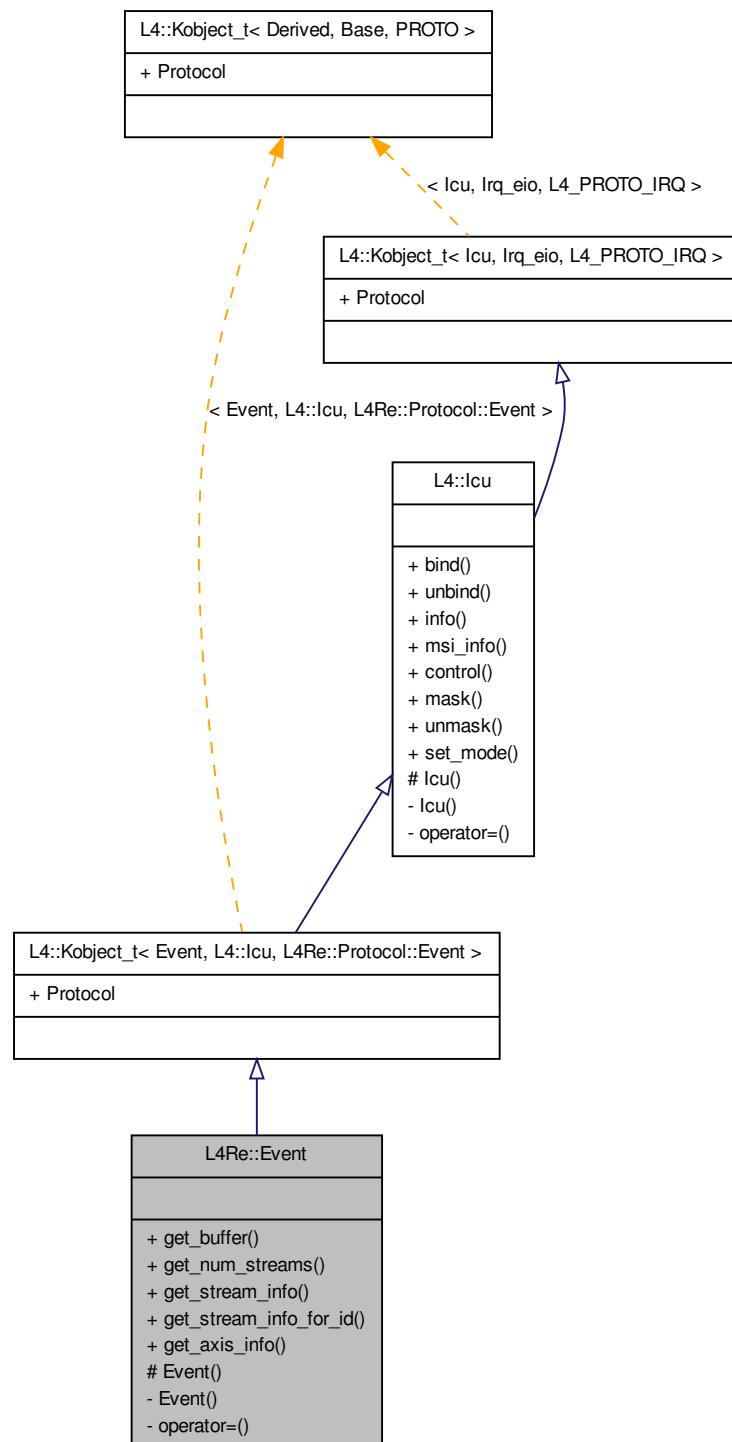
11.56 L4Re::Event Class Reference

[Event](#) class.

Inheritance diagram for L4Re::Event:



Collaboration diagram for L4Re::Event:



Public Member Functions

- long `get_buffer (L4::Cap< Dataspace > ds) const throw ()`

Get event signal buffer.

11.56.1 Detailed Description

[Event](#) class.

Definition at line 103 of file [event](#).

11.56.2 Member Function Documentation

11.56.2.1 long L4Re::Event::get_buffer (L4::Cap< Dataspace > ds) const throw ()

Get event signal buffer.

Return values

ds [Event](#) buffer.

Returns

0 on success, negative error code otherwise.

The documentation for this class was generated from the following file:

- [l4/re/event](#)

11.57 L4Re::Event_buffer_t< PAYLOAD >::Event Struct Reference

[Event](#) structure used in buffer.

Public Member Functions

- void `free () throw ()`

Free the entry.

Data Fields

- long long `time`

Event time stamp.

11.57.1 Detailed Description

```
template<typename PAYLOAD = Default_event_payload> struct L4Re::Event_buffer_t< PAYLOAD >::Event
```

[Event](#) structure used in buffer.

Definition at line 144 of file [event](#).

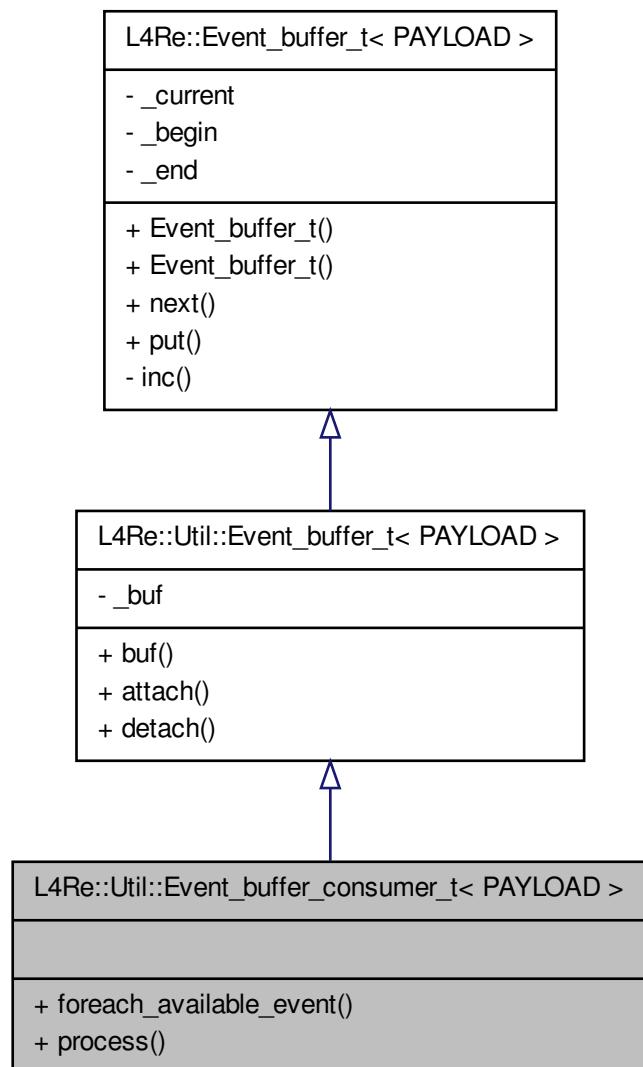
The documentation for this struct was generated from the following file:

- [l4/re/event](#)

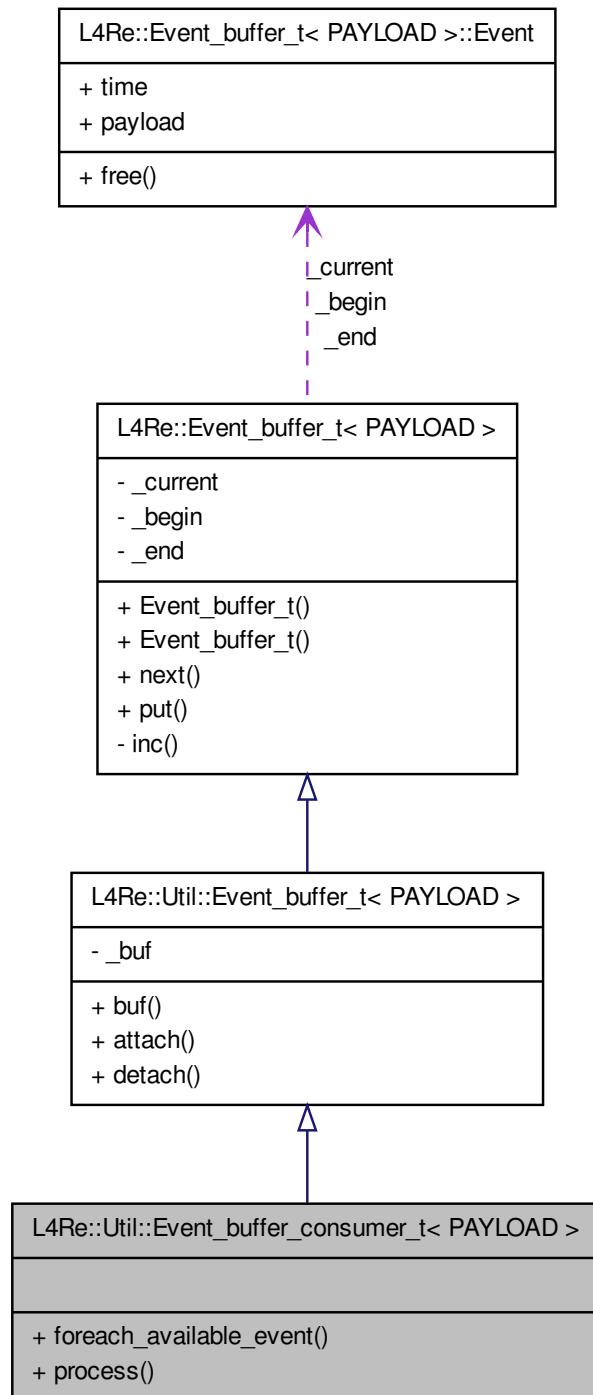
11.58 L4Re::Util::Event_buffer_consumer_t< PAYLOAD > Class Template Reference

An event buffer consumer.

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.

11.58.1 Detailed Description

template<typename PAYLOAD> class L4Re::Util::Event_buffer_consumer_t< PAYLOAD >

An event buffer consumer.

Definition at line 90 of file [event_buffer](#).

11.58.2 Member Function Documentation

11.58.2.1 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

cb Function callback.

Definition at line 100 of file [event_buffer](#).

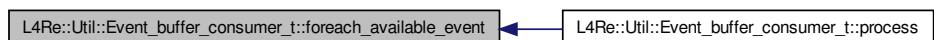
References [L4Re::Event_buffer_t< PAYLOAD >::Event::free\(\)](#).

Referenced by [L4Re::Util::Event_buffer_consumer_t< PAYLOAD >::process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.58.2.2 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

irq [Event](#) signal to wait for.

thread Thread capability of the thread calling this function.

cb Callback function that is called for each received event.

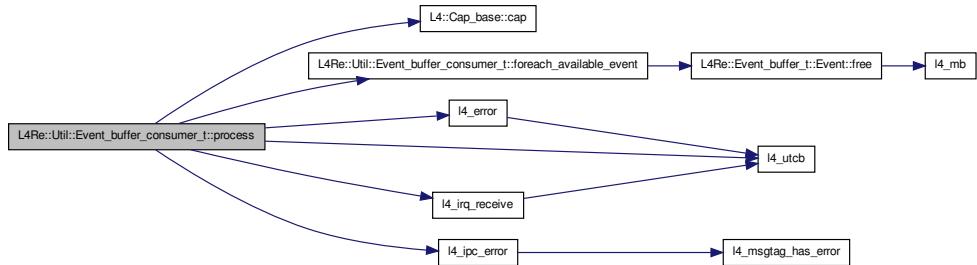
Note

This function never returns.

Definition at line 120 of file [event_buffer](#).

References [L4::Cap_base::cap\(\)](#), [L4Re::Util::Event_buffer_consumer_t< PAYLOAD >::foreach_available_event\(\)](#), [l4_error\(\)](#), [l4_ipc_error\(\)](#), [l4_irq_receive\(\)](#), and [l4_utcb\(\)](#).

Here is the call graph for this function:



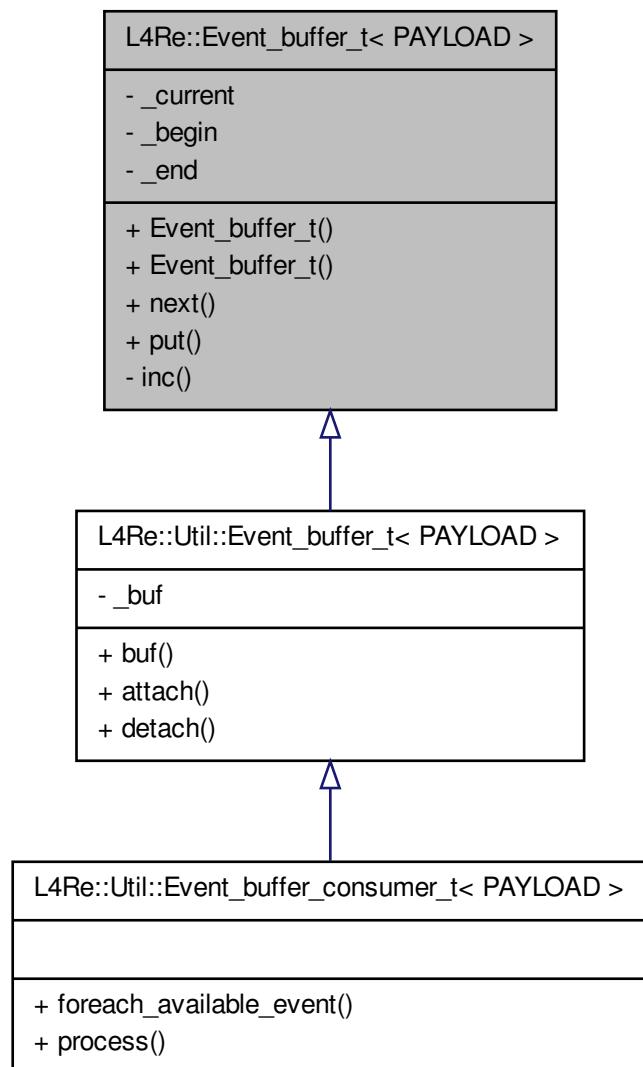
The documentation for this class was generated from the following file:

- [l4/re/util/event_buffer](#)

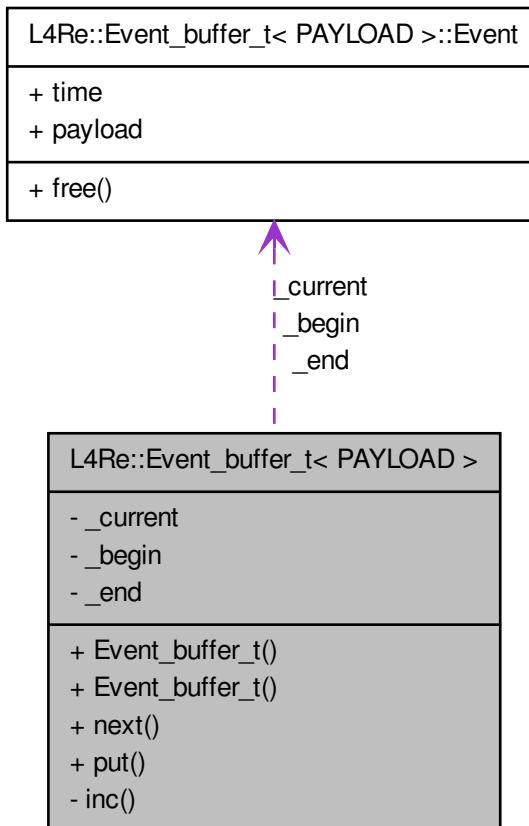
11.59 L4Re::Event_buffer_t< PAYLOAD > Class Template Reference

[Event](#) buffer class.

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 \(\) throw \(\)](#)
Next event in buffer.
- [bool put \(Event const &ev\) throw \(\)](#)

Put event into buffer at current position.

11.59.1 Detailed Description

template<typename PAYLOAD = Default_event_payload> class L4Re::Event_buffer_t< PAYLOAD >

[Event](#) buffer class.

Definition at line 137 of file [event](#).

11.59.2 Constructor & Destructor Documentation

11.59.2.1 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

buffer Pointer to buffer.

size Size of buffer in bytes.

Definition at line 177 of file [event](#).

11.59.3 Member Function Documentation

11.59.3.1 template<typename PAYLOAD = Default_event_payload> Event* L4Re::Event_buffer_t< PAYLOAD >::next () throw () [inline]

Next event in buffer.

Returns

0 if no event available, event otherwise.

Definition at line 187 of file [event](#).

References [L4Re::Event_buffer_t< PAYLOAD >::Event::time](#).

11.59.3.2 template<typename PAYLOAD = Default_event_payload> bool L4Re::Event_buffer_t< PAYLOAD >::put (Event const & *ev*) throw () [inline]

Put event into buffer at current position.

Parameters

ev [Event](#) to put into the buffer.

Returns

false if buffer is full and entry could not be added.

Definition at line 204 of file [event](#).

References [l4_wmb\(\)](#), and [L4Re::Event_buffer_t< PAYLOAD >::Event::time](#).

Here is the call graph for this function:



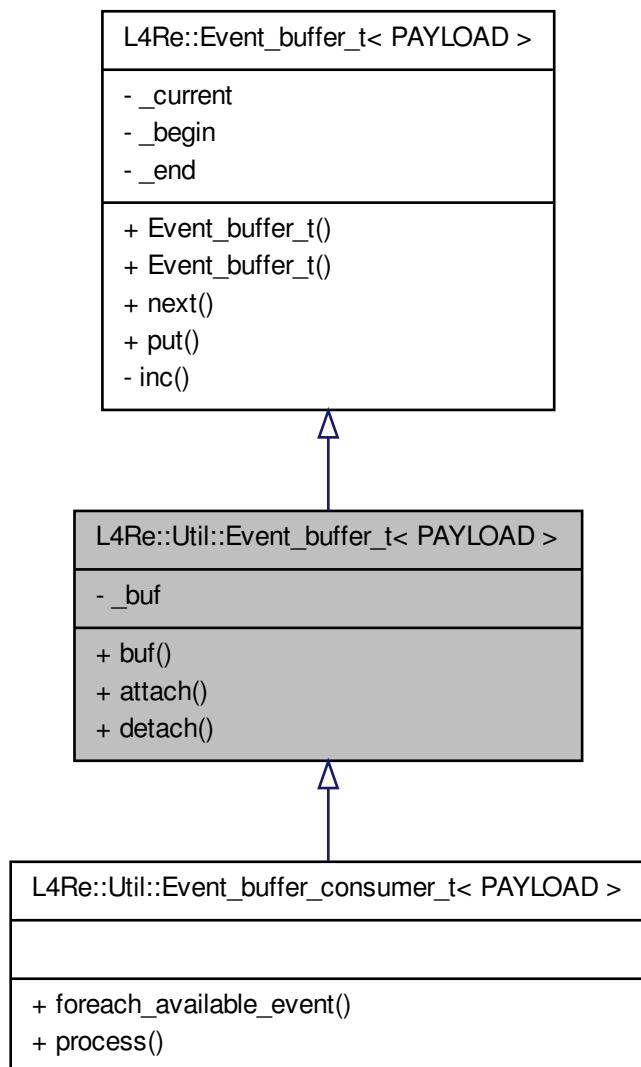
The documentation for this class was generated from the following file:

- [l4/re/event](#)

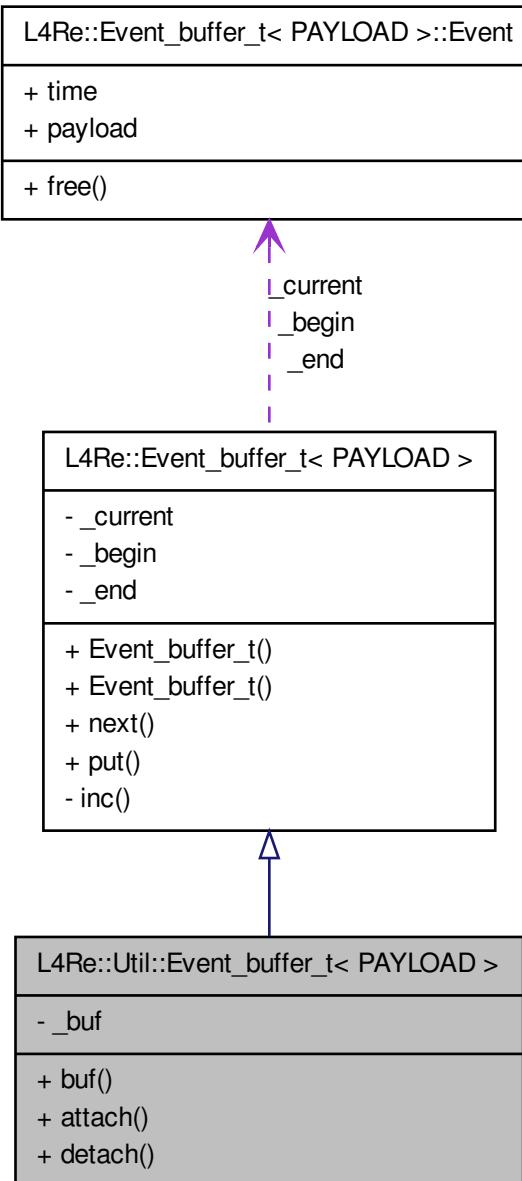
11.60 L4Re::Util::Event_buffer_t< PAYLOAD > Class Template Reference

Event_buffer utility class.

Inheritance diagram for L4Re::Util::Event_buffer_t< PAYLOAD >:



Collaboration diagram for L4Re::Util::Event_buffer_t< PAYLOAD >:



Public Member Functions

- `void * buf() const throw()`

Return the buffer.

- long `attach (L4::Cap< L4Re::Dataspace > ds, L4::Cap< L4Re::Rm > rm) throw ()`
Attach event buffer from address space.
- long `detach (L4::Cap< L4Re::Rm > rm, void *buf) throw ()`
Detach event buffer from address space.

11.60.1 Detailed Description

`template<typename PAYLOAD> class L4Re::Util::Event_buffer_t< PAYLOAD >`

Event_buffer utility class.

Definition at line 36 of file [event_buffer](#).

11.60.2 Member Function Documentation

11.60.2.1 `template<typename PAYLOAD > void* L4Re::Util::Event_buffer_t< PAYLOAD >::buf () const throw () [inline]`

Return the buffer.

Returns

Pointer to the event buffer.

Definition at line 46 of file [event_buffer](#).

Referenced by [L4Re::Util::Event_buffer_t< PAYLOAD >::detach\(\)](#).

Here is the caller graph for this function:



11.60.2.2 `template<typename PAYLOAD > long L4Re::Util::Event_buffer_t< PAYLOAD >::attach (L4::Cap< L4Re::Dataspace > ds, L4::Cap< L4Re::Rm > rm) throw () [inline]`

Attach event buffer from address space.

Parameters

ds `Dataspace` of the event buffer.

rm Region manager to attach buffer to.

Returns

0 on success, negative error code otherwise.

Definition at line 56 of file [event_buffer](#).

References [L4Re::Rm::Search_addr](#).

11.60.2.3 template<typename PAYLOAD> long L4Re::Util::Event_buffer_t< PAYLOAD >::detach (L4::Cap< L4Re::Rm > rm, void * buf) throw () [inline]

Detach event buffer from address space.

Parameters

rm Region manager to detach buffer from.

buf Buffer pointer.

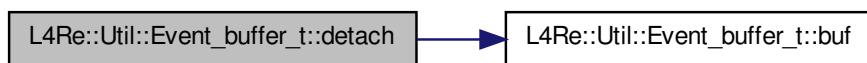
Returns

0 on success, negative error code otherwise.

Definition at line 77 of file [event_buffer](#).

References [L4Re::Util::Event_buffer_t< PAYLOAD >::buf\(\)](#).

Here is the call graph for this function:



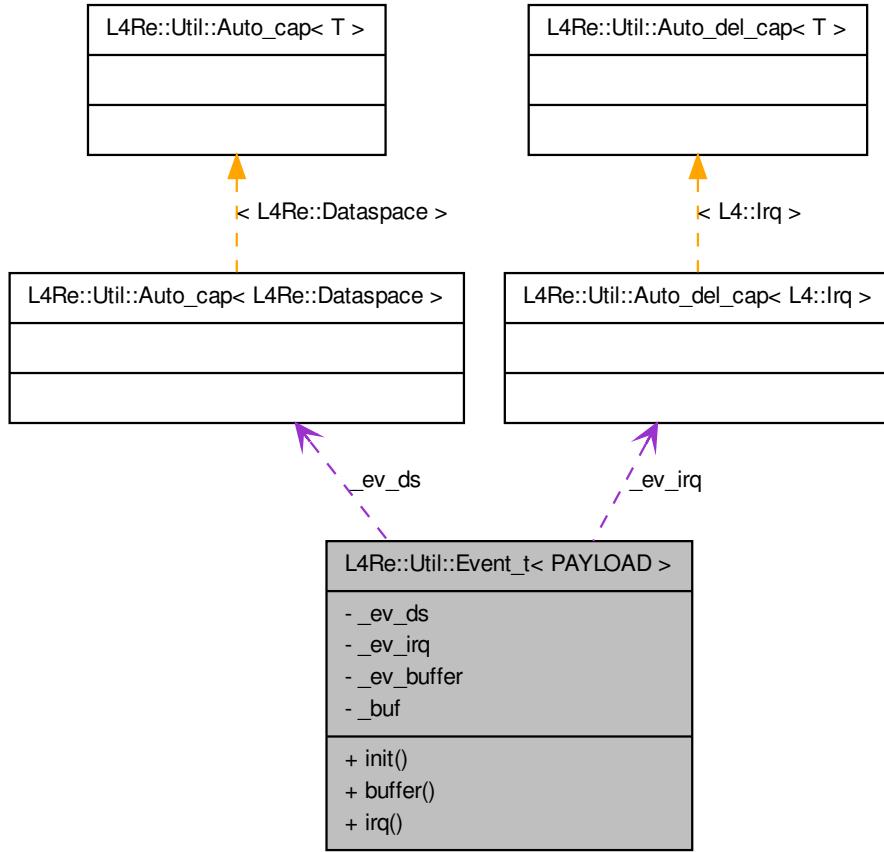
The documentation for this class was generated from the following file:

- [l4/re/util/event_buffer](#)

11.61 L4Re::Util::Event_t< PAYLOAD > Class Template Reference

Convenience wrapper for getting access to an event object.

Collaboration diagram for L4Re::Util::Event_t< PAYLOAD >:



Public Types

- enum `Mode` { `Mode_irq`, `Mode_polling` }
- Modes of operation.*

Public Member Functions

- int `init` (`L4::Cap< L4Re::Event >` event, `Mode` mode=`Mode_irq`, `L4Re::Env` const *env=`L4Re::Env::env()`, `L4Re::Cap_alloc` *ca=`L4Re::Cap_alloc::get_cap_alloc(L4Re::Util::cap_alloc())`)
Initialise an event object.
- `L4Re::Event_buffer_t< PAYLOAD > & buffer()`
Get event buffer.

- `L4::Cap< L4::Irq > irq () const`

Get event IRQ.

11.61.1 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 39 of file [event](#).

11.61.2 Member Enumeration Documentation

11.61.2.1 `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 35 of file [event](#).

11.61.3 Member Function Documentation

11.61.3.1 `template<typename PAYLOAD> int L4Re::Util::Event_t< PAYLOAD >::init (L4::Cap< L4Re::Event > event, Mode mode = Mode_irq, L4Re::Env const * env = L4Re::Env::env(), L4Re::Cap_alloc * ca = L4Re::Cap_alloc::get_cap_alloc(L4Re::Util::cap_alloc)) [inline]`

Initialise an event object.

Parameters

`event` Capability to event.

`env` Optional: Pointer to L4Re-Environment

`ca` Optional: Pointer to capability allocator.

Returns

0 on success, error code on error

Definition at line 49 of file [event](#).

11.61.3.2 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 [98](#) of file [event](#).

11.61.3.3 template<typename PAYLOAD > L4::Cap<L4::Irq> L4Re::Util::Event_t< PAYLOAD >::irq() const [inline]

Get event IRQ.

Returns

[Event](#) IRQ.

Definition at line [104](#) of file [event](#).

The documentation for this class was generated from the following file:

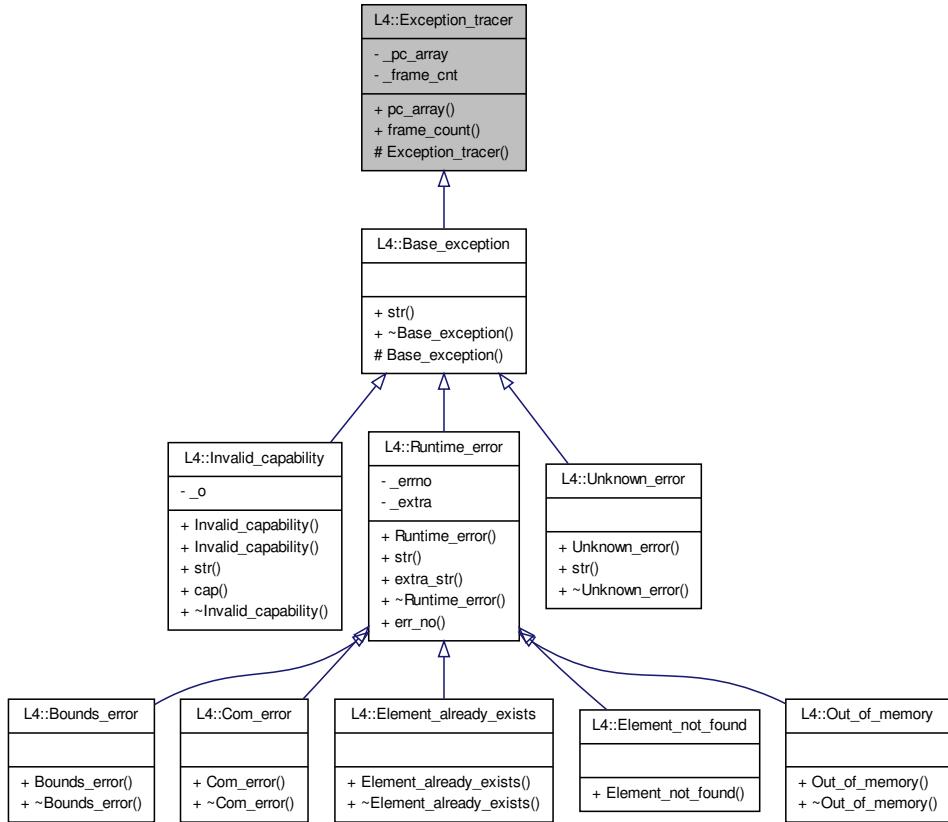
- [l4/re/util/event](#)

11.62 L4::Exception_tracer Class Reference

Back-trace support for exceptions.

```
#include <l4/cxx/exceptions>
```

Inheritance diagram for L4::Exception_tracer:



Public Member Functions

- `void const *const *pc_array () const throw ()`
Get the array containing the call trace.
- `int frame_count () const throw ()`
Get the number of entries that are valid in the call trace.

Protected Member Functions

- `Exception_tracer () throw ()`

Create a back trace.

11.62.1 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 [60](#) of file [exceptions](#).

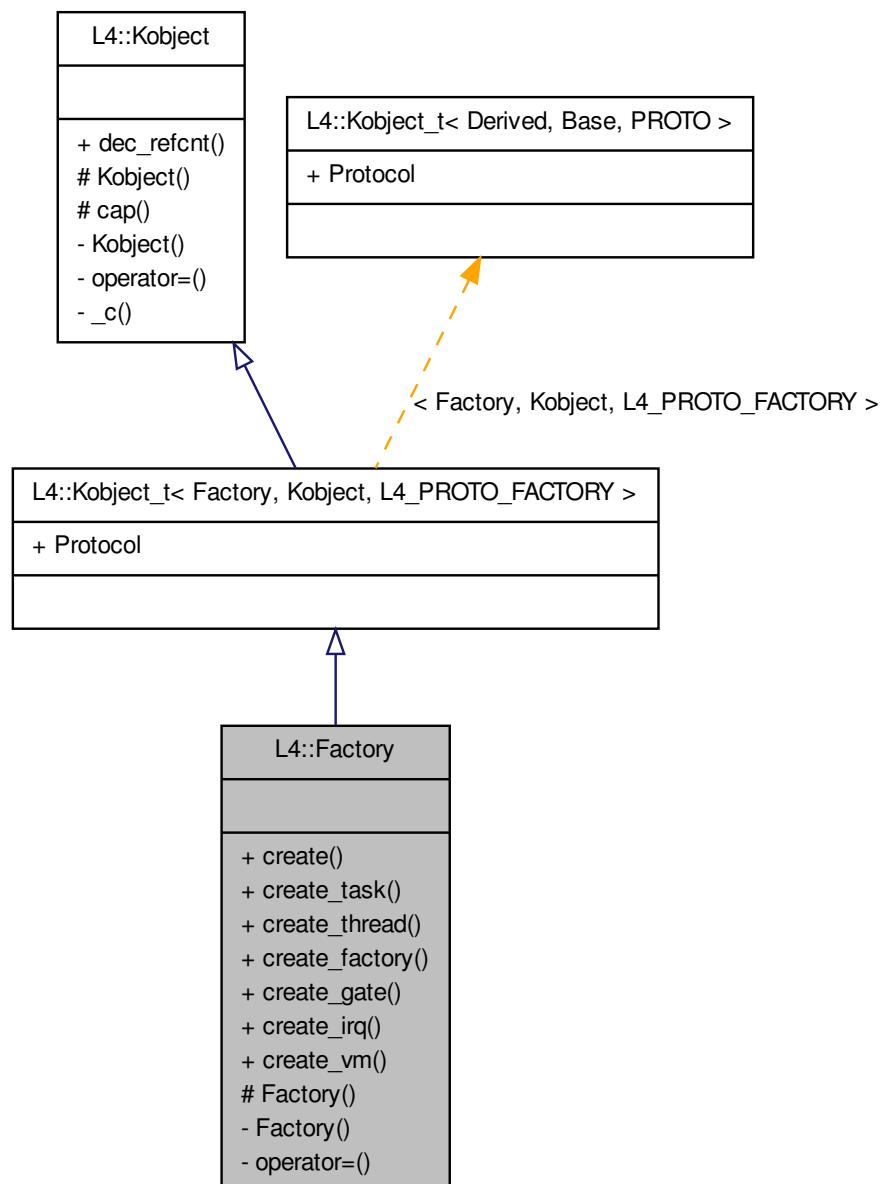
The documentation for this class was generated from the following file:

- [l4/cxx/exceptions](#)

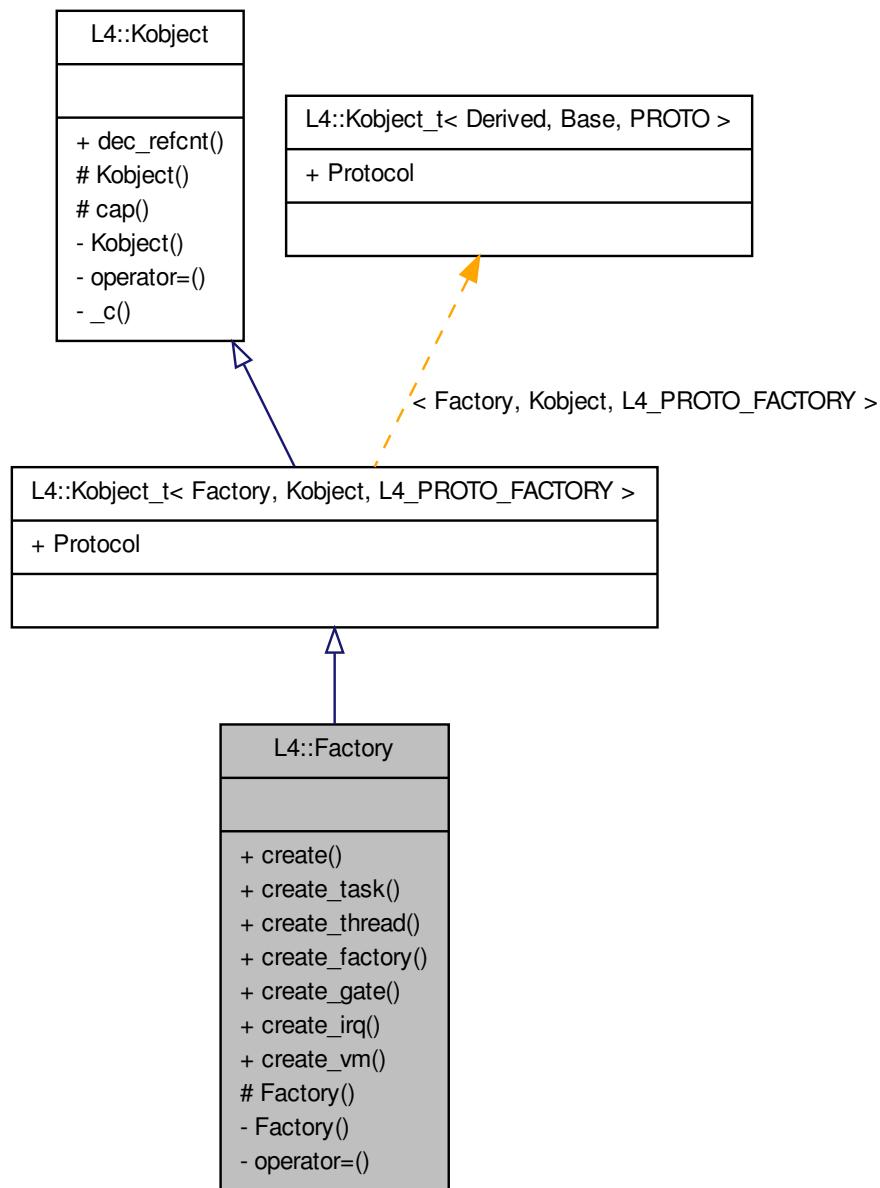
11.63 L4::Factory Class Reference

C++ [L4 Factory](#), to create all kinds of kernel objects.

Inheritance diagram for L4::Factory:



Collaboration diagram for L4::Factory:



Data Structures

- struct [Lstr](#)
Special type to add a pascal string into the factory create stream.
- struct [Nil](#)

Special type to add a void argument into the factory create stream.

- class **S**

Stream class for the [create\(\)](#) argument stream.

Public Member Functions

- **S create (Cap< Kobject > target, long obj, l4_utcb_t *utcb=l4_utcb()) throw ()**
Generic create call to the factory.
- **l4_mshtag_t create_task (Cap< Task > const &target_cap, l4_fpage_t const &utcb_area, l4_utcb_t *utcb=l4_utcb()) throw ()**
- **l4_mshtag_t create_thread (Cap< Thread > const &target_cap, l4_utcb_t *utcb=l4_utcb()) throw ()**
- **l4_mshtag_t create_factory (Cap< Factory > const &target_cap, unsigned long limit, l4_utcb_t *utcb=l4_utcb()) throw ()**
- **l4_mshtag_t create_gate (Cap< Kobject > const &target_cap, Cap< Thread > const &thread_cap, l4_umword_t label, l4_utcb_t *utcb=l4_utcb()) throw ()**
- **l4_mshtag_t create_irq (Cap< Irq >const &target_cap, l4_utcb_t *utcb=l4_utcb()) throw ()**
- **l4_mshtag_t create_vm (Cap< Vm >const &target_cap, l4_utcb_t *utcb=l4_utcb()) throw ()**

11.63.1 Detailed Description

C++ [L4 Factory](#), to create all kinds of kernel objects. `#include <l4/sys/factory>`

See also

[Factory](#) for an overview and C bindings.

Definition at line 41 of file [factory](#).

11.63.2 Member Function Documentation

11.63.2.1 S L4::Factory::create (Cap< Kobject > target, long obj, l4_utcb_t * utcb = **l4_utcb()**) throw () [inline]

Generic create call to the factory.

Parameters

target is the target capability selector where the new object shall be received.

obj is the protocol ID that specifies which kind of object shall be created.

utcb is the UTCB to use for the operation.

Returns

a create stream that allows adding additional arguments to the [create\(\)](#) call.

This method does currently not directly invoke the factory. It returns a stream that shall invoke the factory after adding all additional arguments.

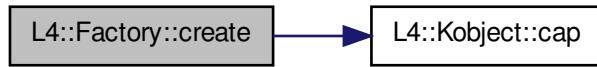
Usage:

```
L4::Cap<L4Re::Namespace> ns = L4Re::Util::cap_alloc.alloc<L4Re::Namespace>();
factory->create(ns, L4Re::Namespace::Protocol) << "Argument text";
```

Definition at line 213 of file [factory](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.63.2.2 l4_mshtag_t L4::Factory::create_task (Cap< Task > const & target_cap, l4_fpage_t const & utcb_area, l4_utcb_t * utcb = l4_utcb()) throw () [inline]

Create a new task.

Parameters

factory Capability selector for factory to use for creation.

target_cap Capability selector for the root capability of the new task.

utcb_area Flexpage that describes the area for the UTCBs of the new task

Note

The size of the UTCB area specifies indirectly the maximum number of UTCBs available for this task and cannot be changed afterwards.

Returns

Syscall return tag

See also

[Task](#)

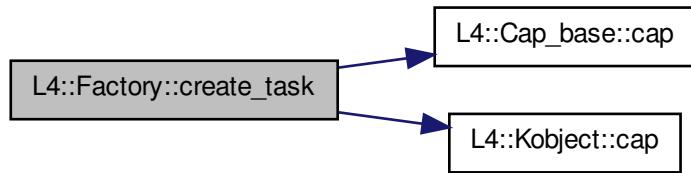
Note

factory is the implicit *this* pointer.

Definition at line 222 of file [factory](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.63.2.3 `l4_mshtag_t L4::Factory::create_thread (Cap< Thread > const & target_cap, l4_utcb_t * utcb = l4_utcb ()) throw () [inline]`

Create a new thread.

Parameters

factory Capability selector for factory to use for creation.
target_cap Capability selector for the root capability of the new thread.

Returns

Syscall return tag

See also

[Thread](#)

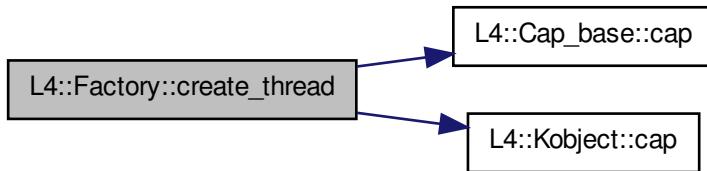
Note

factory is the implicit *this* pointer.

Definition at line 231 of file [factory](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



**11.63.2.4 `l4_mshtag_t L4::Factory::create_factory (Cap< Factory > const & target_cap,
unsigned long limit, l4_utcb_t * utcb = l4_utcb()) throw () [inline]`**

Create a new factory.

Parameters

factory Capability selector for factory to use for creation.

target_cap Capability selector for the root capability of the new factory.

limit Limit for the new factory in bytes

Note

The limit of the new factory is subtracted from the available amount of the factory used for creation.

Returns

Syscall return tag

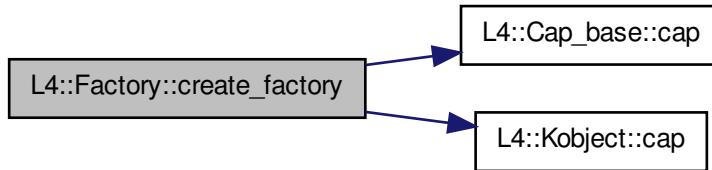
Note

factory is the implicit *this* pointer.

Definition at line 239 of file [factory](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



**11.63.2.5 `l4_mshtag_t L4::Factory::create_gate (Cap< Kobject > const & target_cap, Cap<
Thread > const & thread_cap, l4_umword_t label, l4_utcb_t * utcb = l4_utcb())
throw () [inline]`**

Create a new IPC gate.

Parameters

factory Capability selector for factory to use for creation.

target_cap Capability selector for the root capability of the new IPC gate.

thread_cap Thread to bind the gate to

label Label of the gate

Returns

Syscall return tag

See also

[IPC-Gate API](#)

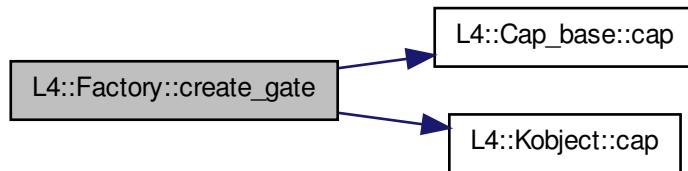
Note

factory is the implicit *this* pointer.

Definition at line 248 of file [factory](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.63.2.6 `l4_mshtag_t L4::Factory::create_irq (Cap< Irq >const & target_cap, l4_utcb_t * utcb = l4_utcb()) throw () [inline]`

Create a new IRQ.

Parameters

factory Capability selector for factory to use for creation.

target_cap Capability selector for the root capability of the new IRQ.

Returns

Syscall return tag

See also

[IRQs](#)

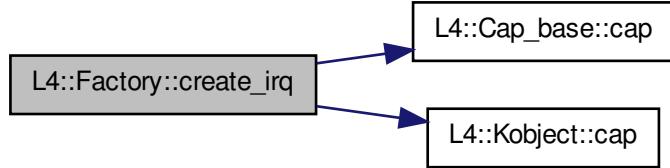
Note

factory is the implicit *this* pointer.

Definition at line 257 of file [factory](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.63.2.7 `l4_mshtag_t L4::Factory::create_vm (Cap< Vm >const & target_cap, l4_utcb_t * utcb = l4_utcb()) throw () [inline]`

Create a new virtual machine.

Parameters

factory Capability selector for factory to use for creation.

target_cap Capability selector for the root capability of the new VM.

Returns

Syscall return tag

See also

[Virtual Machines](#)

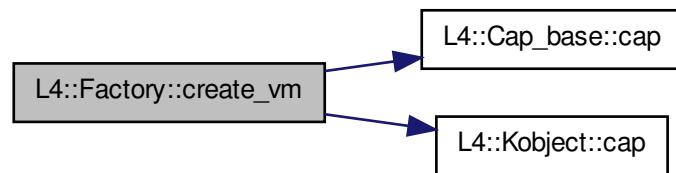
Note

factory is the implicit *this* pointer.

Definition at line 265 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:

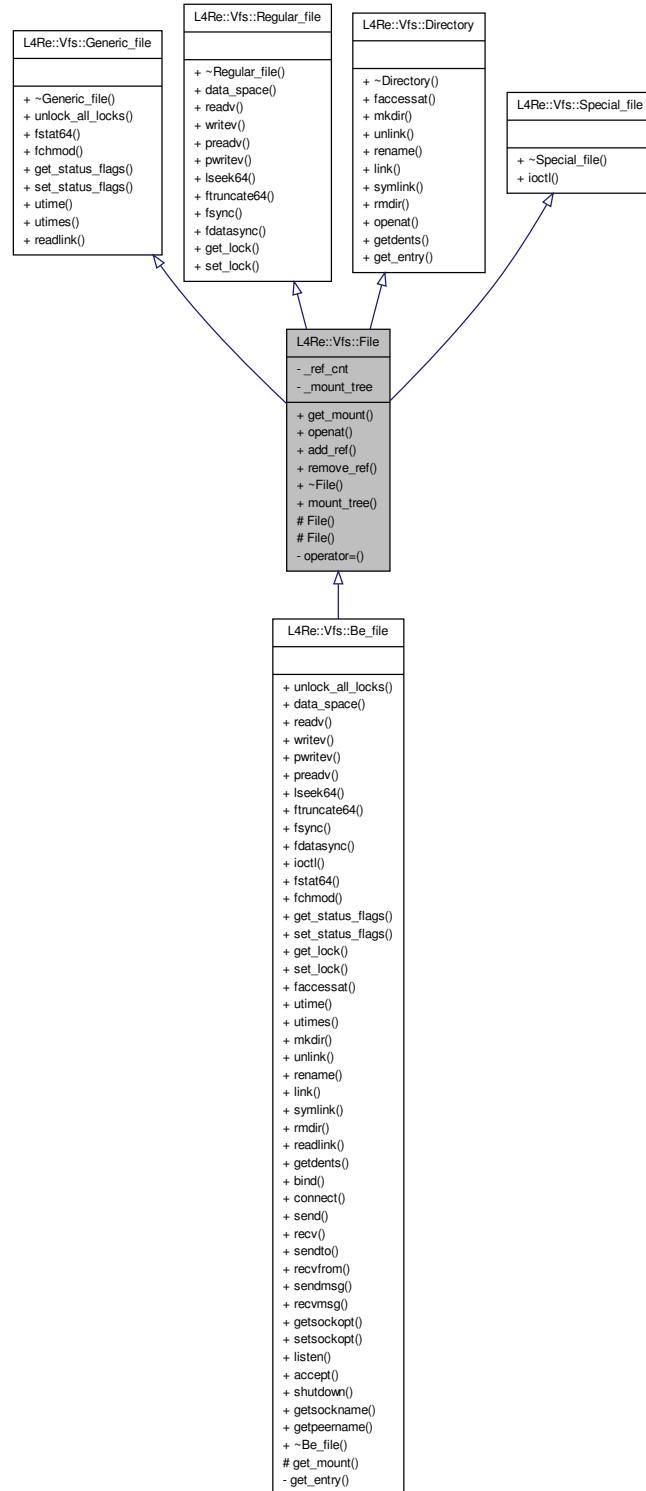
- l4/sys/factory

11.64 L4Re::Vfs::File Class Reference

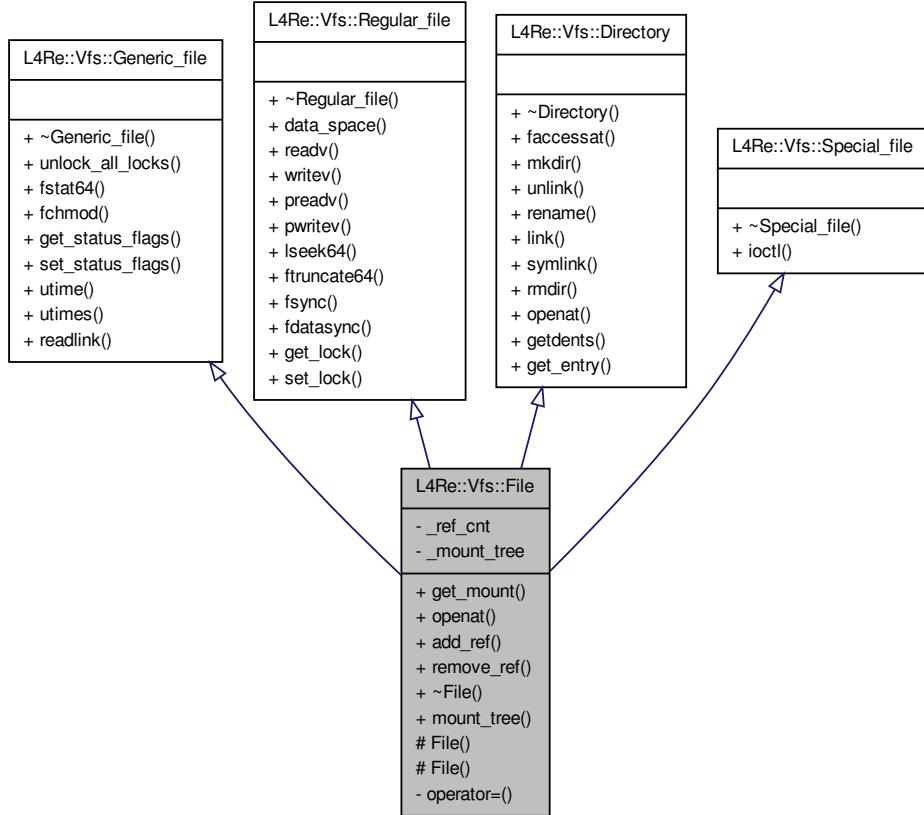
The basic interface for an open POSIX file.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::File:



Collaboration diagram for L4Re::Vfs::File:



11.64.1 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 a 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 you may use [L4Re::Vfs::Be_file](#) as a base class.

Definition at line 430 of file [vfs.h](#).

The documentation for this class was generated from the following file:

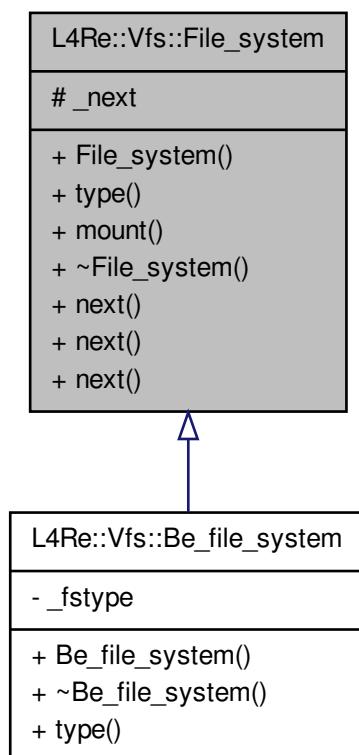
- [14/l4re_vfs/vfs.h](#)

11.65 L4Re::Vfs::File_system Class Reference

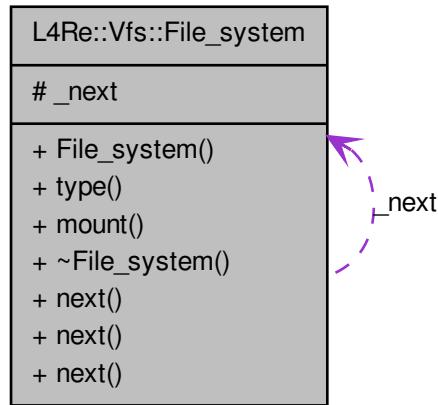
Basic interface for an [L4Re::Vfs](#) file system.

```
#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 =0 throw ()

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)=0 throw ()

Create a directory object dir representing source mounted with this file system.

11.65.1 Detailed Description

Basic interface for an [L4Re::Vfs](#) file system.

Note

For implementing a special file system you may use [L4Re::Vfs::Be_file_system](#) as a base class.

The main purpose of this interface is that there is a single object for each supported file-system type (e.g., ext2, vfat) exists in your application and is registered at the [L4Re::Vfs::Fs](#) singleton available in via [L4Re::Vfs::vfs_ops](#). At the end the POSIX mount function call the [File_system::mount](#) method for the given file-system type given in mount.

Definition at line 827 of file [vfs.h](#).

11.65.2 Member Function Documentation

11.65.2.1 `virtual char const* L4Re::Vfs::File_system::type () const throw () [pure virtual]`

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](#).

11.65.2.2 `virtual int L4Re::Vfs::File_system::mount (char const * source, unsigned long mountflags, void const * data, cxx::Ref_ptr<File> * dir) throw () [pure virtual]`

Create a directory object *dir* representing *source* mounted with this file system.

Parameters

source The path to the source device to mount. This may also be some URL or anything file-system specific.

mountflags The mount flags as specified in the POSIX mount call.

data The data as specified in the POSIX mount call. The contents are file-system specific.

Return values

dir A new directory object representing the file-system root directory.

Returns

0 on success, and <0 on error (e.g. -EINVAL).

Examples:

[tmpfs/lib/src/fs.cc](#).

Referenced by [L4Re::Vfs::Fs::mount\(\)](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

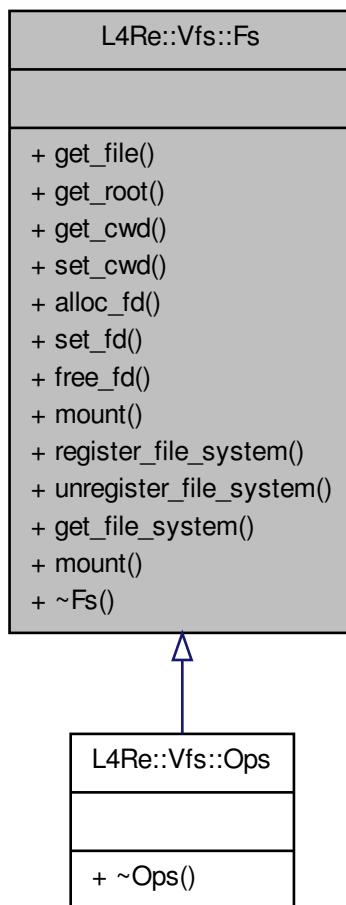
- [l4/re vfs/vfs.h](#)

11.66 L4Re::Vfs::Fs Class Reference

POSIX File-system related functionality.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::Fs:



Public Member Functions

- virtual cxx::Ref_ptr< [File](#) > `get_file` (int fd)=0 throw ()

Get the [L4Re::Vfs::File](#) for the file descriptor fd.
- virtual cxx::Ref_ptr< [File](#) > `get_root` ()=0 throw ()

Get the directory object for the applications root directory.

- virtual cxx::Ref_ptr<[File](#)> [get_cwd](#) () throw ()

Get the directory object for the applications current working directory.
- virtual void [set_cwd](#) (cxx::Ref_ptr<[File](#)> const &) throw ()

Set the current working directory for the application.
- virtual int [alloc_fd](#) (cxx::Ref_ptr<[File](#)> const &f=cxx::Ref_ptr<>::Nil)=0 throw ()

Allocate the next free file descriptor.
- virtual cxx::Ref_ptr<[File](#)> [set_fd](#) (int fd, cxx::Ref_ptr<[File](#)> const &f=cxx::Ref_ptr<>::Nil)=0 throw ()

Set the file object referenced by the file descriptor fd.
- virtual cxx::Ref_ptr<[File](#)> [free_fd](#) (int fd)=0 throw ()

Free the file descriptor fd.
- int [mount](#) (char const *path, cxx::Ref_ptr<[File](#)> const &dir) throw ()

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) throw ()

Backend for the POSIX mount call.

11.66.1 Detailed Description

POSIX File-system related functionality.

Note

This class usually exists as a singleton as a superclass of [L4Re::Vfs::Ops](#) (

See also

[L4Re::Vfs::vfs_ops](#)).

Definition at line 879 of file [vfs.h](#).

11.66.2 Member Function Documentation

11.66.2.1 virtual cxx::Ref_ptr<[File](#)> [L4Re::Vfs::Fs::get_file](#) (int *fd*) throw () [pure virtual]

Get the [L4Re::Vfs::File](#) for the file descriptor *fd*.

Parameters

fd The POSIX file descriptor number.

Returns

A pointer to the [File](#) object, or 0 if *fd* is not open.

11.66.2.2 virtual int L4Re::Vfs::Fs::alloc_fd (*cxx::Ref_ptr< File > const & f = cxx::Ref_ptr<>::Nil*) throw () [pure virtual]

Allocate the next free file descriptor.

Parameters

f The file to assign to that file descriptor.

Returns

the allocated file descriptor, or -EMFILE on error.

11.66.2.3 virtual cxx::Ref_ptr<File> L4Re::Vfs::Fs::set_fd (int *fd*, *cxx::Ref_ptr< File > const & f = cxx::Ref_ptr<>::Nil*) throw () [pure virtual]

Set the file object referenced by the file descriptor *fd*.

Parameters

fd The file descriptor to set to *f*;

f The file object to assign.

Returns

A pointer to the file object that was previously assigned to fd.

11.66.2.4 virtual cxx::Ref_ptr<File> L4Re::Vfs::Fs::free_fd (int *fd*) throw () [pure virtual]

Free the file descriptor *fd*.

Parameters

fd The file descriptor to free.

Returns

A pointer to the file object that was assigned to the fd.

11.66.2.5 int L4Re::Vfs::Fs::mount (char const * *path*, *cxx::Ref_ptr< File > const & dir*) throw () [inline]

Mount a given file object at the given global path in the VFS.

Parameters

path The global path to mount *dir* at.

dir A pointer to the file/directory object that shall be mounted at *path*.

Returns

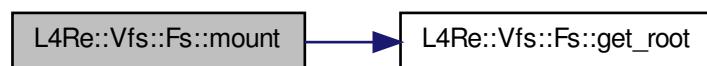
0 on success, or <0 on error.

Definition at line 968 of file [vfs.h](#).

References [get_root\(\)](#).

Referenced by [mount\(\)](#).

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:

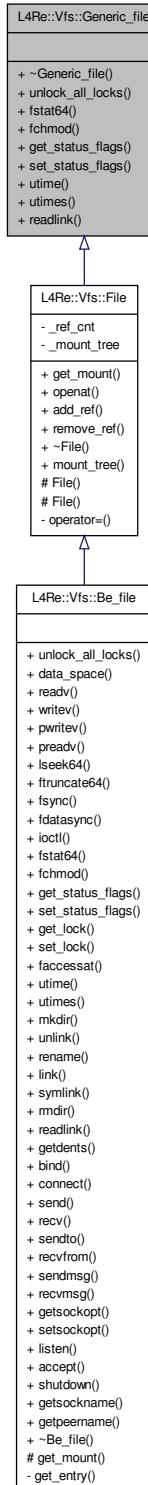
- [14/l4re_vfs/vfs.h](#)

11.67 L4Re::Vfs::Generic_file Class Reference

The common interface for an open POSIX file.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::Generic_file:



Public Member Functions

- virtual int `unlock_all_locks` ()=0 throw ()
Unlock all locks on the file.
- virtual int `fstat64` (struct stat64 *buf) const =0 throw ()
Get status information for the file.
- virtual int `fchmod` (mode_t)=0 throw ()
Change POSIX access rights on that file.
- virtual int `get_status_flags` () const =0 throw ()
Get file status flags (fcntl F_GETFL).
- virtual int `set_status_flags` (long flags)=0 throw ()
Set file status flags (fcntl F_SETFL).

11.67.1 Detailed Description

The common interface for an open POSIX file. This interface is common to all kinds of open files, independent of the 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 mor information.

Definition at line 63 of file [vfs.h](#).

11.67.2 Member Function Documentation

11.67.2.1 virtual int `L4Re::Vfs::Generic_file::unlock_all_locks` () throw () [pure virtual]

Unlock all locks on the file.

Note

All locks means all locks independent by which file the locks were taken.

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.

11.67.2.2 virtual int L4Re::Vfs::Generic_file::fstat64 (struct stat64 * *buf*) const throw () [pure virtual]

Get status information for the file.

This is the backend for POSIX fstat, stat, fstat64 and friends.

Return values

buf This buffer is filled with the status information.

Returns

0 on success, or <0 on error.

11.67.2.3 virtual int L4Re::Vfs::Generic_file::fchmod (mode_t) throw () [pure virtual]

Change POSIX access rights on that file.

Backend for POSIX chmod and fchmod.

11.67.2.4 virtual int L4Re::Vfs::Generic_file::get_status_flags () const throw () [pure virtual]

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.

11.67.2.5 virtual int L4Re::Vfs::Generic_file::set_status_flags (long *flags*) throw () [pure virtual]

Set file status flags (fcntl F_SETFL).

This function is used by the fcntl implementation for the F_SETFL command).

Parameters

flags 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.

The documentation for this class was generated from the following file:

- l4/re vfs/vfs.h

11.68 gfxbitmap_offset Struct Reference

offsets in pmap[] and bmap[]

```
#include <bitmap.h>
```

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

11.68.1 Detailed Description

offsets in pmap[] and bmap[]

Definition at line 67 of file [bitmap.h](#).

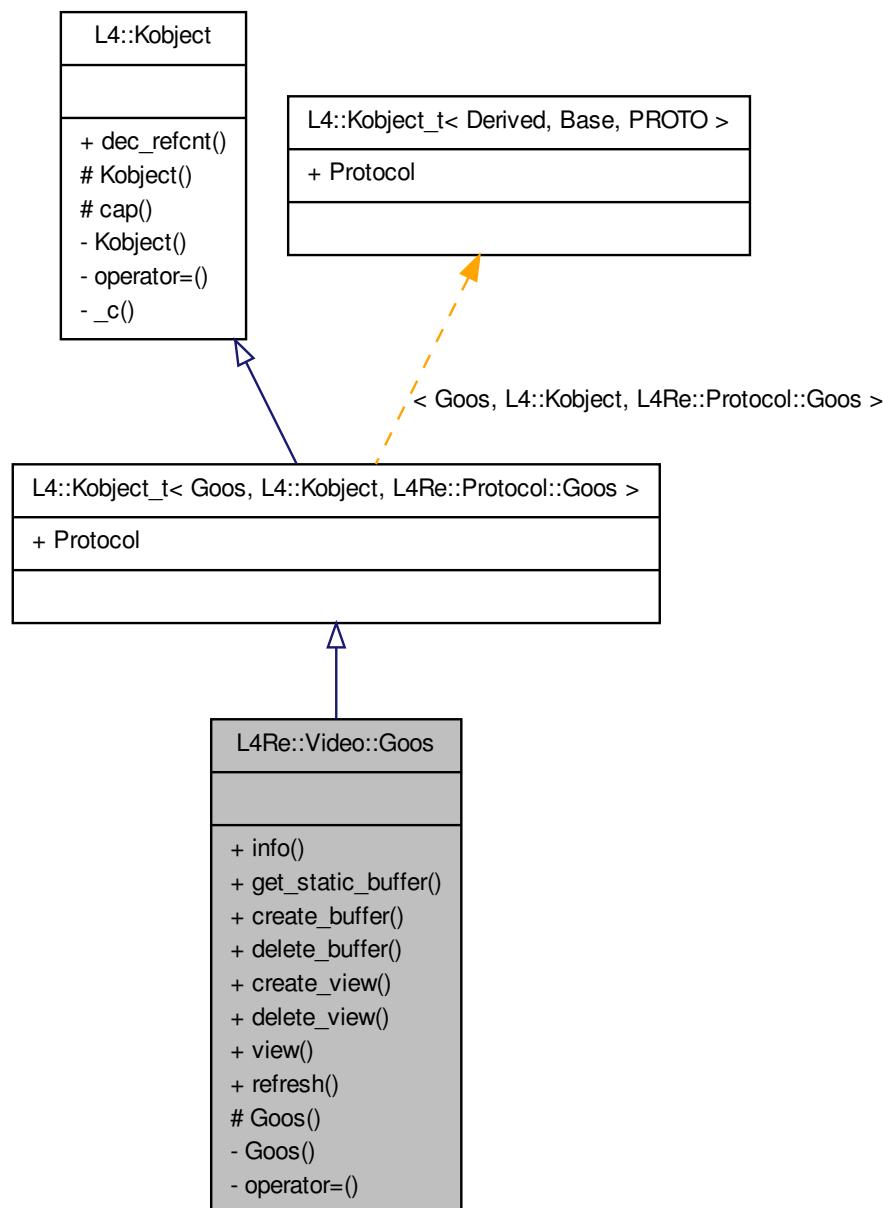
The documentation for this struct was generated from the following file:

- `l4/libgfxbitmap(bitmap.h)`

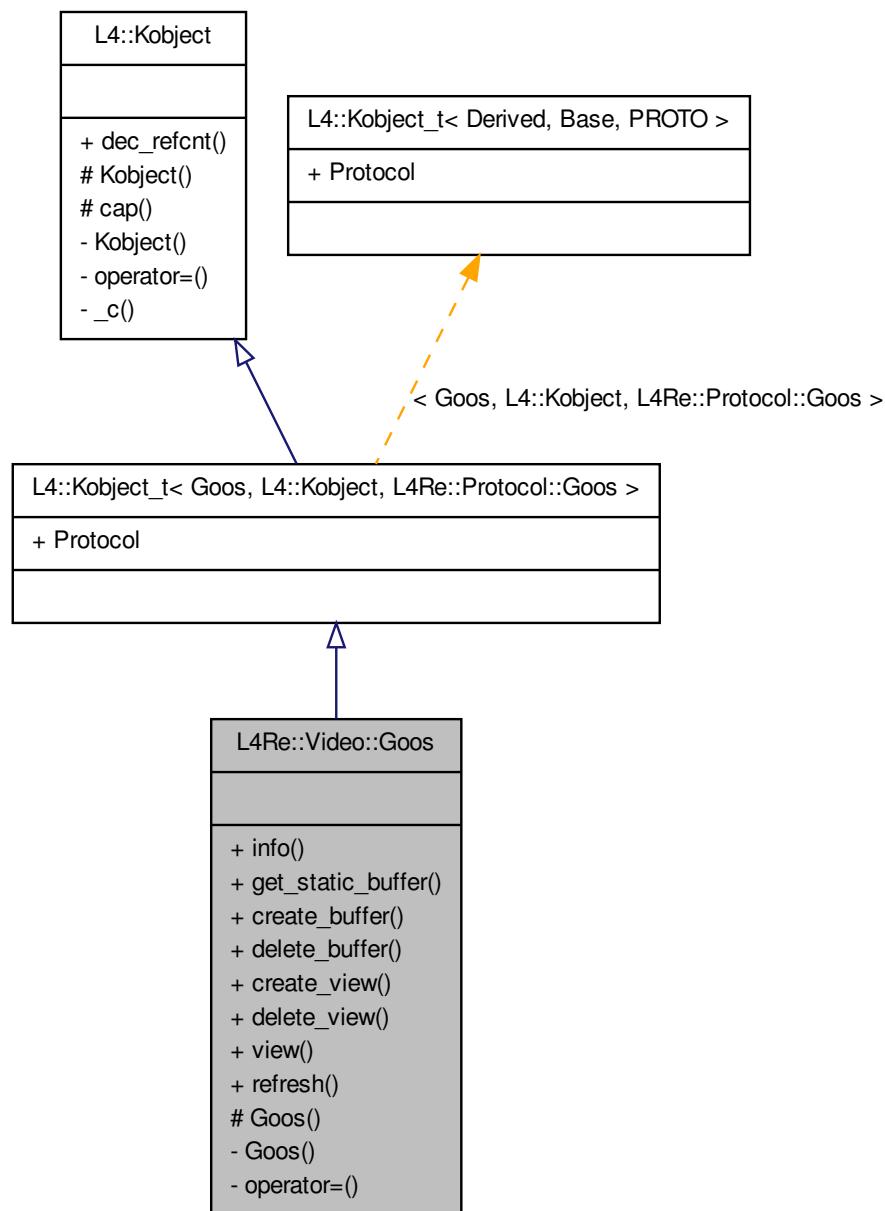
11.69 L4Re::Video::Goos Class Reference

A 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

- int **info** (**Info** *) const throw ()
Return the goos information of the goos.
- int **get_static_buffer** (unsigned idx, **L4::Cap< L4Re::Dataspace > rbuf**) const throw ()
Return a static buffer of a goos.
- int **create_buffer** (unsigned long size, **L4::Cap< L4Re::Dataspace > rbuf**) const throw ()
Create a buffer.
- int **delete_buffer** (unsigned idx) const throw ()
Delete a buffer.
- int **create_view** (**View** *view) const throw ()
Create a view.
- int **delete_view** (**View** const &v) const throw ()
Delete a view.
- **View** **view** (unsigned index) const throw ()
Return a view.
- int **refresh** (int x, int y, int w, int h) throw ()
Trigger refreshing of the given area on the virtual screen.

11.69.1 Detailed Description

A goos.

Definition at line 39 of file [goos](#).

11.69.2 Member Enumeration Documentation

11.69.2.1 enum L4Re::Video::Goos::Flags

Flags for a goos.

Enumerator:

F_auto_refresh The graphics display is automatically refreshed.

F_pointer We have a mouse pointer.

F_dynamic_views Supports dynamically allocated views.

F_dynamic_buffers Supports dynamically allocated buffers.

Definition at line 46 of file [goos](#).

11.69.3 Member Function Documentation

11.69.3.1 int L4Re::Video::Goos::info (Info *) const throw ()

Return the goos information of the goos.

Return values

info Goos information structure pointer.

Returns

0 on success, error otherwise

11.69.3.2 int L4Re::Video::Goos::get_static_buffer (unsigned idx, L4::Cap< L4Re::Dataspace > rbuf) const throw ()

Return a static buffer of a goos.

Parameters

idx Index of the static buffer.

rbuf Capability slot to point the buffer dataspace to.

Returns

0 on success, error otherwise

11.69.3.3 int L4Re::Video::Goos::create_buffer (unsigned long size, L4::Cap< L4Re::Dataspace > rbuf) const throw ()

Create a buffer.

Parameters

size Size of buffer in bytes.

rbuf Capability slot to point the buffer dataspace to.

Returns

Positive: buffer index, negative: Error code

11.69.3.4 int L4Re::Video::Goos::delete_buffer (*unsigned idx*) const throw ()

Delete a buffer.

Parameters

idx Buffer to delete.

Returns

0 on success, error otherwise

11.69.3.5 int L4Re::Video::Goos::create_view (*View * view*) const throw ()

Create a view.

Return values

view A view object.

Returns

Positive: view index, negative: Error code

11.69.3.6 int L4Re::Video::Goos::delete_view (*View const & v*) const throw ()

Delete a view.

Parameters

v The view object to delete.

Returns

0 on success, error otherwise

11.69.3.7 View L4Re::Video::Goos::view (*unsigned index*) const throw () [inline]

Return a view.

Parameters

index Index of the view to return.

Returns

The view.

Definition at line 133 of file [goos](#).

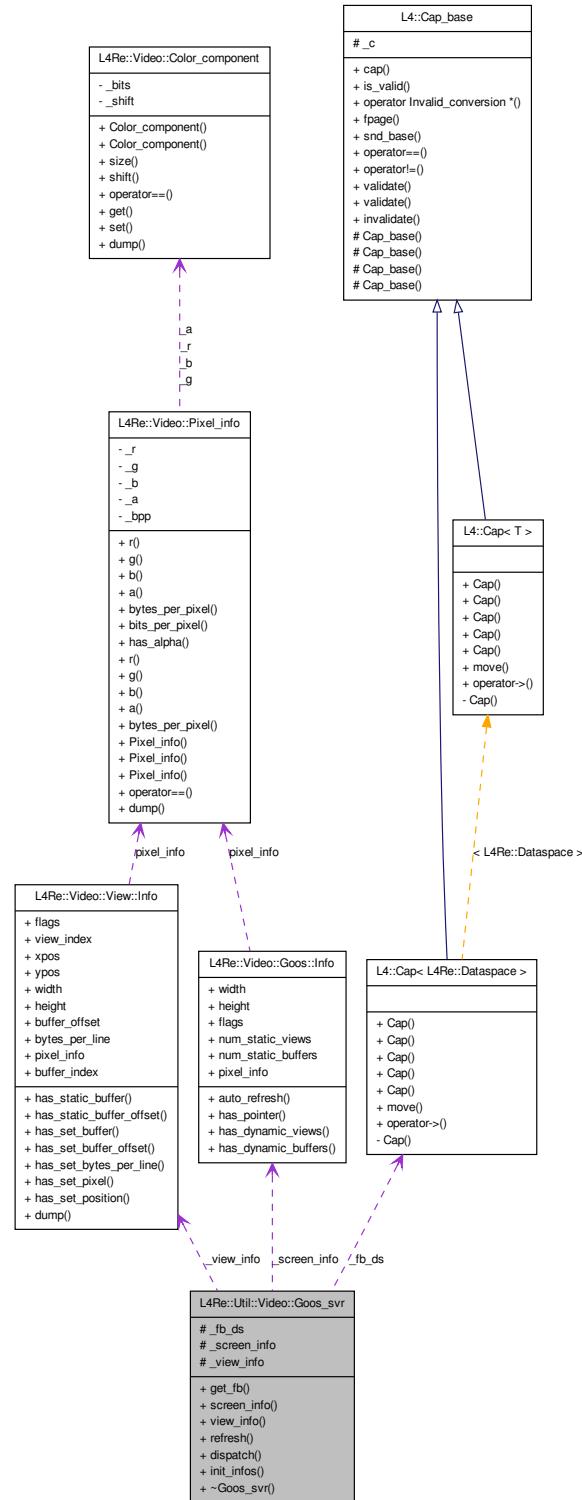
The documentation for this class was generated from the following file:

- [l4/re/video/goos](#)

11.70 L4Re::Util::Video::Goos_svr Class Reference

Goos server class.

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.
- `int dispatch (l4_umword_t obj, L4::Ipc::Iostream &ios)`
Server dispatch function.
- `void init_infos ()`
Initialize the view information structure of this object.
- `virtual ~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.

11.70.1 Detailed Description

Goos server class.

Definition at line 33 of file `goos_svr`.

11.70.2 Member Function Documentation

11.70.2.1 `L4::Cap<L4Re::Dataspace> L4Re::Util::Video::Goos_svr::get_fb () const` `[inline]`

Return framebuffer memory dataspace.

Returns

Goos memory dataspace

Definition at line 41 of file [goos_svr](#).

11.70.2.2 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 47 of file [goos_svr](#).

11.70.2.3 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 53 of file [goos_svr](#).

References [_fb_ds](#).

11.70.2.4 virtual int L4Re::Util::Video::Goos_svr::refresh (int *x*, int *y*, int *w*, int *h*) [inline, virtual]

Refresh area of the framebuffer.

Parameters

x X coordinate (pixels)

y Y coordinate (pixels)

w Width of area in pixels

h Height of area in pixels

Returns

0 on success, negative error code otherwise

Definition at line 65 of file [goos_svr](#).

References [_view_info](#).

11.70.2.5 int L4Re::Util::Video::Goos_svr::dispatch (l4_umword_t *obj*, L4::Ipc::Iostream & *ios*) [inline]

Server dispatch function.

Parameters

obj Server object ID to work on

ios Input/Output stream.

Returns

error code.

Definition at line 116 of file [goos_svr](#).

11.70.2.6 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 86 of file [goos_svr](#).

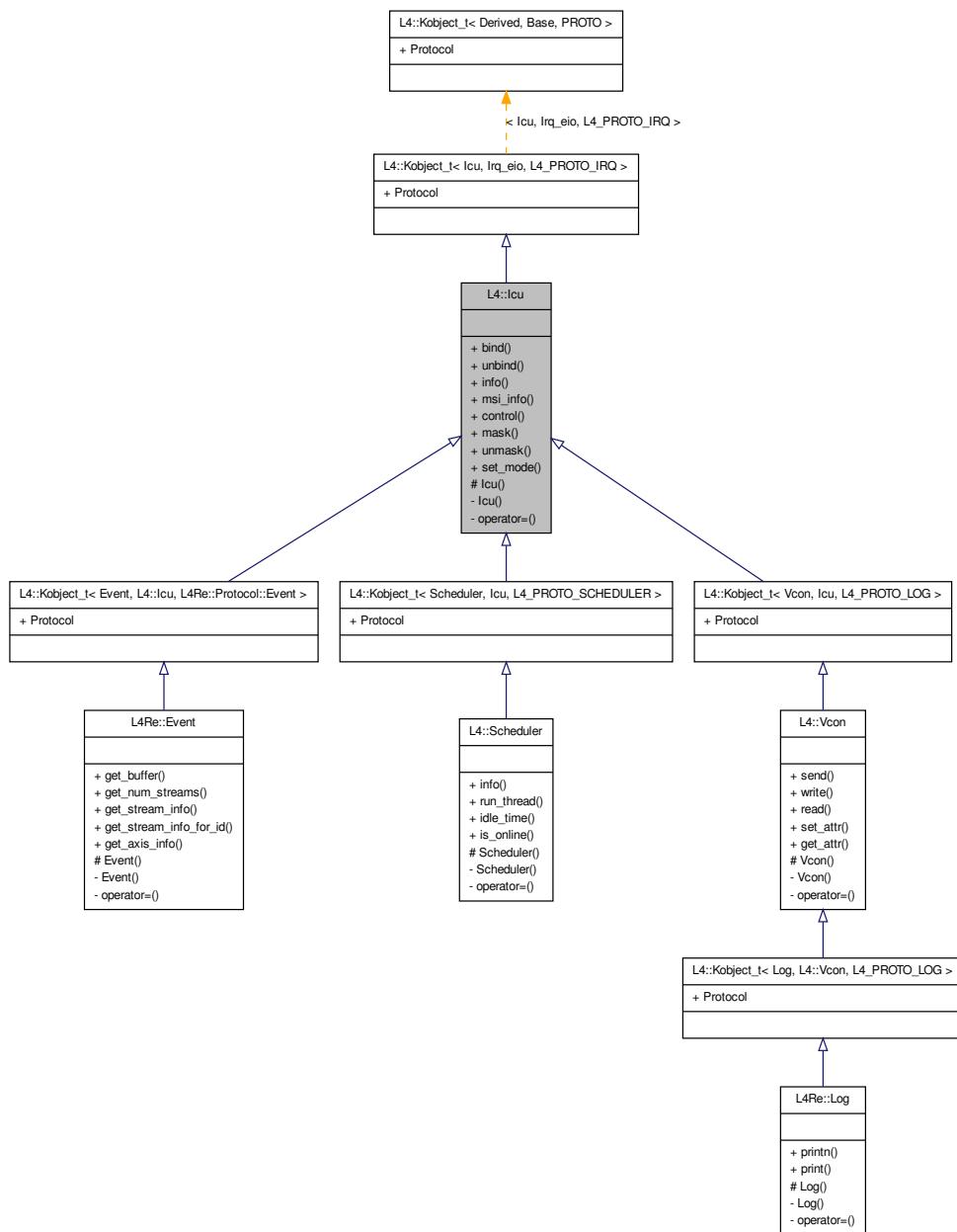
The documentation for this class was generated from the following file:

- l4/re/util/video/goos_svr

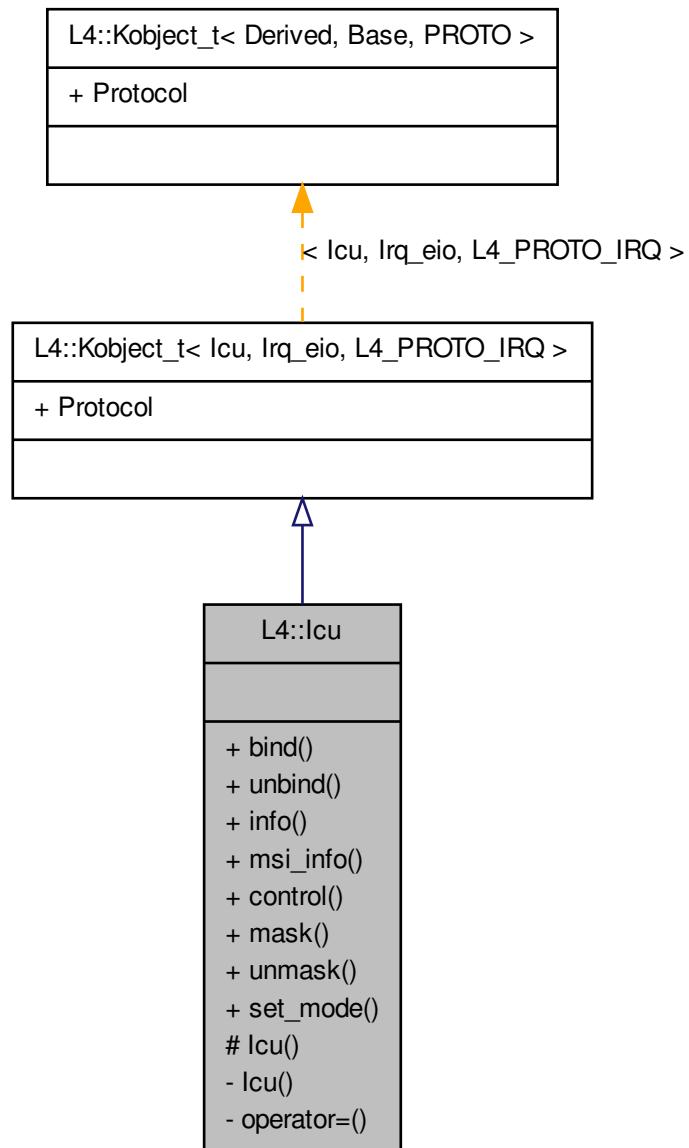
11.71 L4::Icu Class Reference

C++ version of an interrupt controller.

Inheritance diagram for L4::Icu:



Collaboration diagram for L4::Icu:



Data Structures

- class [Info](#)

Info for an ICU.

Public Member Functions

- [l4_mshtag_t bind](#) (unsigned irqnum, [L4::Cap<Irq> irq](#), [l4_utcb_t *utcb=l4_utcb\(\)](#)) throw ()
- [l4_mshtag_t unbind](#) (unsigned irqnum, [L4::Cap<Irq> irq](#), [l4_utcb_t *utcb=l4_utcb\(\)](#)) throw ()
- [l4_mshtag_t info](#) ([l4_icu_info_t *info](#), [l4_utcb_t *utcb=l4_utcb\(\)](#)) throw ()
- [l4_mshtag_t msi_info](#) (unsigned irqnum, [l4_umword_t *msg](#), [l4_utcb_t *utcb=l4_utcb\(\)](#)) throw ()
- [l4_mshtag_t mask](#) (unsigned irqnum, [l4_umword_t *label=0](#), [l4_timeout_t to=L4_IPC_NEVER](#), [l4_utcb_t *utcb=l4_utcb\(\)](#)) throw ()
- [l4_mshtag_t unmask](#) (unsigned irqnum, [l4_umword_t *label=0](#), [l4_timeout_t to=L4_IPC_NEVER](#), [l4_utcb_t *utcb=l4_utcb\(\)](#)) throw ()
- [l4_mshtag_t set_mode](#) (unsigned irqnum, [l4_umword_t mode](#), [l4_utcb_t *utcb=l4_utcb\(\)](#)) throw ()

11.71.1 Detailed Description

C++ version of an interrupt controller. #include <l4/sys/icu>

See also

[Interrupt controller](#) for an overview and C bindings.

Definition at line [125](#) of file [irq](#).

11.71.2 Member Function Documentation

11.71.2.1 [l4_mshtag_t L4::Icu::bind \(unsigned irqnum, L4::Cap<Irq> irq, l4_utcb_t * utcb = l4_utcb\(\) \) throw \(\) \[inline\]](#)

Bind an interrupt vector of an interrupt controller to an interrupt object.

Parameters

icu ICU to use.

irqnum IRQ vector at the ICU.

irq IRQ capability to bind the IRQ to.

Returns

Syscall return tag

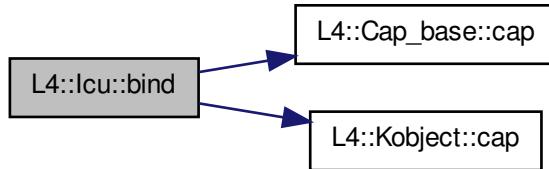
Note

the icu argument is the implicit this pointer.

Definition at line [163](#) of file [irq](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.71.2.2 `l4_mshtag_t L4::Icu::unbind (unsigned irqnum, L4::Cap< Irq > irq, l4_utcb_t * utcb = l4_utcb()) throw () [inline]`

Remove binding of an interrupt vector from the interrupt controller object.

Parameters

icu ICU to use.
irqnum IRQ vector at the ICU.
irq IRQ object to remove from the ICU.

Returns

Syscall return tag

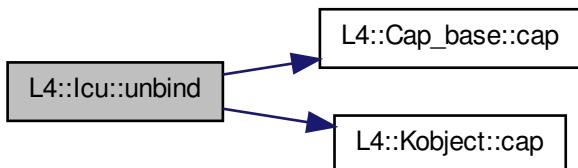
Note

*the icu argument is the implicit *this* pointer.*

Definition at line 171 of file `irq.h`.

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.71.2.3 `l4_mshtag_t L4::Icu::info (l4_icu_info_t * info, l4_utcb_t * utcb = l4_utcb()) throw () [inline]`

Get info about capabilites of ICU.

Parameters

icu ICU to use.

info Pointer to an info structure to be filled with information.

Returns

Syscall return tag

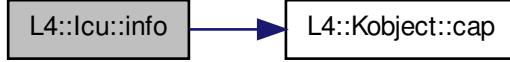
Note

*the icu argument is the implicit *this* pointer.*

Definition at line 179 of file [irq](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.71.2.4 `l4_mshtag_t L4::Icu::msi_info (unsigned irqnum, l4_umword_t * msg, l4_utcb_t * utcb = l4_utcb()) throw () [inline]`

Get MSI info about IRQ.

Parameters

icu ICU to use.

irqnum IRQ vector at the ICU.

msg Pointer to a word to receive the message that must be used for the PCI devices MSI message.

Returns

Syscall return tag

Note

*the icu argument is the implicit *this* pointer.*

Definition at line 186 of file irq.

References L4::Kobject::cap()

Here is the call graph for this function:



11.71.2.5 `l4_msghdr_t L4::Icu::mask (unsigned irqnum, l4_umword_t * label = 0, l4_timeout_t to = L4_IPC_NEVER, l4_utcb_t * utcb = L4_utcb ()) throw () [inline]`

Mask an IRQ vector.

Parameters

icu ICU to use.

irqnum IRQ vector at the ICU.

label If non-NULL the function also waits for the next message.

to Timeout for message to ICU, if unsure use L4_IPC_NEVER.

Returns

Syscall return tag

Note

the icu argument is the implicit *this* pointer.

Definition at line 201 of file [irq](#).

References L4::Kobject::cap().

Here is the call graph for this function:



11.71.2.6 `l4_msntag_t L4::Icu::unmask (unsigned irqnum, l4_umword_t * label = 0, l4_timeout_t to = L4_IPC_NEVER, l4_utcb_t * utcb = L4_utcb()) throw () [inline]`

Unmask an IRQ vector.

Parameters

icu ICU to use.

irqnum IRQ vector at the ICU.

label If non-NULL the function also waits for the next message.

to Timeout for message to ICU, if unsure use L4_IPC_NEVER.

Returns

Syscall return tag

Note

the icu argument is the implicit *this* pointer.

Definition at line 211 of file [irq](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.71.2.7 `l4_msntag_t L4::Icu::set_mode (unsigned irqnum, l4_umword_t mode, l4_utcb_t * utcb = L4_utcb()) throw () [inline]`

Set mode of interrupt.

Parameters

icu ICU to use.

irqnum IRQ vector at the ICU.

mode Mode, see L4_irq_flow_type.

Returns

Syscall return tag

Note

*the icu argument is the implicit *this* pointer.*

Definition at line 221 of file [irq](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



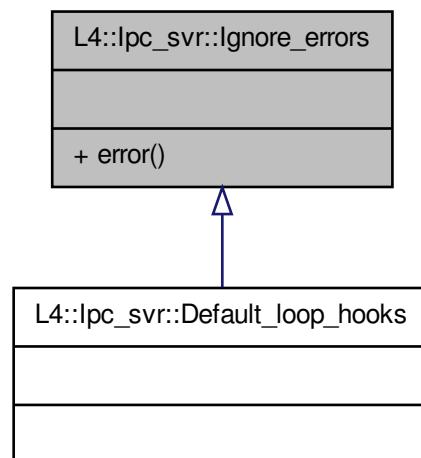
The documentation for this class was generated from the following file:

- [l4/sys/irq](#)

11.72 L4::Ipc_svr::Ignore_errors Struct Reference

Mix in for LOOP_HOOKS to ignore IPC errors.

Inheritance diagram for L4::Ipc_svr::Ignore_errors:



11.72.1 Detailed Description

Mix in for LOOP_HOOKS to ignore IPC errors.

Definition at line 53 of file [ipc_server](#).

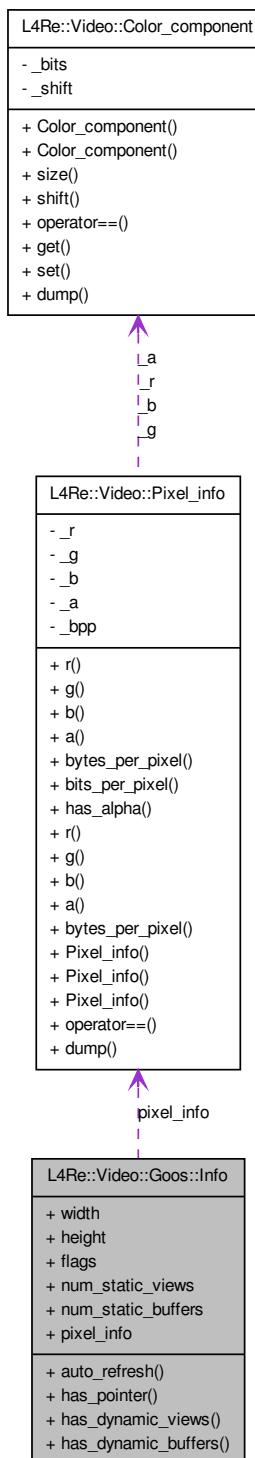
The documentation for this struct was generated from the following file:

- l4/cxx/ipc_server

11.73 L4Re::Video::Goos::Info Struct Reference

Information structure of a 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.

11.73.1 Detailed Description

Information structure of a goos.

Definition at line 55 of file `goos`.

11.73.2 Member Function Documentation

11.73.2.1 `bool L4Re::Video::Goos::Info::auto_refresh () const [inline]`

Return whether this goos does auto refreshing or the view refresh functions must be used to make changes visible.

Definition at line [66](#) of file [goos](#).

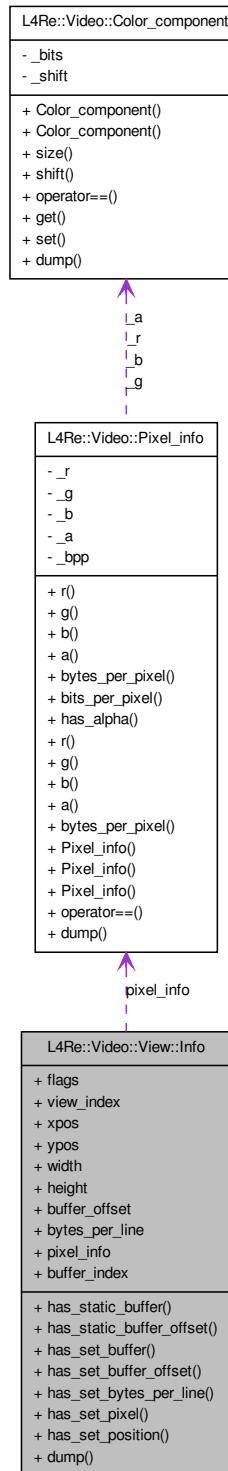
The documentation for this struct was generated from the following file:

- [l4/re/video/goos](#)

11.74 L4Re::Video::View::Info Struct Reference

Information structure of a view.

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 STREAM >
 STREAM & dump (STREAM &s) const`
Dump information on the view information to a stream.

Data Fields

- `unsigned flags`
Flags. ,
- `unsigned view_index`
Index of the view.
- `unsigned long xpos`
X position in pixels of the view in the goos.
- `unsigned long ypos`
Y position in pixels of the view in the goos.
- `unsigned long width`
Width of the view in pixels.
- `unsigned long height`
Height of the view in pixels.
- `unsigned long buffer_offset`
Offset in the memory buffer in bytes.

- `unsigned long bytes_per_line`

Bytes per line.

- `Pixel_info pixel_info`

Pixel information.

- `unsigned buffer_index`

Number of the buffer used for this view.

11.74.1 Detailed Description

Information structure of a view.

Definition at line 85 of file [view](#).

11.74.2 Field Documentation

11.74.2.1 `unsigned L4Re::Video::View::Info::flags`

Flags,.

See also

[Flags](#) and [V_flags](#)

Definition at line 87 of file [view](#).

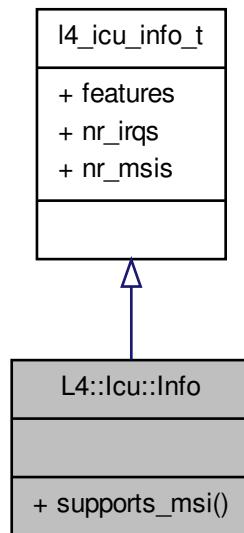
The documentation for this struct was generated from the following file:

- `14/re/video/view`

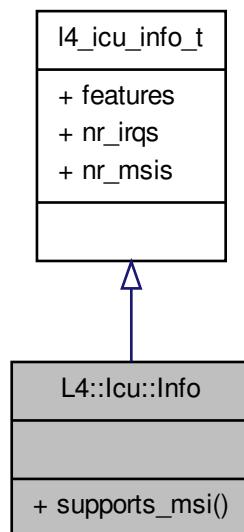
11.75 L4::Icu::Info Class Reference

[Info](#) for an ICU.

Inheritance diagram for L4::Icu::Info:



Collaboration diagram for L4::Icu::Info:



11.75.1 Detailed Description

[Info](#) for an ICU. This class adds access functions.

See also

[l4_icu_info\(\)](#).

Definition at line [153](#) of file [irq](#).

The documentation for this class was generated from the following file:

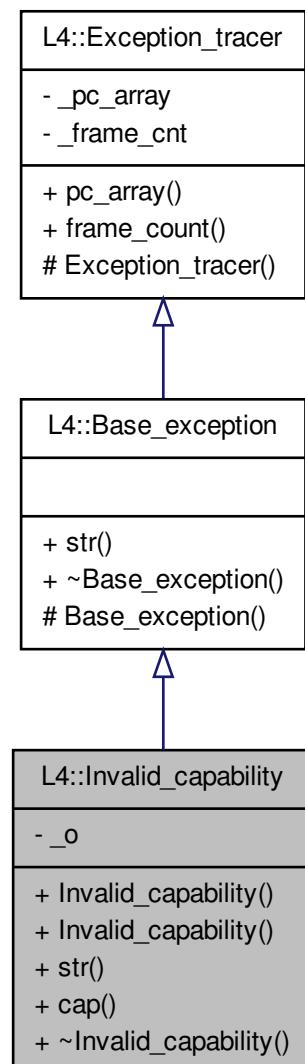
- [l4/sys/irq](#)

11.76 L4::Invalid_capability Class Reference

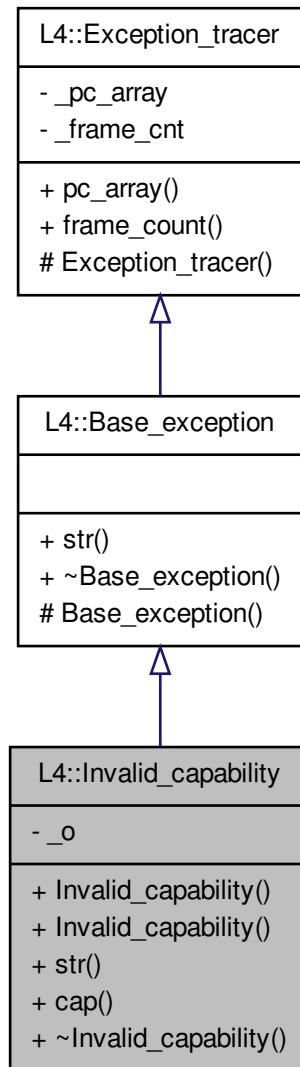
Indicates that an invalid object was invoked.

```
#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`) `throw ()`

Create an Invalid_object exception for the Object o.

- `char const * str () const throw ()`

Should return a human readable string for the exception.

- `Cap< void > const & cap () const throw ()`
Get the object that caused the error.

11.76.1 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 225 of file [exceptions](#).

11.76.2 Constructor & Destructor Documentation

11.76.2.1 `L4::Invalid_capability::Invalid_capability (Cap< void > const & o) throw () [inline, explicit]`

Create an Invalid_object exception for the Object o.

Parameters

- `o` The object that caused the server side error.

Definition at line 235 of file [exceptions](#).

11.76.3 Member Function Documentation

11.76.3.1 `Cap<void> const& L4::Invalid_capability::cap () const throw () [inline]`

Get the object that caused the error.

Returns

- The object that caused the error on invocation.

Definition at line 244 of file [exceptions](#).

The documentation for this class was generated from the following file:

- `l4/cxx/exceptions`

11.77 L4::IOModifier Class Reference

Modifier class for the IO stream.

11.77.1 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 33 of file [basic_ostream](#).

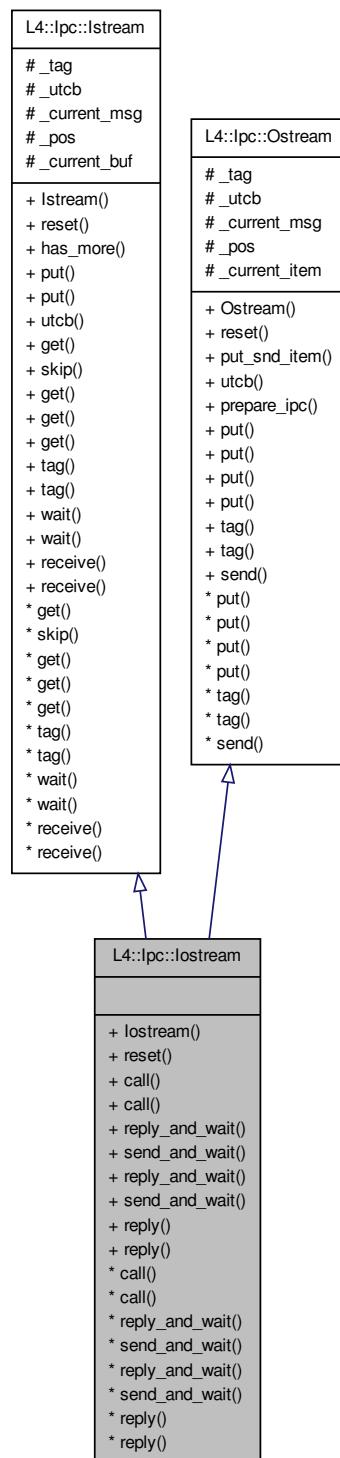
The documentation for this class was generated from the following file:

- l4/cxx/basic_ostream

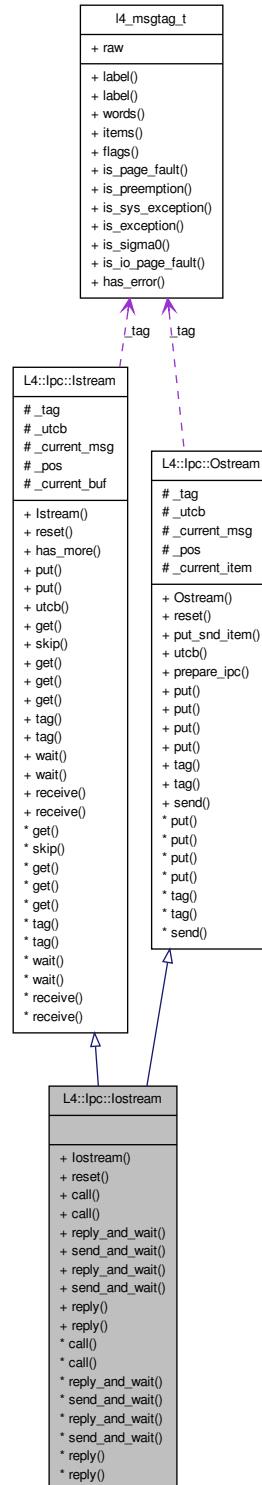
11.78 L4::Ipc::Iostream Class Reference

Input/Output stream for IPC [un]marshalling.

Inheritance diagram for L4::Ipc::Iostream:



Collaboration diagram for L4::Ipc::Iostream:



Public Member Functions

- [Iostream \(l4_utcb_t *utcb\)](#)
Create an IPC IO stream with a single message buffer.
- [void reset \(\)](#)
Reset the stream to its initial state.

IPC operations.

- [l4_mshtag_t call \(l4_cap_idx_t dst\)](#)
Do an IPC call using the message in the output stream and receiving to the input stream.
- [l4_mshtag_t call \(l4_cap_idx_t dst, long label\)](#)
- [l4_mshtag_t reply_and_wait \(l4_umword_t *src_dst, long proto=0\)](#)
Do an IPC reply and wait.
- [l4_mshtag_t send_and_wait \(l4_cap_idx_t dest, l4_umword_t *src, long proto=0\)](#)
- [l4_mshtag_t reply_and_wait \(l4_umword_t *src_dst, l4_timeout_t timeout, long proto=0\)](#)
Do an IPC reply and wait.
- [l4_mshtag_t send_and_wait \(l4_cap_idx_t dest, l4_umword_t *src, l4_timeout_t timeout, long proto=0\)](#)
- [l4_mshtag_t reply \(l4_timeout_t timeout, long proto=0\)](#)
- [l4_mshtag_t reply \(long proto=0\)](#)

11.78.1 Detailed Description

Input/Output stream for IPC [un]marshalling. The [Ipc::Iostream](#) is part of the AW Env IPC framework as well as [Ipc::Istream](#) and [Ipc::Ostream](#). In particular an [Ipc::Iostream](#) is a combination of an [Ipc::Istream](#) and an [Ipc::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/clntsrv/client.cc](#), [examples/clntsrv/server.cc](#), [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 997 of file [ipc_stream](#).

11.78.2 Constructor & Destructor Documentation

11.78.2.1 L4::Ipc::Iostream::Iostream (l4_utcb_t * utcb) [inline, explicit]

Create an IPC IO stream with a single message buffer.

Parameters

msg 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 1008 of file [ipc_stream](#).

11.78.3 Member Function Documentation

11.78.3.1 void L4::Ipc::Iostream::reset() [inline]

Reset the stream to its initial state.

Input as well as the output stream are reset.

Reimplemented from [L4::Ipc::Ostream](#).

Definition at line 1018 of file [ipc_stream](#).

11.78.3.2 l4_mshtag_t L4::Ipc::Iostream::call(l4_cap_idx_t dst) [inline]

Do an IPC call using the message in the output stream and receiving to the input stream.

Parameters

dst The destination [L4](#) UID (thread) to call.

Returns

the result dope 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:

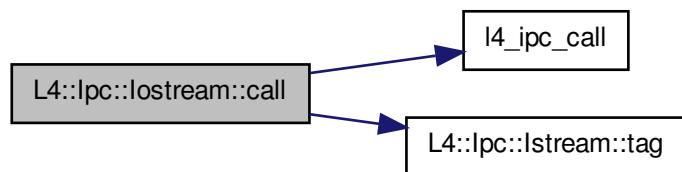
[examples/clntsrv/client.cc](#), and [examples/libs/l4re/streammap/client.cc](#).

Definition at line 1156 of file [ipc_stream](#).

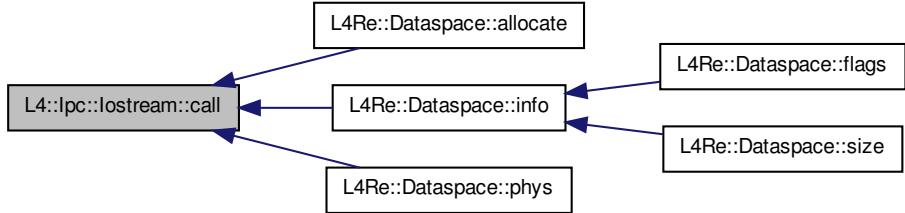
References [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), and [L4::Ipc::Istream::tag\(\)](#).

Referenced by [L4Re::Dataspace::allocate\(\)](#), [L4Re::Dataspace::info\(\)](#), and [L4Re::Dataspace::phys\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.78.3.3 `l4_mshtag_t L4::Ipc::Iostream::reply_and_wait (l4_umword_t * src_dst, long proto = 0) [inline]`

Do an IPC reply and wait.

Parameters

`src_dst` Input: the destination for the send operation.

Output: the source of the received message.

Returns

the result dope of the IPC operation.

This is a combined IPC operation consisting of a send operation and an open wait for any message.

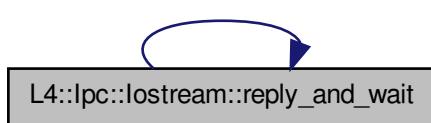
A reply and wait is usually used by servers that reply to a client and wait for the next request by any other client.

Definition at line 1073 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:



11.78.3.4 **`l4_mshtag_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

`src_dst` Input: the destination for the send operation.
 Output: the source of the received message.
`timeout` Timeout used for IPC.

Returns

the result dope of the IPC operation.

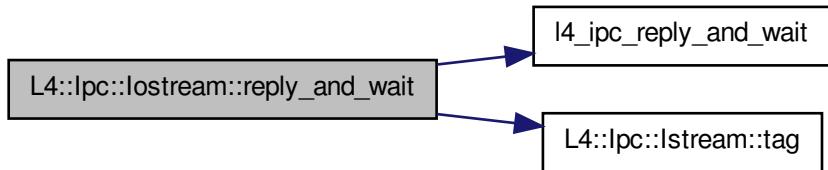
This is a combined IPC operation consisting of a send operation and an open wait for any message.

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 1177 of file `ipc_stream`.

References [l4_ipc_reply_and_wait\(\)](#), and [L4::Ipc::Istream::tag\(\)](#).

Here is the call graph for this function:



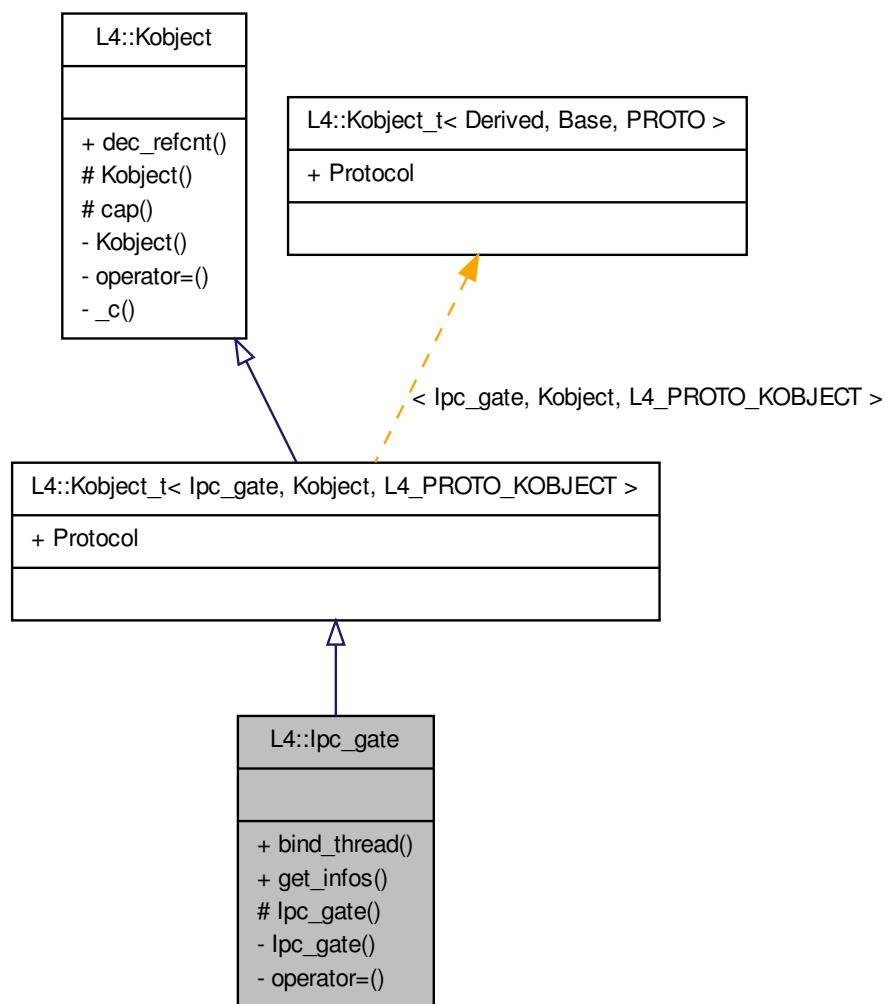
The documentation for this class was generated from the following file:

- l4/cxx/ ipc_stream

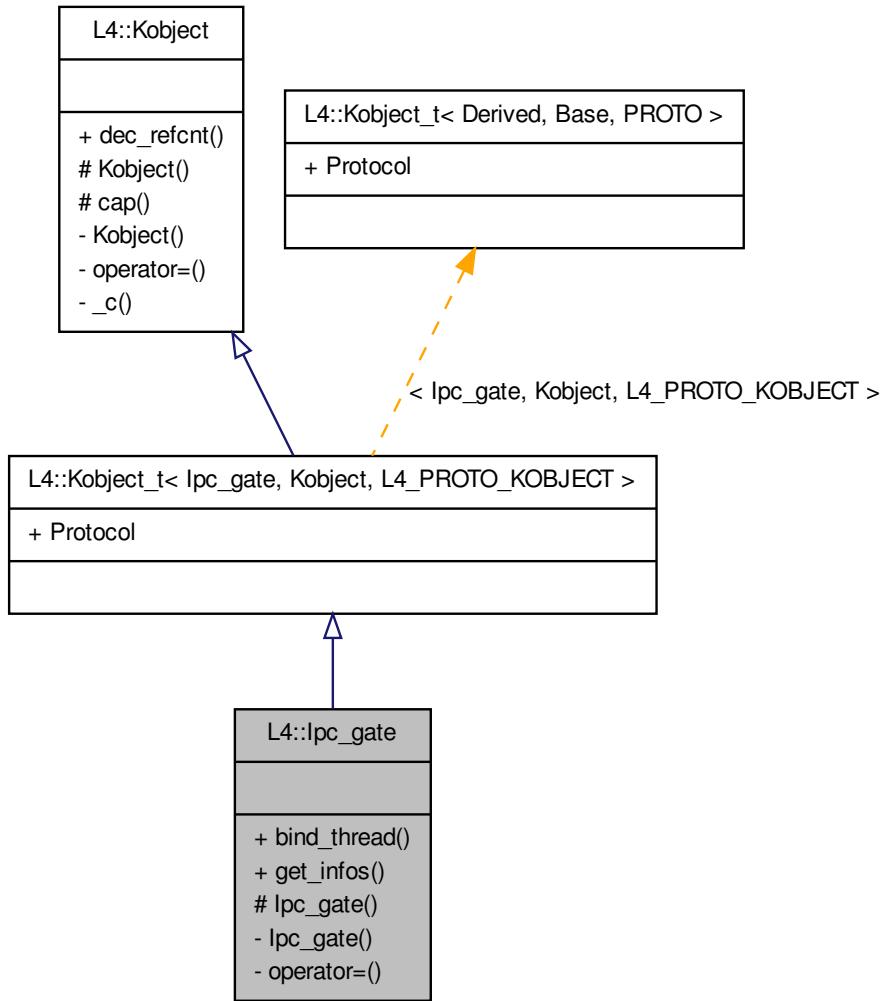
11.79 L4::Ipc_gate Class Reference

[L4](#) IPC gate.

Inheritance diagram for L4::Ipc_gate:



Collaboration diagram for L4::Ipc_gate:



Public Member Functions

- `l4_msgtag_t bind_thread (Cap< Thread > t, l4_umword_t label, l4_utcb_t *utcb=l4_utcb()) throw ()`
Bind the IPC-gate to the thread.
- `l4_msgtag_t get_infos (l4_umword_t *label, l4_utcb_t *utcb=l4_utcb()) throw ()`
Get information on the IPC-gate.

11.79.1 Detailed Description

[L4](#) IPC gate. #include <l4/sys/ ipc_gate>

Definition at line 39 of file [ipc_gate](#).

11.79.2 Member Function Documentation

11.79.2.1 l4_mshtag_t L4::Ipc_gate::bind_thread (Cap< Thread > t, l4_umword_t label, l4_utcb_t * utcb = [l4_utcb\(\)](#)) throw () [inline]

Bind the IPC-gate to the thread.

See also

[l4_ipc_gate_bind_thread](#)

Definition at line 50 of file [ipc_gate](#).

References [L4::Cap_base::cap\(\)](#).

Here is the call graph for this function:



11.79.2.2 l4_mshtag_t L4::Ipc_gate::get_infos (l4_umword_t * label, l4_utcb_t * utcb = [l4_utcb\(\)](#)) throw () [inline]

Get information on the IPC-gate.

See also

[l4_ipc_gate_get_infos](#)

Definition at line 59 of file [ipc_gate](#).

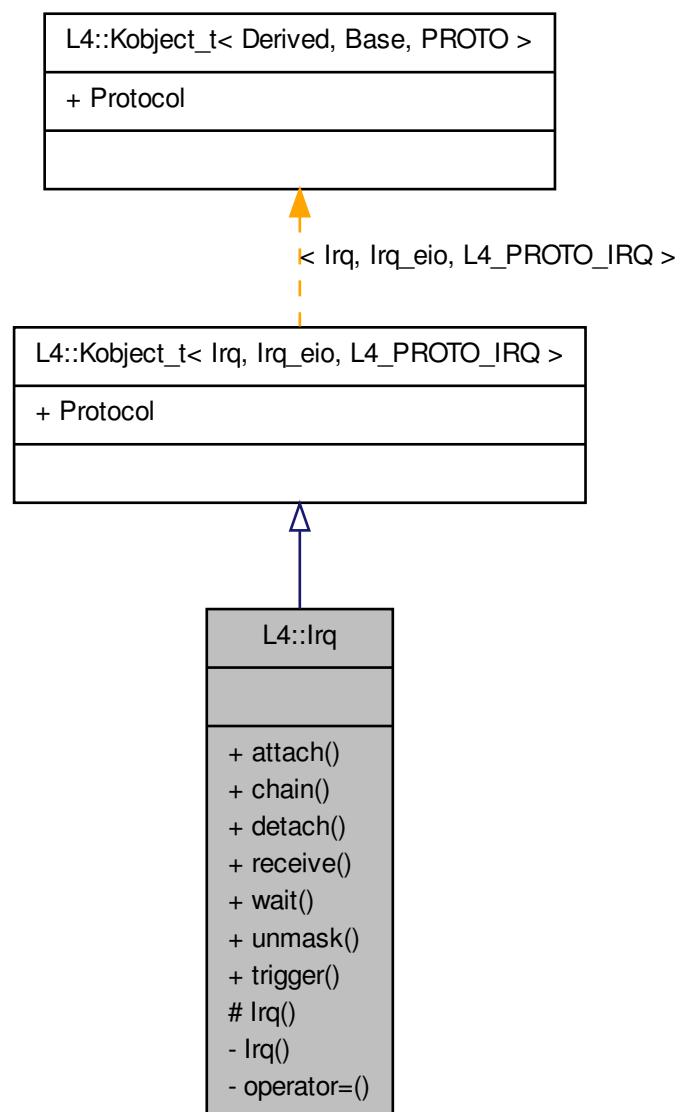
The documentation for this class was generated from the following file:

- l4/sys/ ipc_gate

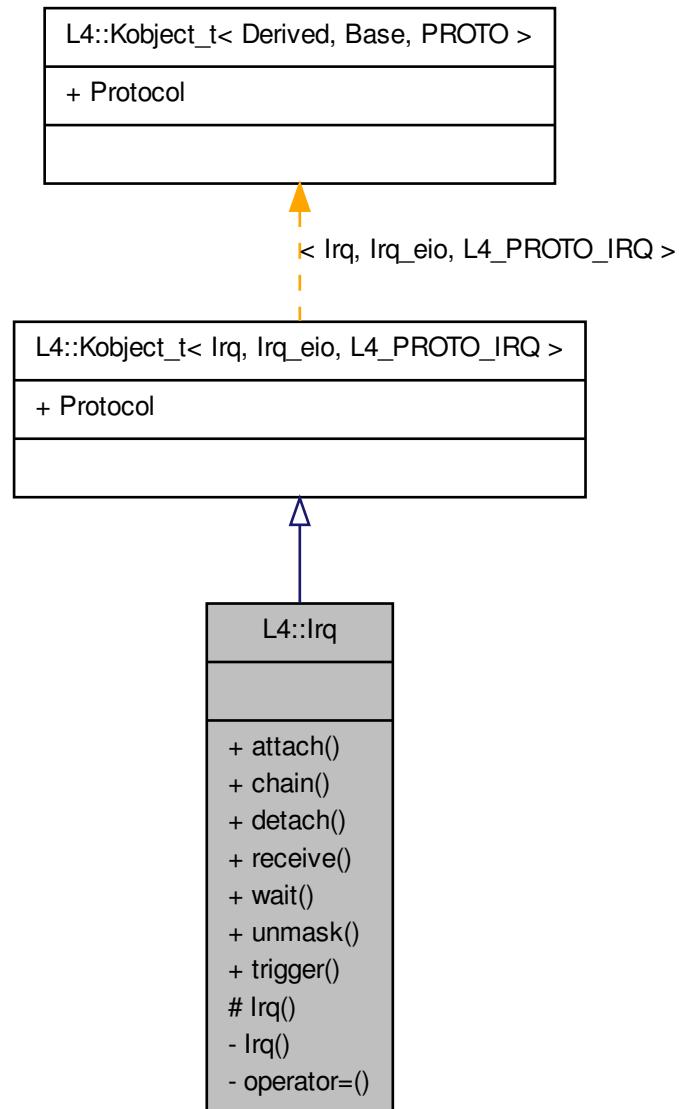
11.80 L4::Irq Class Reference

C++ version of an [L4](#) IRQ.

Inheritance diagram for L4::Irq:



Collaboration diagram for L4::Irq:



Public Member Functions

- `l4_mshtag_t attach (l4_umword_t label, Cap< Thread > const &thread, l4_utcb_t *utcb=l4_utcb())` throw ()
- `l4_mshtag_t chain (l4_umword_t label, Cap< Irq > const &slave, l4_utcb_t *utcb=l4_utcb())` throw ()
- `l4_mshtag_t detach (l4_utcb_t *utcb=l4_utcb())` throw ()

- `l4_mshtag_t receive (l4_timeout_t to=L4_IPC_NEVER, l4_utcb_t *utcb=l4_utcb()) throw ()`
- `l4_mshtag_t wait (l4_umword_t *label, l4_timeout_t to=L4_IPC_NEVER, l4_utcb_t *utcb=l4_utcb()) throw ()`
- `l4_mshtag_t unmask (l4_utcb_t *utcb=l4_utcb()) throw ()`
- `l4_mshtag_t trigger (l4_utcb_t *utcb=l4_utcb()) throw ()`

11.80.1 Detailed Description

C++ version of an L4 IRQ. #include <l4/sys/irq>

See also

[IRQs](#) for an overview and C bindings.

Examples:

[examples/libs/l4re/c++/shared_ds/ds_clnt.cc](#).

Definition at line 54 of file [irq](#).

11.80.2 Member Function Documentation

11.80.2.1 `l4_mshtag_t L4::Irq::attach (l4_umword_t label, Cap< Thread > const & thread, l4_utcb_t * utcb = l4_utcb()) throw () [inline]`

Attach to an interrupt source.

Parameters

irq IRQ to attach to.

label Identifier of the IRQ.

thread Thread to attach the interrupt to.

Returns

Syscall return tag

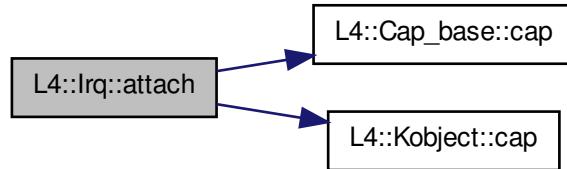
Note

irq is the implicit *this* pointer.

Definition at line 64 of file [irq](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.80.2.2 `l4_mshtag_t L4::Irq::chain (l4_umword_t label, Cap<Irq> const & slave, l4_utcb_t * utcb = l4_utcb()) throw () [inline]`

Chain an IRQ to another master IRQ source.

The chaining feature of IRQ objects allows to deal with shared IRQs. For chaining IRQs there must be a master IRQ object, bound to the real IRQ source. Note, the master IRQ must not have a thread attached to it. This function allows to add a limited number of slave IRQs to this master IRQ, with the semantics that each of the slave IRQs is triggered whenever the master IRQ is triggered. The master IRQ will be masked automatically when an IRQ is delivered and shall be unmasked when all attached slave IRQs are unmasked.

Parameters

irq The master IRQ object.

label Identifier of the IRQ.

slave The slave that shall be attached to the master.

Returns

Syscall return tag

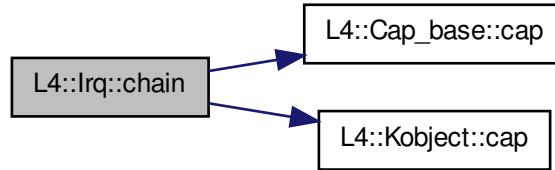
Note

irq is the implicit *this* pointer.

Definition at line 72 of file [irq](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.80.2.3 l4_mshtag_t L4::Irq::detach (l4_utcb_t * utcb = l4_utcb()) throw () [inline]

Detach from an interrupt source.

Parameters

irq IRQ to detach from.

Returns

Syscall return tag

Note

irq is the implicit *this* pointer.

Definition at line 80 of file [irq](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.80.2.4 l4_mshtag_t L4::Irq::receive (l4_timeout_t to = L4_IPC_NEVER, l4_utcb_t * utcb = l4_utcb()) throw () [inline]

Unmask and wait for specified IRQ.

Parameters

irq IRQ to wait for.

to Timeout.

Returns

Syscall return tag

Note

irq is the implicit *this* pointer.

Definition at line 88 of file [irq](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.80.2.5 `l4_msgtag_t L4::Irq::wait (l4_umword_t * label, l4_timeout_t to = L4_IPC_NEVER, l4_utcb_t * utcb = L4_utcb()) throw () [inline]`

Unmask IRQ and wait for any message.

Parameters

irq IRQ to wait for.

label Receive label.

to Timeout.

Returns

Syscall return tag

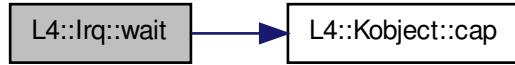
Note

irq is the implicit *this* pointer.

Definition at line 96 of file [irq](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.80.2.6 l4_mshtag_t L4::Irq::unmask (l4_utcb_t * utcb = l4_utcb()) throw () [inline]

Unmask IRQ.

Parameters

irq IRQ to unmask.

Returns

Syscall return tag

Note

`l4_irq_wait` and `l4_irq_receive` are doing the unmask themselves.

Note

irq is the implicit *this* pointer.

Definition at line 104 of file `irq.h`.

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.80.2.7 l4_mshtag_t L4::Irq::trigger (l4_utcb_t * utcb = l4_utcb()) throw () [inline]

Trigger an IRQ.

Parameters

irq IRQ to trigger.

Precondition

irq must be a reference to an IRQ.

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.

Note

irq is the implicit *this* pointer.

Definition at line 111 of file [irq](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



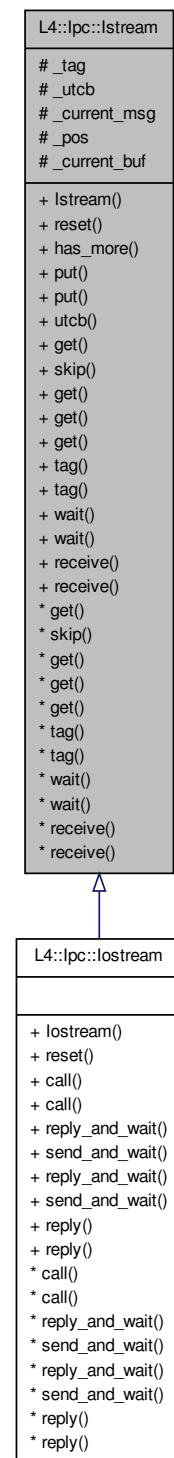
The documentation for this class was generated from the following file:

- [l4/sys/irq](#)

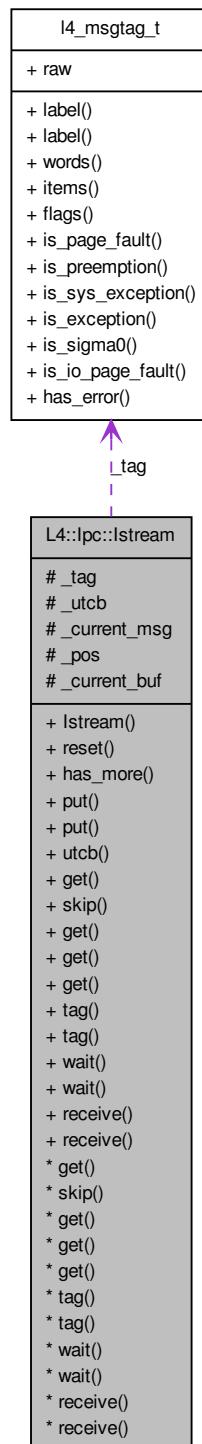
11.81 L4::Ipc::Istream Class Reference

Input stream for IPC unmarshalling.

Inheritance diagram for L4::Ipc::Istream:



Collaboration diagram for L4::Ipc::Istream:



Public Member Functions

- **Istream (l4_utcb_t *utcbl)**
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 ()
Check whether a value of type T can be obtained from the stream.
- **l4_utcb_t * utcb () const**
Return utcb pointer.

Get/Put Functions.

These functions are basically used to implement the extraction operators (>>) and should not be called directly.

See [IPC stream operators](#) .

- 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 size)
Skip size elements of type T in the stream.
- template<typename T >
unsigned long get (Msg_ptr< T > const &buf, unsigned long size=1)
Read one size elements of type T from the stream and return a pointer.
- template<typename T >
void get (T &v)
Extract a single element of type T from the stream.
- **bool get (Ipc::Varg *va)**
- **l4_mshtag_t tag () const**
Get the message tag of a received IPC.
- **l4_mshtag_t & tag ()**
Get the message tag of a received IPC.

IPC operations.

- **l4_mshtag_t wait (l4_umword_t *src)**
Wait for an incoming message from any sender.
- **l4_mshtag_t wait (l4_umword_t *src, l4_timeout_t timeout)**
Wait for an incoming message from any sender.

- [l4_mshtag_t receive \(l4_cap_idx_t src\)](#)
Wait for a message from the specified sender.
- [l4_mshtag_t receive \(l4_cap_idx_t src, l4_timeout_t timeout\)](#)

11.81.1 Detailed Description

Input stream for IPC unmarshalling. [Ipc::Istream](#) is part of the dynamic IPC marshalling infrastructure, as well as [Ipc::Ostream](#) and [Ipc::Iostream](#).

[Ipc::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 [Ipc_buf_cp_in](#) and [Ipc_buf_in](#)) and indirect strings (see [Msg_in_buffer](#) and [Msg_io_buffer](#)).

Definition at line 572 of file [ipc_stream](#).

11.81.2 Constructor & Destructor Documentation

11.81.2.1 [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

msg The message buffer to receive IPC messages.

Definition at line 586 of file [ipc_stream](#).

11.81.3 Member Function Documentation

11.81.3.1 [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 596 of file [ipc_stream](#).

References [l4_msg_regs_t::mr](#).

11.81.3.2 [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

buf Pointer to a buffer for size elements of type *T*.

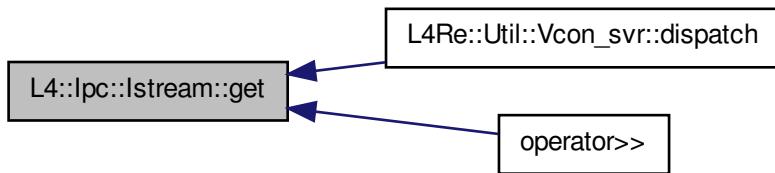
size number of elements of type T to copy out.

See [IPC stream operators](#).

Definition at line 631 of file [ipc_stream](#).

Referenced by [L4Re::Util::Vcon_svr< SVR >::dispatch\(\)](#), and [operator>>\(\)](#).

Here is the caller graph for this function:



11.81.3.3 template<typename T > void L4::Ipc::Istream::skip (unsigned long *size*) [inline]

Skip *size* elements of type T in the stream.

Parameters

size number of elements to skip.

Definition at line 649 of file [ipc_stream](#).

11.81.3.4 template<typename T > unsigned long L4::Ipc::Istream::get (Msg_ptr< T > const & *buf*, unsigned long *size* = 1) [inline]

Read one *size* elements of type T from the stream and return a pointer.

In contrast to a normal get, this version does actually not copy the data but returns a pointer to the data.

Parameters

buf a [Msg_ptr](#) that is actually set to point to the element in the stream.

size number of elements to extract (default is 1).

See [IPC stream operators](#).

Definition at line 672 of file [ipc_stream](#).

11.81.3.5 template<typename T > void L4::Ipc::Istream::get (T & *v*) [inline]

Extract a single element of type T from the stream.

Parameters

- ✓ Output: the element.

See [IPC stream operators](#) .

Definition at line 692 of file [ipc_stream](#).

11.81.3.6 **l4_mshtag_t L4::Ipc::Istream::tag () const [inline]**

Get the message tag of a received IPC.

Returns

The [L4](#) message tag for the received IPC.

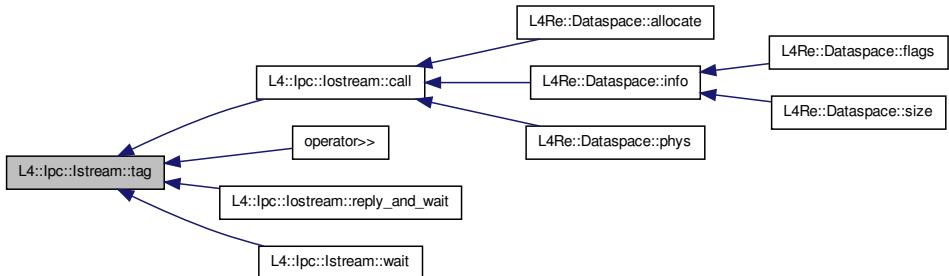
This is in particular useful for handling page faults or exceptions.

See [IPC stream operators](#) .

Definition at line 728 of file [ipc_stream](#).

Referenced by [L4::Ipc::Iostream::call\(\)](#), [operator>>\(\)](#), [L4::Ipc::Iostream::reply_and_wait\(\)](#), and [wait\(\)](#).

Here is the caller graph for this function:



11.81.3.7 **l4_mshtag_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 [IPC stream operators](#) .

Definition at line 739 of file [ipc_stream](#).

11.81.3.8 l4_mshtag_t L4::Ipc::Istream::wait (l4_umword_t * src) [inline]

Wait for an incoming message from any sender.

Parameters

src contains the sender after a successful IPC operation.

Returns

The IPC result dope (l4_mshtag_t).

This wait is actually known as 'open wait'.

Definition at line 768 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:

**11.81.3.9 l4_mshtag_t L4::Ipc::Istream::wait (l4_umword_t * src, l4_timeout_t timeout) [inline]**

Wait for an incoming message from any sender.

Parameters

src contains the sender after a successful IPC operation.

timeout Timeout used for IPC.

Returns

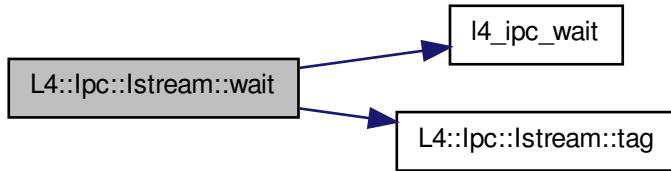
The IPC result dope ([l4_mshtag_t](#)).

This wait is actually known as 'open wait'.

Definition at line [1209](#) of file [ipc_stream](#).

References [l4_ipc_wait\(\)](#), and [tag\(\)](#).

Here is the call graph for this function:



11.81.3.10 l4_mshtag_t L4::Ipc::Istream::receive (l4_cap_idx_t src) [inline]

Wait for a message from the specified sender.

Parameters

src The sender id to receive from.

Returns

The IPC result dope ([l4_mshtag_t](#)).

This is commonly known as 'closed wait'.

Definition at line [788](#) 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:



The documentation for this class was generated from the following file:

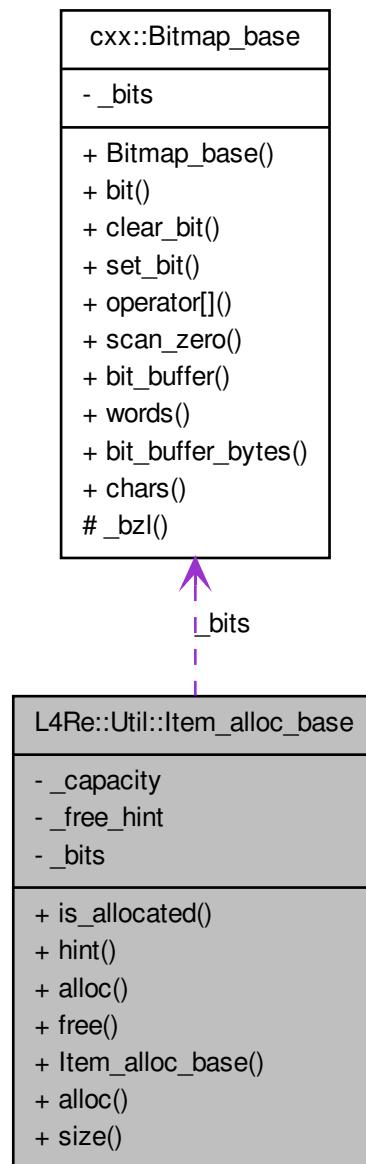
- l4/cxx/ ipc_stream

11.82 L4Re::Util::Item_alloc_base Class Reference

Item allocator.

Inherited by L4Re::Util::Item_alloc< Bits >.

Collaboration diagram for L4Re::Util::Item_alloc_base:



11.82.1 Detailed Description

Item allocator.

Definition at line 38 of file [item_alloc](#).

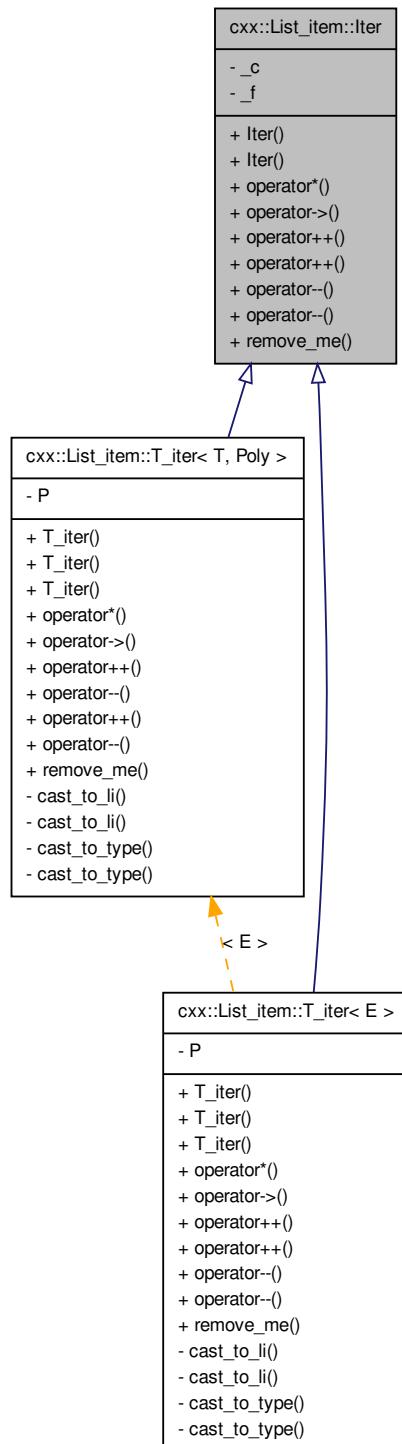
The documentation for this class was generated from the following file:

- l4/re/util/item_alloc

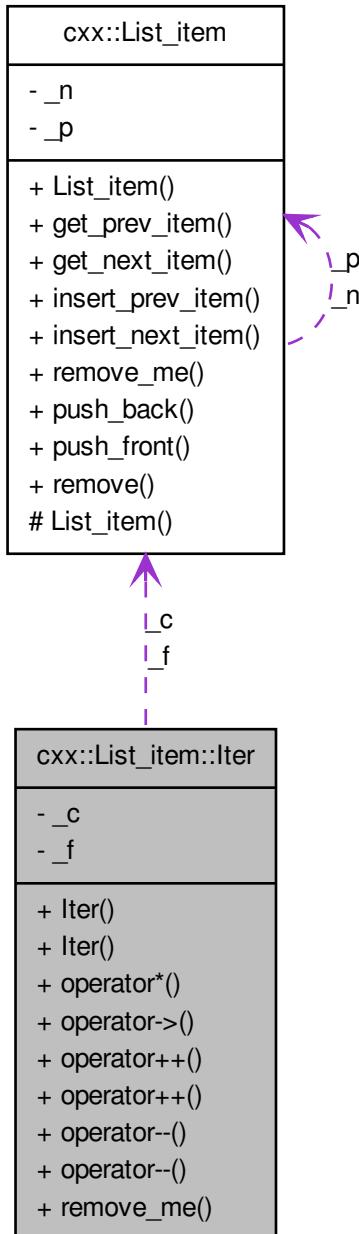
11.83 cxx::List_item::Iter Class Reference

Iterator for a list of ListItem-s.

Inheritance diagram for cxx::List_item::Iter:



Collaboration diagram for cxx::List_item::Iter:



Public Member Functions

- `List_item * remove_me () throw ()`

Remove item pointed to by iterator, and return pointer to element.

11.83.1 Detailed Description

Iterator for a list of ListItem-s. The Iterator iterates til it finds the first element again.

Definition at line 45 of file [list](#).

11.83.2 Member Function Documentation

11.83.2.1 `List_item* cxx::List_item::Iter::remove_me() throw() [inline]`

Remove item pointed to by iterator, and return pointer to element.

Reimplemented in [cxx::List_item::T_iter< T, Poly >](#), and [cxx::List_item::T_iter< E >](#).

Definition at line 86 of file [list](#).

References [cxx::List_item::remove_me\(\)](#).

Referenced by [cxx::List_item::T_iter< T, Poly >::remove_me\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



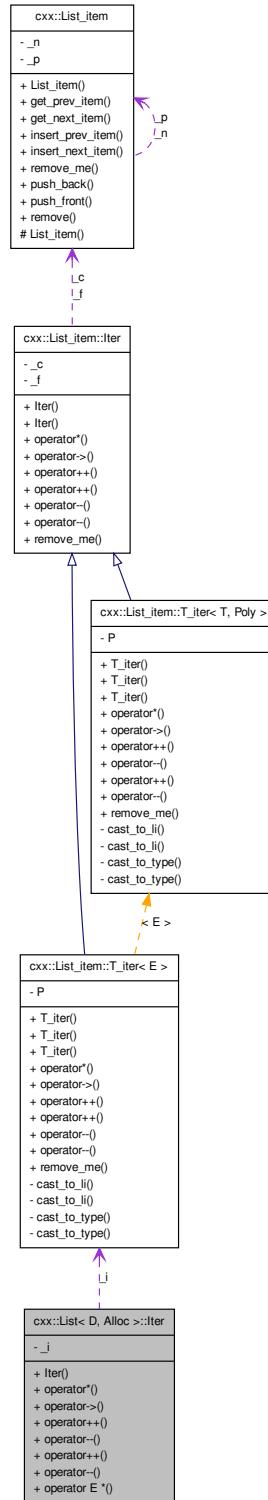
The documentation for this class was generated from the following file:

- [l4/cxx/list](#)

11.84 cxx::List< D, Alloc >::Iter Class Reference

Iterator.

Collaboration diagram for `cxx::List< D, Alloc >::Iter`:



Public Member Functions

- **operator E * () const throw ()**
operator for testing validity (syntactically equal to pointers)

11.84.1 Detailed Description

```
template<typename D, template< typename A > class Alloc = New_allocator> class cxx::List< D, Alloc >::Iter
```

Iterator. Forward and backward iteratable.

Definition at line 356 of file [list](#).

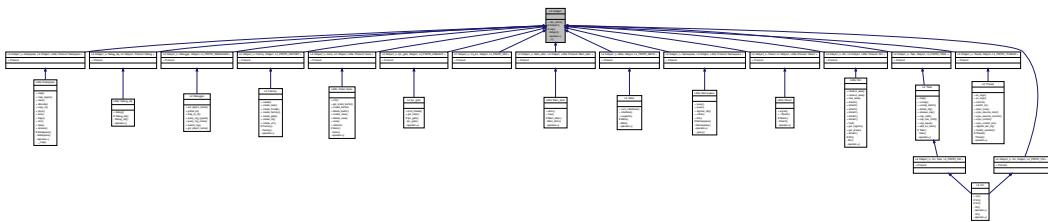
The documentation for this class was generated from the following file:

- 14/cxx/list

11.85 L4::Kobject Class Reference

Base class for all kinds of kernel objects, referred to by capabilities.

Inheritance diagram for L4::Kobject:



Public Member Functions

- `l4_msntag_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 throw ()`
Return capability selector.

Friends

- `template<typename T >`
`Type_info const * kobject_typeid()`

Get the `L4::Type_info` for the `L4Re` interface given in `T`.

11.85.1 Detailed Description

Base class for all kinds of kernel objects, referred to by capabilities. `#include <14/sys/capability>`

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 451 of file `capability`.

11.85.2 Member Function Documentation

11.85.2.1 `l4_cap_idx_t L4::Kobject::cap() const throw() [inline, protected]`

Return capability selector.

Returns

Capability selector.

This method is for derived classes to gain access to the actual capability selector.

Definition at line 477 of file `capability`.

Referenced by `L4::Task::add_ku_mem()`, `L4::Irq::attach()`, `L4::Icu::bind()`, `L4::Task::cap_equal()`, `L4::Task::cap_has_child()`, `L4::Task::cap_valid()`, `L4::Irq::chain()`, `L4::Thread::control()`, `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_vm()`, `dec_refcnt()`, `L4::Task::delete_obj()`, `L4::Irq::detach()`, `L4::Thread::ex_regs()`, `L4::Vcon::get_attr()`, `L4::Scheduler::idle_time()`, `L4::Icu::info()`, `L4::Scheduler::is_online()`, `L4::Task::map()`, `L4::Icu::mask()`, `L4::Thread::modify_senders()`, `L4::Icu::msi_info()`, `L4::Vcon::read()`, `L4::Irq::receive()`, `L4::Thread::register_del_irq()`, `L4::Task::release_cap()`, `L4::Vm::run()`, `L4::Scheduler::run_thread()`, `L4::Vcon::send()`, `L4::Vcon::set_attr()`, `L4::Icu::set_mode()`, `L4::Thread::stats_time()`, `L4::Debugger::switch_log()`, `L4::Thread::switch_to()`, `L4::Irq::trigger()`, `L4::Icu::unbind()`, `L4::Task::unmap()`, `L4::Task::unmap_batch()`, `L4::Icu::unmask()`, `L4::Irq::unmask()`, `L4::Thread::vcpu_control()`, `L4::Thread::vcpu_control_ext()`, `L4::Thread::vcpu_resume_commit()`, `L4::Irq::wait()`, and `L4::Vcon::write()`.

11.85.2.2 `l4_mshtag_t L4::Kobject::dec_refcnt(l4_mword_t diff, l4_utcb_t * utcb = 14_utcb()) [inline]`

Decrement the in kernel reference counter for the object.

Parameters

`diff` is the delta that shall be subtracted from the reference count.

`utcb` is the utcb to use for the invocation.

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.

Definition at line 502 of file [capability](#).

References [cap\(\)](#).

Here is the call graph for this function:



11.85.3 Friends And Related Function Documentation

11.85.3.1 template<typename T > Type_info const* kobject_typeid() [friend]

Get the [L4::Type_info](#) for the [L4Re](#) interface given in *T*.

Parameters

T The type ([L4Re](#) interface) for which the information shall be returned.

Returns

A pointer to the [L4::Type_info](#) structure for *T*.

Reimplemented in [L4::Kobject_t< Ipc_gate, Kobject, L4_PROTO_KOBJECT >](#), [L4::Kobject_t< Namespace, L4::Kobject, L4Re::Protocol::Namespace >](#), [L4::Kobject_t< Parent, L4::Kobject, L4Re::Protocol::Parent >](#), [L4::Kobject_t< Irq_eio, Kobject, L4_PROTO_IRQ >](#), [L4::Kobject_t< Factory, Kobject, L4_PROTO_FACTORY >](#), [L4::Kobject_t< Task, Kobject, L4_PROTO_TASK >](#), [L4::Kobject_t< Debug_obj, L4::Kobject, Protocol::Debug >](#), [L4::Kobject_t< Vm, Kobject, L4_PROTO_VM >](#), [L4::Kobject_t< Event, L4::Icu, L4Re::Protocol::Event >](#), [L4::Kobject_t< Rm, L4::Kobject, L4Re::Protocol::Rm >](#), [L4::Kobject_t< Log, L4::Vcon, L4_PROTO_LOG >](#), [L4::Kobject_t< Vcon, Icu, L4_PROTO_LOG >](#), [L4::Kobject_t< Meta, Kobject, L4_PROTO_META >](#), [L4::Kobject_t< Dataspace, L4::Kobject, L4Re::Protocol::Dataspace >](#), [L4::Kobject_t< Vm, Task, L4_PROTO_VM >](#), [L4::Kobject_t< Debugger, Kobject, L4_PROTO_DEBUGGER >](#), [L4::Kobject_t< Mem_alloc, L4::Kobject, L4Re::Protocol::Mem_alloc >](#), [L4::Kobject_t< Thread, Kobject, L4_PROTO_THREAD >](#), [L4::Kobject_t< Irq, Irq_eio, L4_PROTO_IRQ >](#), [L4::Kobject_t< Scheduler, Icu, L4_PROTO_SCHEDULER >](#), [L4::Kobject_t< Icu, Irq_eio, L4_PROTO_IRQ >](#), and [L4::Kobject_t< Goos, L4::Kobject, L4Re::Protocol::Goos >](#).

Definition at line 87 of file [__typeinfo.h](#).

The documentation for this class was generated from the following file:

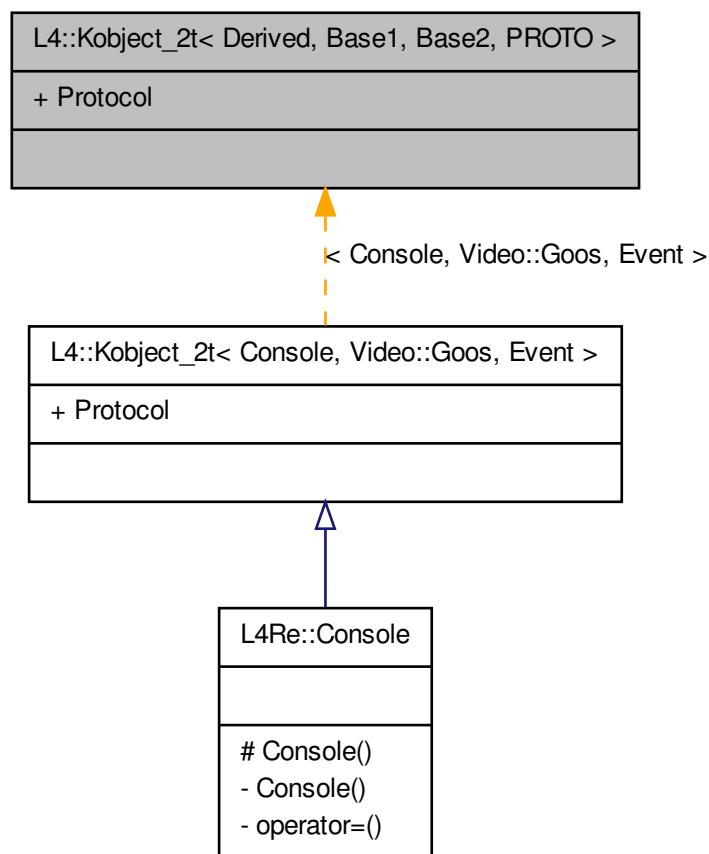
- [l4/sys/capability](#)

11.86 L4::Kobject_2t< Derived, Base1, Base2, PROTO > Class Template Reference

Helper class to create an [L4Re](#) interface class that is derived from two base classes.

```
#include <__typeinfo.h>
```

Inheritance diagram for L4::Kobject_2t< Derived, Base1, Base2, PROTO >:



Friends

- template<typename T >
`Type_info` const * `kobject_typeid ()`

Get the [L4::Type_info](#) for the [L4Re](#) interface given in T.

11.86.1 Detailed Description

```
template<typename Derived, typename Base1, typename Base2, long PROTO = 0> class
L4::Kobject_2t< Derived, Base1, Base2, PROTO >
```

Helper class to create an [L4Re](#) interface class that is derived from two base classes.

Parameters

Derived is the name of the new interface.

Base1 is the name of the interfaces first base class.

Base2 is the name of the interfaces second base class.

PROTO may be set to the statically assigned protocol number used to communicate with this interface.

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 [175](#) of file [__typeinfo.h](#).

11.86.2 Friends And Related Function Documentation

11.86.2.1 template<typename Derived, typename Base1, typename Base2, long PROTO = 0> template<typename T > Type_info const* kobject_typeid() [friend]

Get the [L4::Type_info](#) for the [L4Re](#) interface given in *T*.

Parameters

T 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 [87](#) of file [__typeinfo.h](#).

The documentation for this class was generated from the following file:

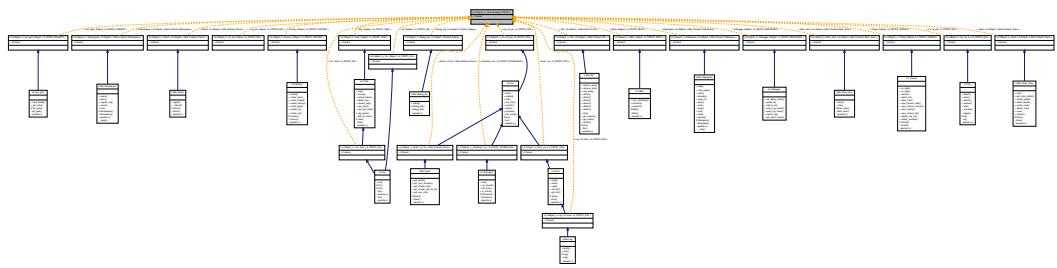
- [l4/sys/__typeinfo.h](#)

11.87 L4::Kobject_t< Derived, Base, PROTO > Class Template Reference

Helper class to create an [L4Re](#) interface class that is derived from a single base class.

```
#include <__typeinfo.h>
```

Inheritance diagram for L4::Kobject_t< Derived, Base, PROTO >:



Friends

- template<typename T >
Type_info const * **kobject_typeid** ()
Get the L4::Type_info for the L4Re interface given in T.

11.87.1 Detailed Description

template<typename Derived, typename Base, long PROTO = 0> class L4::Kobject_t< Derived, Base, PROTO >

Helper class to create an L4Re interface class that is derived from a single base class.

Parameters

- Derived** is the name of the new interface.
- Base** is the name of the interfaces single base class.
- PROTO** may be set to the statically assigned protocol number used to communicate with this interface.

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>
{
    ...
};
```

Definition at line 137 of file [__typeinfo.h](#).

11.87.2 Friends And Related Function Documentation

11.87.2.1 template<typename Derived, typename Base, long PROTO = 0> template<typename T > Type_info const* kobject_typeid () [friend]

Get the L4::Type_info for the L4Re interface given in T.

Parameters

- T** The type (L4Re interface) for which the information shall be returned.

Returns

A pointer to the [L4::Type_info](#) structure for T .

Reimplemented in [L4::Kobject_t< Event, L4::Icu, L4Re::Protocol::Event >](#), [L4::Kobject_t< Log, L4::Vcon, L4_PROTO_LOG >](#), [L4::Kobject_t< Vcon, Icu, L4_PROTO_LOG >](#), [L4::Kobject_t< Vm, Task, L4_PROTO_VM >](#), [L4::Kobject_t< Irq, Irq_eio, L4_PROTO_IRQ >](#), [L4::Kobject_t< Scheduler, Icu, L4_PROTO_SCHEDULER >](#), and [L4::Kobject_t< Icu, Irq_eio, L4_PROTO_IRQ >](#).

Definition at line 87 of file [__typeinfo.h](#).

The documentation for this class was generated from the following file:

- [l4/sys/__typeinfo.h](#)

11.88 l4_buf_regs_t Struct Reference

Encapsulation of the buffer-registers block in the UTCB.

```
#include <l4/sys/utcbl.h>
```

Data Fields

- [l4_umword_t bdr](#)
Buffer descriptor.
- [l4_umword_t br](#) [L4_UTCB_GENERIC_BUFFERS_SIZE]
Buffer registers.

11.88.1 Detailed Description

Encapsulation of the buffer-registers block in the UTCB.

Definition at line 95 of file [utcbl.h](#).

The documentation for this struct was generated from the following file:

- [l4/sys/utcbl.h](#)

11.89 l4_exc_regs_t Struct Reference

UTCB structure for exceptions.

```
#include <utcbl.h>
```

Data Fields

- [l4_umword_t pfa](#)
page fault address

- `l4_umword_t err`
error code
- `l4_umword_t tpidruro`
Thread-ID register.
- `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 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 ip`
instruction pointer
- `l4_umword_t dummy1`
dummy
- `l4_umword_t flags`
rflags
- `l4_umword_t ss`
stack segment 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`

- **l4_umword_t edx**
edx register
- **l4_umword_t ecx**
ecx register
- **l4_umword_t eax**
eax register

11.89.1 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 58 of file [utcb.h](#).

11.89.2 Field Documentation

11.89.2.1 l4_umword_t l4_exc_regs_t::flags

rflags

eflags

Definition at line 80 of file [utcb.h](#).

The documentation for this struct was generated from the following files:

- [arm/l4/sys/utcb.h](#)
- [amd64/l4/sys/utcb.h](#)
- [x86/l4/sys/utcb.h](#)

11.90 l4_fpage_t Union Reference

[L4](#) flexpage type.

```
#include <__l4_fpage.h>
```

Data Fields

- **l4_umword_t fpage**
Raw value.
- **l4_umword_t raw**
Raw value.

11.90.1 Detailed Description

[L4](#) flexpage type.

Definition at line 78 of file [__l4_fpage.h](#).

The documentation for this union was generated from the following file:

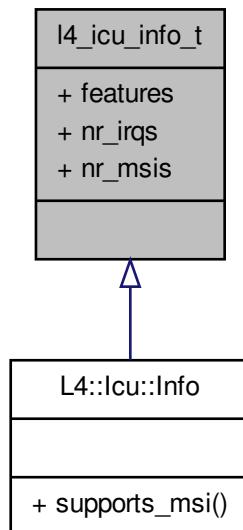
- [l4/sys/__l4_fpage.h](#)

11.91 l4_icu_info_t Struct Reference

Info structure for an ICU.

```
#include <icu.h>
```

Inheritance 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.,

11.91.1 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 [148](#) of file [icu.h](#).

11.91.2 Field Documentation

11.91.2.1 `unsigned l4_icu_info_t::features`

Feature flags.

If [L4_ICU_FLAG_MSI](#) is set the ICU supports MSIs.

Definition at line [155](#) of file [icu.h](#).

The documentation for this struct was generated from the following file:

- [l4/sys/icu.h](#)

11.92 `l4_kernel_info_mem_desc_t` Struct Reference

Memory descriptor data structure.

```
#include <memdesc.h>
```

11.92.1 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 [64](#) of file [memdesc.h](#).

The documentation for this struct was generated from the following file:

- [l4/sys/memdesc.h](#)

11.93 `l4_kernel_info_t` Struct Reference

[L4](#) Kernel Interface Page.

```
#include <__kip-32bit.h>
```

Data Fields

- `l4_uint32_t magic`
Kernel Info Page identifier ("L4tK").
- `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 [3]`
reserved
- `l4_umword_t scheduler_granularity`
for rounding time slices
- `l4_umword_t _res00 [3]`
default_kdebug_end
- `l4_umword_t sigma0_esp`
Sigma0 start stack pointer.
- `l4_umword_t sigma0_eip`
Sigma0 instruction pointer.
- `l4_umword_t _res01 [2]`
reserved
- `l4_umword_t sigma1_esp`
Sigma1 start stack pointer.
- `l4_umword_t sigma1_eip`
Sigma1 instruction pointer.
- `l4_umword_t _res02 [2]`
reserved
- `l4_umword_t root_esp`
Root task stack pointer.
- `l4_umword_t root_eip`
Root task instruction pointer.

- `l4_umword_t _res03` [2]
reserved
- `l4_umword_t _res50` [1]
reserved
- `l4_umword_t mem_info`
memory information
- `l4_umword_t _res58` [2]
reserved
- `l4_umword_t _res04` [16]
reserved
- volatile `l4_cpu_time_t clock`
L4 system clock (ts).
- `l4_umword_t _res05` [2]
reserved
- `l4_umword_t frequency_cpu`
CPU frequency in kHz.
- `l4_umword_t frequency_bus`
Bus frequency.
- `l4_umword_t _res06` [10]
reserved
- `l4_umword_t user_ptr`
user_ptr
- `l4_umword_t vhw_offset`
offset to vhw structure
- `l4_uint64_t magic`
Kernel Info Page identifier ("L4tK").
- `l4_uint64_t version`
Kernel version.
- `l4_uint8_t fill2` [7]
reserved
- `l4_uint8_t fill3` [7]
reserved
- `l4_umword_t _res_a0` [1]
reserved

- [l4_umword_t _res_b0](#) [2]
reserver

11.93.1 Detailed Description

[L4](#) Kernel Interface Page.

Examples:

[examples/sys/ux-vhw/main.c](#).

Definition at line 38 of file [_kip-32bit.h](#).

The documentation for this struct was generated from the following files:

- l4/sys/_kip-32bit.h
- l4/sys/_kip-64bit.h

11.94 l4_msg_regs_t Struct Reference

Encapsulation of the message-register block in the UTCB.

```
#include <l4/sys/utcb.h>
```

Data Fields

- [l4_umword_t mr](#) [L4_UTCB_GENERIC_DATA_SIZE]
Message registers.

11.94.1 Detailed Description

Encapsulation of the message-register block in the UTCB.

Examples:

[examples/sys/utcb-ipc/main.c](#).

Definition at line 80 of file [utcb.h](#).

The documentation for this struct was generated from the following file:

- l4/sys/utcb.h

11.95 l4_mshtag_t Struct Reference

Message tag data structure.

```
#include <types.h>
```

Public Member Functions

- long `label` () const throw ()
Get the protocol value.
- void `label` (long v) throw ()
Set the protocol value.
- unsigned `words` () const throw ()
Get the number of untyped words.
- unsigned `items` () const throw ()
Get the number of typed items.
- unsigned `flags` () const throw ()
Get the flags value.
- bool `is_page_fault` () const throw ()
Test if protocol indicates page-fault protocol.
- bool `is_preemption` () const throw ()
Test if protocol indicates preemption protocol.
- bool `is_sys_exception` () const throw ()
Test if protocol indicates system-exception protocol.
- bool `is_exception` () const throw ()
Test if protocol indicates exception protocol.
- bool `is_sigma0` () const throw ()
Test if protocol indicates sigma0 protocol.
- bool `is_io_page_fault` () const throw ()
Test if protocol indicates IO-page-fault protocol.
- unsigned `has_error` () const throw ()
Test if flags indicate an error.

Data Fields

- `l4_mword_t raw`
raw value

11.95.1 Detailed Description

Message tag data structure. `#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/clntsrv/server.cc`, `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/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 158 of file `types.h`.

11.95.2 Member Function Documentation

11.95.2.1 `unsigned l4_mshtag_t::flags () const throw () [inline]`

Get the flags value.

The flags are a combination of the flags defined by `l4_mshtag_flags`.

Definition at line 176 of file `types.h`.

References `raw`.

The documentation for this struct was generated from the following file:

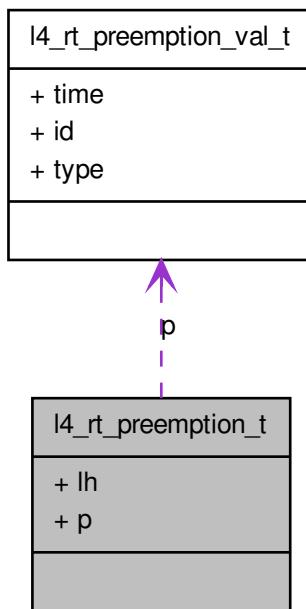
- `l4/sys/types.h`

11.96 l4_rt_preemption_t Union Reference

Struct.

```
#include <rt_sched-proto.h>
```

Collaboration diagram for l4_rt_preemption_t:



Data Fields

- l4_low_high_t `lh`
lh value
- l4_rt_preemption_val_t `p`
p value

11.96.1 Detailed Description

Struct.

Definition at line 56 of file [rt_sched-proto.h](#).

The documentation for this union was generated from the following file:

- x86/l4/sys/rt_sched-proto.h

11.97 l4_rt_preemption_val32_t Struct Reference

Struct.

```
#include <rt_sched-proto.h>
```

Data Fields

- l4_uint32_t time_high:24
time_high
- l4_uint32_t id:7
id
- l4_uint32_t type:1
type

11.97.1 Detailed Description

Struct.

Definition at line 50 of file [rt_sched-proto.h](#).

The documentation for this struct was generated from the following file:

- x86/l4/sys/rt_sched-proto.h

11.98 l4_rt_preemption_val_t Struct Reference

Struct.

```
#include <rt_sched-proto.h>
```

Data Fields

- l4_uint64_t time:56
time
- l4_uint64_t id:7
id
- l4_uint64_t type:1
id

11.98.1 Detailed Description

Struct.

Definition at line 44 of file [rt_sched-proto.h](#).

The documentation for this struct was generated from the following file:

- x86/l4/sys/rt_sched-proto.h

11.99 l4_sched_cpu_set_t Struct Reference

CPU sets.

```
#include <scheduler.h>
```

Data Fields

- [l4_umword_t offset](#)

First CPU of interest (must be aligned to 2^{\wedge} granularity).

- [l4_umword_t map](#)

Bitmap of online CPUs.

- [unsigned char granularity](#)

One bit in map represents 2^{\wedge} granularity CPUs.

11.99.1 Detailed Description

CPU sets.

Examples:

[examples/sys/migrate/thread_migrate.cc](#).

Definition at line [40](#) of file [scheduler.h](#).

The documentation for this struct was generated from the following file:

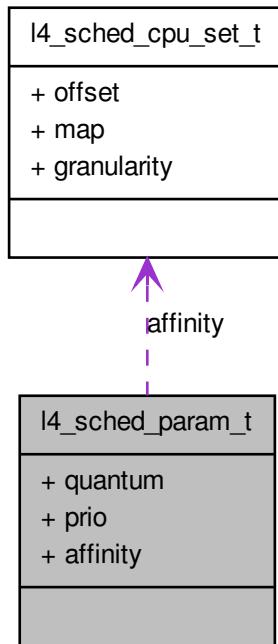
- [l4/sys/scheduler.h](#)

11.100 l4_sched_param_t Struct Reference

Scheduler parameter set.

```
#include <scheduler.h>
```

Collaboration diagram for l4_sched_param_t:



Data Fields

- [l4_cpu_time_t quantum](#)
Timeslice in micro seconds.
- [unsigned prio](#)
Priority for scheduling.
- [l4_sched_cpu_set_t affinity](#)
CPU affinity.

11.100.1 Detailed Description

Scheduler parameter set.

Examples:

[examples/sys/migrate/thread_migrate.cc](#).

Definition at line [101](#) of file [scheduler.h](#).

The documentation for this struct was generated from the following file:

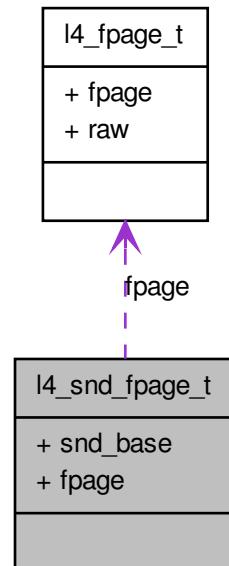
- l4/sys/scheduler.h

11.101 l4_snd_fpage_t Struct Reference

Send-flex-page 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 flex-page descriptor.

11.101.1 Detailed Description

Send-flex-page types.

Definition at line 95 of file [__l4_fpage.h](#).

The documentation for this struct was generated from the following file:

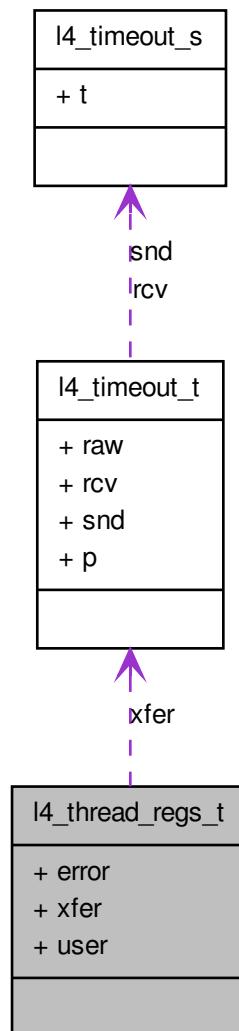
- [l4/sys/__l4_fpage.h](#)

11.102 l4_thread_regs_t Struct Reference

Encapsulation of the thread-control-register block of the UTCB.

```
#include <l4/sys/utcbl.h>
```

Collaboration diagram for l4_thread_regs_t:



Data Fields

- [l4_umword_t error](#)
System call error codes.
- [l4_timeout_t xfer](#)
Message transfer timeout.
- [l4_umword_t user](#) [3]
User values (ignored and preserved by the kernel).

11.102.1 Detailed Description

Encapsulation of the thread-control-register block of the UTCB.

Definition at line [113](#) of file [utcb.h](#).

The documentation for this struct was generated from the following file:

- [l4/sys/utcb.h](#)

11.103 l4_timeout_s Struct Reference

Basic timeout specification.

```
#include <__timeout.h>
```

Data Fields

- [l4_uint16_t t](#)
timeout value

11.103.1 Detailed Description

Basic timeout specification. Basically a floating point number with 10 bits mantissa and 5 bits exponent ($t = m \cdot 2^e$).

The timeout can also specify an absolute point in time (bit 16 == 1).

Definition at line [45](#) of file [__timeout.h](#).

The documentation for this struct was generated from the following file:

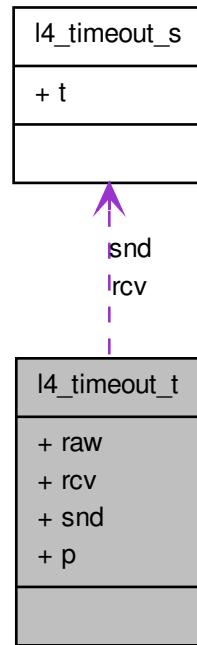
- [l4/sys/__timeout.h](#)

11.104 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

11.104.1 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 [57](#) of file `__timeout.h`.

The documentation for this union was generated from the following file:

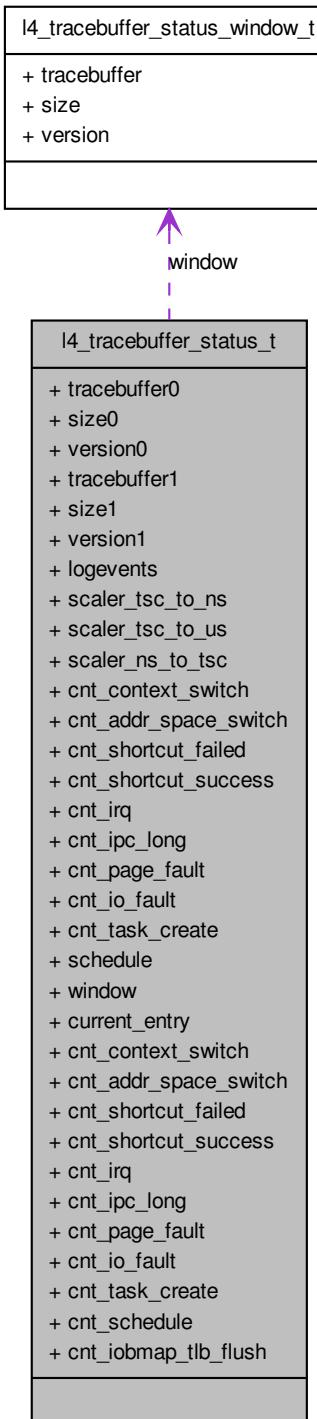
- `l4/sys/__timeout.h`

11.105 `l4_tracebuffer_status_t` Struct Reference

Trace buffer status.

```
#include <ktrace.h>
```

Collaboration diagram for l4_tracebuffer_status_t:



Data Fields

- `l4_umword_t tracebuffer0`
Address of trace buffer 0.
- `l4_umword_t size0`
Size of trace buffer 0.
- `l4_umword_t version0`
Version number of trace buffer 0 (incremented if tb0 overruns).
- `l4_umword_t tracebuffer1`
Address of trace buffer 1 (there is no gap between tb0 and tb1).
- `l4_umword_t size1`
Size of trace buffer 1 (same as tb0).
- `l4_umword_t version1`
Version number of trace buffer 1 (incremented if tb1 overruns).
- `l4_umword_t logevents [16]`
Available LOG events.
- `l4_umword_t scaler_tsc_to_ns`
Scaler used for translation of CPU cycles to nano seconds.
- `l4_umword_t scaler_tsc_to_us`
Scaler used for translation of CPU cycles to micro seconds.
- `l4_umword_t scaler_ns_to_tsc`
Scaler used for translation of nano seconds to CPU cycles.
- `l4_umword_t cnt_context_switch`
Number of context switches (intra AS or inter AS).
- `l4_umword_t cnt_addr_space_switch`
Number of inter AS context switches.
- `l4_umword_t cnt_shortcut_failed`
How often was the IPC shortcut not taken.
- `l4_umword_t cnt_shortcut_success`
How often was the IPC shortcut taken.
- `l4_umword_t cnt_irq`
Number of hardware interrupts (without kernel scheduling interrupt).
- `l4_umword_t cnt_ipc_long`
Number of long IPCs.

- **l4_umword_t cnt_page_fault**
Number of page faults.
- **l4_umword_t cnt_io_fault**
Number of faults (application runs at IOPL 0 and tries to execute cli, sti, in, or out but does not have a sufficient right in the I/O bitmap).
- **l4_umword_t cnt_task_create**
Number of tasks created.
- **l4_umword_t schedule**
Number of reschedules.
- **volatile l4_tracebuffer_entry_t * current_entry**
Address of the most current event in trace-buffer.
- **volatile l4_umword_t cnt_context_switch**
Number of context switches (intra AS or inter AS).
- **volatile l4_umword_t cnt_addr_space_switch**
Number of inter AS context switches.
- **volatile l4_umword_t cnt_shortcut_failed**
How often was the IPC shortcut taken.
- **volatile l4_umword_t cnt_shortcut_success**
How often was the IPC shortcut not taken.
- **volatile l4_umword_t cnt_irq**
Number of hardware interrupts (without kernel scheduling interrupt).
- **volatile l4_umword_t cnt_ipc_long**
Number of long IPCs.
- **volatile l4_umword_t cnt_page_fault**
Number of page faults.
- **volatile l4_umword_t cnt_io_fault**
Number of faults (application runs at IOPL 0 and tries to execute cli, sti, in, or out but does not have a sufficient right in the I/O bitmap).
- **volatile l4_umword_t cnt_task_create**
Number of tasks created.
- **volatile l4_umword_t cnt_schedule**
Number of reschedules.
- **volatile l4_umword_t cnt_iobmap_tlb_flush**
Number of flushes of the I/O bitmap.

11.105.1 Detailed Description

Trace buffer status. Trace-buffer status.

Tracebuffer status.

Definition at line 55 of file [ktrace.h](#).

11.105.2 Field Documentation

11.105.2.1 l4_umword_t l4_tracebuffer_status_t::tracebuffer0

Address of trace buffer 0.

Address of tracebuffer 0.

Definition at line 58 of file [ktrace.h](#).

11.105.2.2 l4_umword_t l4_tracebuffer_status_t::size0

Size of trace buffer 0.

Size of tracebuffer 0.

Definition at line 60 of file [ktrace.h](#).

11.105.2.3 l4_umword_t l4_tracebuffer_status_t::version0

Version number of trace buffer 0 (incremented if tb0 overruns).

Version number of tracebuffer 0 (incremented if tb0 overruns).

Definition at line 62 of file [ktrace.h](#).

11.105.2.4 l4_umword_t l4_tracebuffer_status_t::tracebuffer1

Address of trace buffer 1 (there is no gap between tb0 and tb1).

Address of tracebuffer 1 (there is no gap between tb0 and tb1).

Definition at line 64 of file [ktrace.h](#).

11.105.2.5 l4_umword_t l4_tracebuffer_status_t::size1

Size of trace buffer 1 (same as tb0).

Size of tracebuffer 1 (same as tb0).

Definition at line 66 of file [ktrace.h](#).

11.105.2.6 l4_umword_t l4_tracebuffer_status_t::version1

Version number of trace buffer 1 (incremented if tb1 overruns).

Version number of tracebuffer 1 (incremented if tb1 overruns).

Definition at line 68 of file [ktrace.h](#).

11.105.2.7 volatile l4_umword_t l4_tracebuffer_status_t::cnt_iobmap_tlb_flush

Number of flushes of the I/O bitmap.

Increases on context switches between two small address spaces if at least one of the spaces has an I/O bitmap allocated.

Definition at line 99 of file [ktrace.h](#).

The documentation for this struct was generated from the following files:

- arm/l4/sys/ktrace.h
- amd64/l4f/l4/sys/ktrace.h
- x86/l4/sys/ktrace.h

11.106 l4_tracebuffer_status_window_t Struct Reference

Trace-buffer status window descriptor.

```
#include <ktrace.h>
```

Data Fields

- `l4_tracebuffer_entry_t * tracebuffer`
Address of trace-buffer.
- `l4_umword_t size`
Size of trace-buffer.
- `volatile l4_uint64_t version`
Version number of trace-buffer (incremented if trace-buffer overruns).

11.106.1 Detailed Description

Trace-buffer status window descriptor.

Definition at line 45 of file [ktrace.h](#).

The documentation for this struct was generated from the following file:

- x86/l4/sys/ktrace.h

11.107 l4_vcon_attr_t Struct Reference

Vcon attribute structure.

```
#include <vcon.h>
```

Data Fields

- [l4_umword_t i_flags](#)

input flags

- [l4_umword_t o_flags](#)

output flags

- [l4_umword_t l_flags](#)

local flags

11.107.1 Detailed Description

Vcon attribute structure.

Definition at line [115](#) of file [vcon.h](#).

The documentation for this struct was generated from the following file:

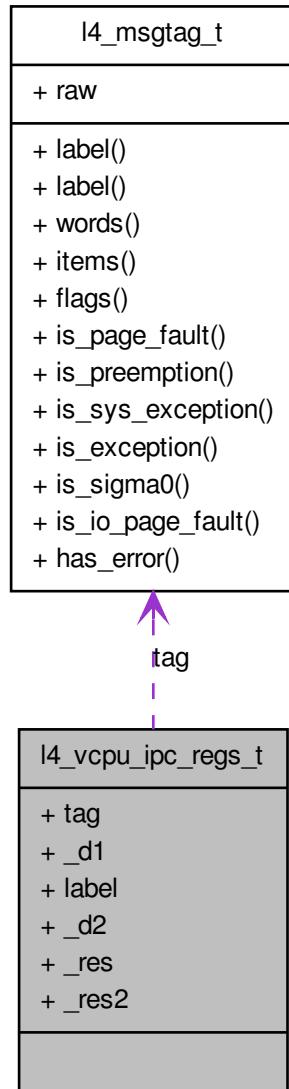
- [l4/sys/vcon.h](#)

11.108 l4_vcpu_ipc_regs_t Struct Reference

vCPU message registers.

```
#include <__vcpu-arch.h>
```

Collaboration diagram for l4_vcpu_ipc_regs_t:



11.108.1 Detailed Description

vCPU message registers.

Definition at line 46 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

11.109 l4_vcpu_regs_t Struct Reference

vCPU registers.

```
#include <____vcpu-arch.h>
```

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 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 dummy1`
dummy
- `l4_umword_t ss`
ss register
- `l4_umword_t es`
gs register
- `l4_umword_t ds`
fs register
- `l4_umword_t gs`
gs register
- `l4_umword_t fs`
fs register

11.109.1 Detailed Description

vCPU registers.

Definition at line 27 of file [__vcpu-arch.h](#).

11.109.2 Field Documentation

11.109.2.1 l4_umword_t l4_vcpu_regs_t::di

rdi register

edi register

Definition at line 44 of file [__vcpu-arch.h](#).

11.109.2.2 l4_umword_t l4_vcpu_regs_t::si

rsi register

esi register

Definition at line 45 of file [__vcpu-arch.h](#).

11.109.2.3 l4_umword_t l4_vcpu_regs_t::bp

rbp register

ebp register

Definition at line 46 of file [__vcpu-arch.h](#).

11.109.2.4 l4_umword_t l4_vcpu_regs_t::bx

rbx register

ebx register

Definition at line 48 of file [__vcpu-arch.h](#).

11.109.2.5 l4_umword_t l4_vcpu_regs_t::dx

rdx register

edx register

Definition at line 49 of file [__vcpu-arch.h](#).

11.109.2.6 l4_umword_t l4_vcpu_regs_t::cx

rcx register

ecx register

Definition at line 50 of file [__vcpu-arch.h](#).

11.109.2.7 l4_umword_t l4_vcpu_regs_t::ax

rax register

eax register

Definition at line 51 of file [__vcpu-arch.h](#).

The documentation for this struct was generated from the following files:

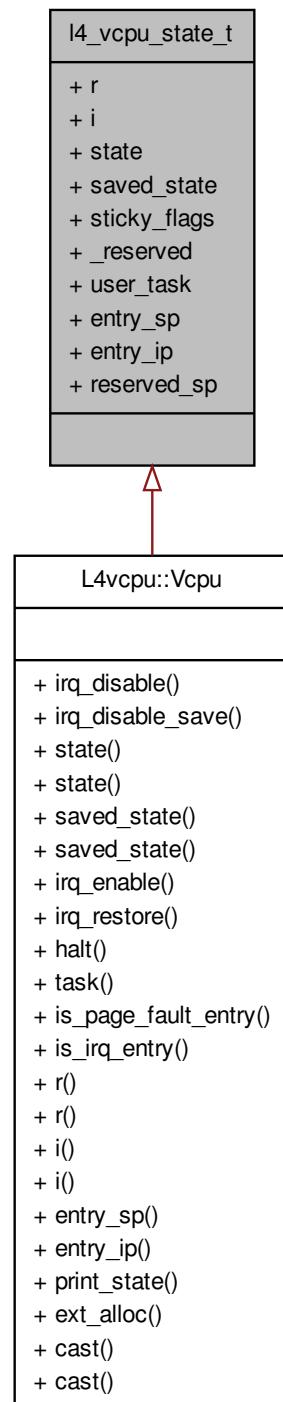
- arm/l4/sys/__vcpu-arch.h
- amd64/l4/sys/__vcpu-arch.h
- x86/l4/sys/__vcpu-arch.h

11.110 l4_vcpu_state_t Struct Reference

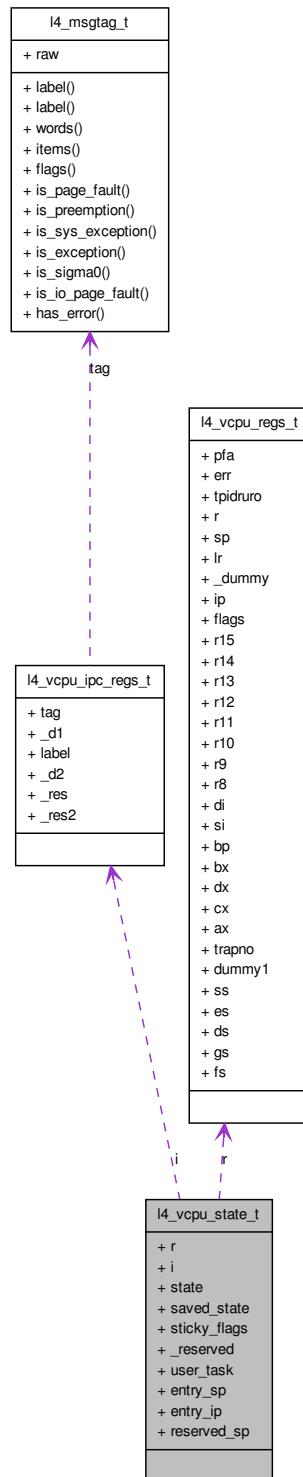
State of a vCPU.

```
#include <vcpu.h>
```

Inheritance diagram for l4_vcpu_state_t:



Collaboration diagram for l4_vcpu_state_t:



Data Fields

- `l4_vcpu_regs_t r`

Register state.

- `l4_vcpu_ipc_regs_t i`

IPC state.

- `l4_uint16_t state`

Current vCPU state.

- `l4_uint16_t saved_state`

Saved vCPU state.

- `l4_uint16_t sticky_flags`

Pending 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.

11.110.1 Detailed Description

State of a vCPU.

Definition at line 34 of file `vcpu.h`.

The documentation for this struct was generated from the following file:

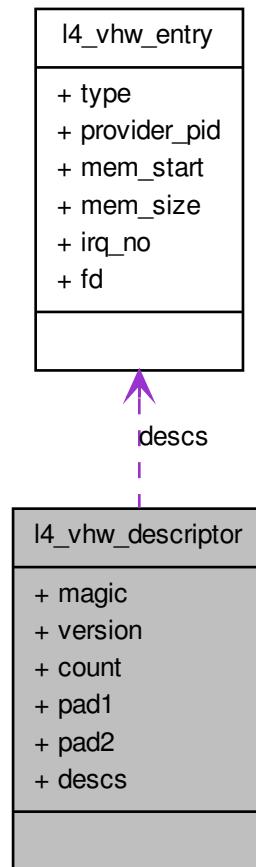
- `l4/sys/vcpu.h`

11.111 l4_vhw_descriptor Struct Reference

Virtual hardware devices description.

```
#include <vhw.h>
```

Collaboration diagram for l4_vhw_descriptor:



Data Fields

- `l4_uint32_t` `magic`
Magic.
- `l4_uint8_t` `version`
Version of the descriptor.
- `l4_uint8_t` `count`
Number of entries.
- `l4_uint8_t` `pad1`
padding

- `l4_uint8_t pad2`
padding
- `struct l4_vhw_entry_descs []`
Array of device descriptions.

11.111.1 Detailed Description

Virtual hardware devices description.

Examples:

`examples/sys/ux-vhw/main.c.`

Definition at line [70](#) of file `vhw.h`.

11.111.2 Field Documentation

11.111.2.1 `l4_uint32_t l4_vhw_descriptor::magic`

Magic.

Examples:

`examples/sys/ux-vhw/main.c.`

Definition at line [71](#) of file `vhw.h`.

11.111.2.2 `l4_uint8_t l4_vhw_descriptor::version`

Version of the descriptor.

Examples:

`examples/sys/ux-vhw/main.c.`

Definition at line [72](#) of file `vhw.h`.

11.111.2.3 `l4_uint8_t l4_vhw_descriptor::count`

Number of entries.

Examples:

`examples/sys/ux-vhw/main.c.`

Definition at line [73](#) of file `vhw.h`.

11.111.2.4 struct l4_vhw_entry l4_vhw_descriptor::descs[]

Array of device descriptions.

Definition at line 77 of file [vhw.h](#).

The documentation for this struct was generated from the following file:

- [l4/sys/vhw.h](#)

11.112 l4_vhw_entry Struct Reference

Description of a device.

```
#include <vhw.h>
```

Data Fields

- enum [l4_vhw_entry_type](#) type
Type of virtual hardware.
- [l4_uint32_t](#) provider_pid
Host PID of the VHW provider.
- [l4_addr_t](#) mem_start
Start of memory region.
- [l4_addr_t](#) mem_size
Size of memory region.
- [l4_uint32_t](#) irq_no
IRQ number.
- [l4_uint32_t](#) fd
File descriptor.

11.112.1 Detailed Description

Description of a device.

Examples:

[examples/sys/ux-vhw/main.c](#).

Definition at line 55 of file [vhw.h](#).

11.112.2 Field Documentation

11.112.2.1 enum l4_vhw_entry_type l4_vhw_entry::type

Type of virtual hardware.

Examples:

[examples/sys/ux-vhw/main.c](#).

Definition at line [56](#) of file [vhw.h](#).

11.112.2.2 l4_uint32_t l4_vhw_entry::provider_pid

Host PID of the VHW provider.

Examples:

[examples/sys/ux-vhw/main.c](#).

Definition at line [57](#) of file [vhw.h](#).

11.112.2.3 l4_addr_t l4_vhw_entry::mem_start

Start of memory region.

Examples:

[examples/sys/ux-vhw/main.c](#).

Definition at line [59](#) of file [vhw.h](#).

11.112.2.4 l4_addr_t l4_vhw_entry::mem_size

Size of memory region.

Examples:

[examples/sys/ux-vhw/main.c](#).

Definition at line [60](#) of file [vhw.h](#).

11.112.2.5 l4_uint32_t l4_vhw_entry::irq_no

IRQ number.

Examples:

[examples/sys/ux-vhw/main.c](#).

Definition at line [62](#) of file [vhw.h](#).

11.112.2.6 l4_uint32_t l4_vhw_entry::fd

File descriptor.

Definition at line 63 of file [vhw.h](#).

The documentation for this struct was generated from the following file:

- l4/sys/vhw.h

11.113 l4_vm_state Struct Reference

state structure for TrustZone VMs

```
#include <vm.h>
```

11.113.1 Detailed Description

state structure for TrustZone VMs

Definition at line 38 of file [vm.h](#).

The documentation for this struct was generated from the following file:

- arm/l4/sys/vm.h

11.114 l4_vm_svm_vmcb_control_area Struct Reference

VMCB structure for SVM VMs.

```
#include <__vm-svm.h>
```

11.114.1 Detailed Description

VMCB structure for SVM VMs.

Definition at line 40 of file [__vm-svm.h](#).

The documentation for this struct was generated from the following file:

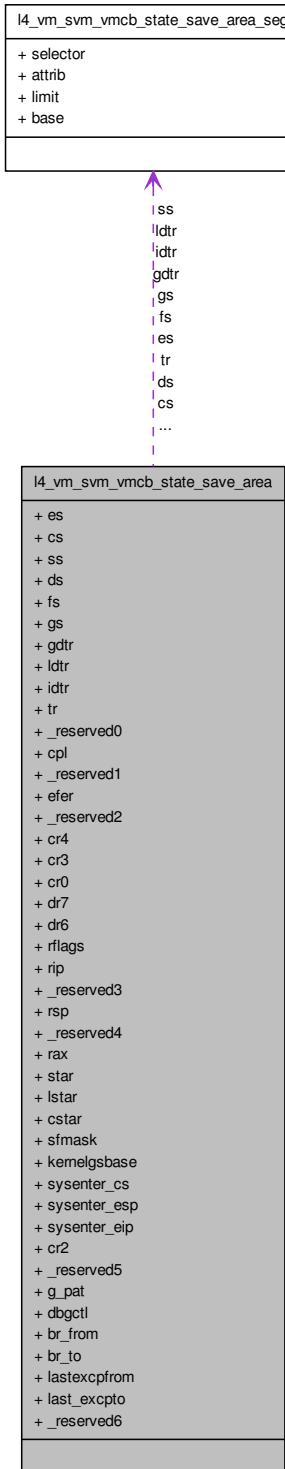
- l4/sys/__vm-svm.h

11.115 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:



11.115.1 Detailed Description

State save area structure for SVM VMs.

Definition at line [92](#) of file [__vm-svm.h](#).

The documentation for this struct was generated from the following file:

- l4/sys/__vm-svm.h

11.116 l4_vm_svm_vmcb_state_save_area_seg Struct Reference

State save area segment selector struct.

```
#include <__vm-svm.h>
```

11.116.1 Detailed Description

State save area segment selector struct.

Definition at line [80](#) of file [__vm-svm.h](#).

The documentation for this struct was generated from the following file:

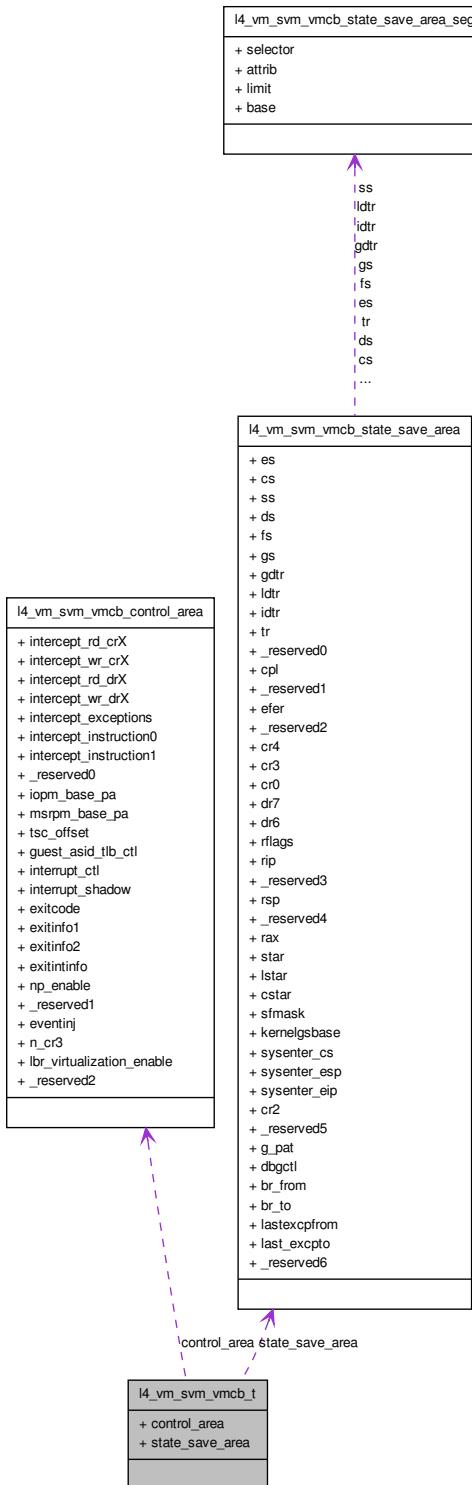
- l4/sys/__vm-svm.h

11.117 l4_vm_svm_vmcb_t Struct Reference

Control structure for SVM VMs.

```
#include <__vm-svm.h>
```

Collaboration diagram for l4_vm_svm_vmcb_t:



11.117.1 Detailed Description

Control structure for SVM VMs.

Definition at line 157 of file [__vm-svm.h](#).

The documentation for this struct was generated from the following file:

- l4/sys/__vm-svm.h

11.118 l4re_aux_t Struct Reference

Auxiliary descriptor.

```
#include <l4aux.h>
```

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`.

11.118.1 Detailed Description

Auxiliary descriptor.

Definition at line 51 of file [l4aux.h](#).

The documentation for this struct was generated from the following file:

- l4/re/l4aux.h

11.119 l4re_ds_stats_t Struct Reference

Information about the data space.

```
#include <dataspace.h>
```

Data Fields

- `unsigned long size`

size

- unsigned long [flags](#)

flags

11.119.1 Detailed Description

Information about the data space.

Definition at line [45](#) of file [dataspace.h](#).

The documentation for this struct was generated from the following file:

- [l4/re/c/dataspace.h](#)

11.120 l4re_elf_aux_mword_t Struct Reference

Auxiliary vector element for a single unsigned data word.

```
#include <elf_aux.h>
```

11.120.1 Detailed Description

Auxiliary vector element for a single unsigned data word.

Definition at line [125](#) of file [elf_aux.h](#).

The documentation for this struct was generated from the following file:

- [l4/re/elf_aux.h](#)

11.121 l4re_elf_aux_t Struct Reference

Generic header for each auxiliary vector element.

```
#include <elf_aux.h>
```

11.121.1 Detailed Description

Generic header for each auxiliary vector element.

Definition at line [105](#) of file [elf_aux.h](#).

The documentation for this struct was generated from the following file:

- [l4/re/elf_aux.h](#)

11.122 l4re_elf_aux_vma_t Struct Reference

Auxiliary vector element for a reserved virtual memory area.

```
#include <elf_aux.h>
```

11.122.1 Detailed Description

Auxiliary vector element for a reserved virtual memory area.

Definition at line 114 of file [elf_aux.h](#).

The documentation for this struct was generated from the following file:

- [14/re/elf_aux.h](#)

11.123 l4re_env_cap_entry_t Struct Reference

Entry in the [L4Re](#) environment array for the named initial objects.

```
#include <env.h>
```

Public Member Functions

- [l4re_env_cap_entry_t \(\)](#)
Create an invalid entry.
- [l4re_env_cap_entry_t \(char const *n, l4_cap_idx_t c, l4_umword_t f=0\)](#)
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](#)
Some flags for the object.
- [char name \[16\]](#)
The name of the object.

11.123.1 Detailed Description

Entry in the [L4Re](#) environment array for the named initial objects.

Definition at line 35 of file [env.h](#).

11.123.2 Constructor & Destructor Documentation

11.123.2.1 `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

n is the name of the initial object.

c is the capability selector that refers the initial object.

f are the additional flags for the object.

Definition at line 67 of file [env.h](#).

References [name](#).

11.123.3 Field Documentation

11.123.3.1 `l4_umword_t l4re_env_cap_entry_t::flags`

Some flags for the object.

Note

Currently unused.

Definition at line 46 of file [env.h](#).

Referenced by [l4re_env_get_cap_l\(\)](#).

The documentation for this struct was generated from the following file:

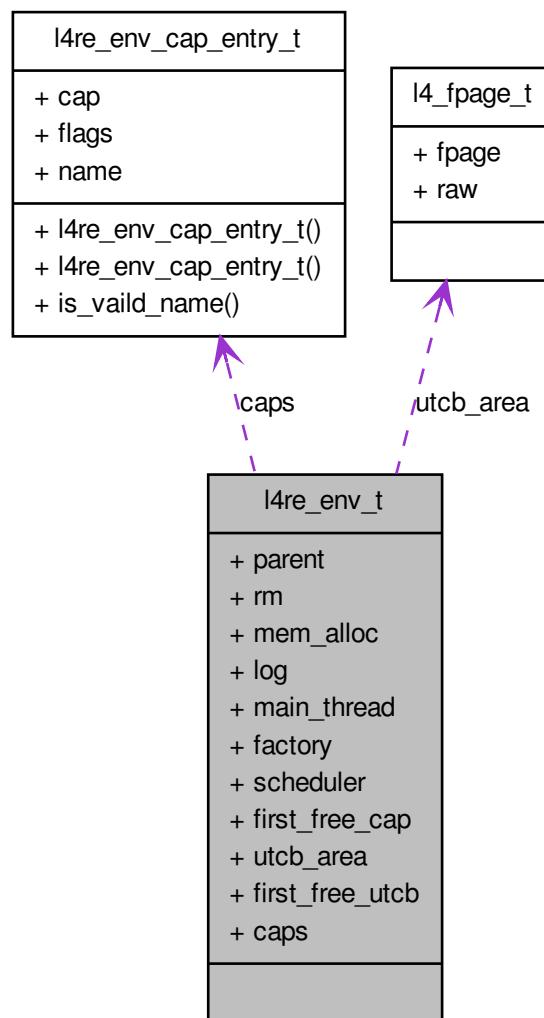
- [l4/re/env.h](#)

11.124 `l4re_env_t` Struct Reference

Initial Environment structure (C version).

```
#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 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.

11.124.1 Detailed Description

Initial Environment structure (C version).

See also

[Initial environment](#)

Definition at line [96](#) of file [env.h](#).

The documentation for this struct was generated from the following file:

- [l4/re/env.h](#)

11.125 l4re_event_t Struct Reference

Event structure used in buffer.

```
#include <event.h>
```

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.

11.125.1 Detailed Description

Event structure used in buffer.

Definition at line 40 of file `event.h`.

The documentation for this struct was generated from the following file:

- `l4/re/c/event.h`

11.126 l4re_video_color_component_t Struct Reference

Color component structure.

```
#include <colors.h>
```

Data Fields

- unsigned char `size`

Size in bits.

- unsigned char `shift`

offset in pixel

11.126.1 Detailed Description

Color component structure.

Definition at line 29 of file `colors.h`.

The documentation for this struct was generated from the following file:

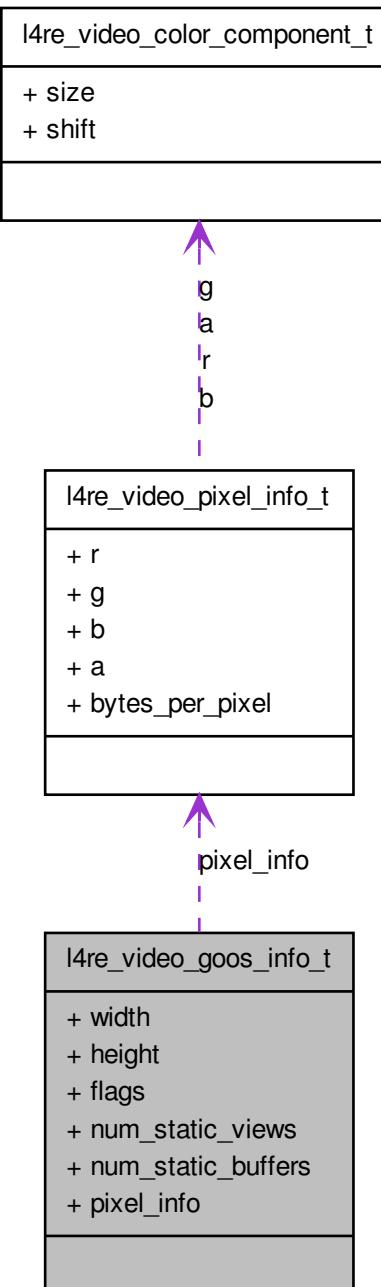
- `l4/re/c/video/colors.h`

11.127 l4re_video_goos_info_t Struct Reference

Goos information structure.

```
#include <goos.h>
```

Collaboration diagram for l4re_video_goops_info_t:



Data Fields

- unsigned long [width](#)

Width of the goos.

- unsigned long [height](#)

Height of the goos.

- unsigned [flags](#)

Flags of the framebuffer.

- 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.

11.127.1 Detailed Description

Goos information structure.

Definition at line 51 of file [goos.h](#).

The documentation for this struct was generated from the following file:

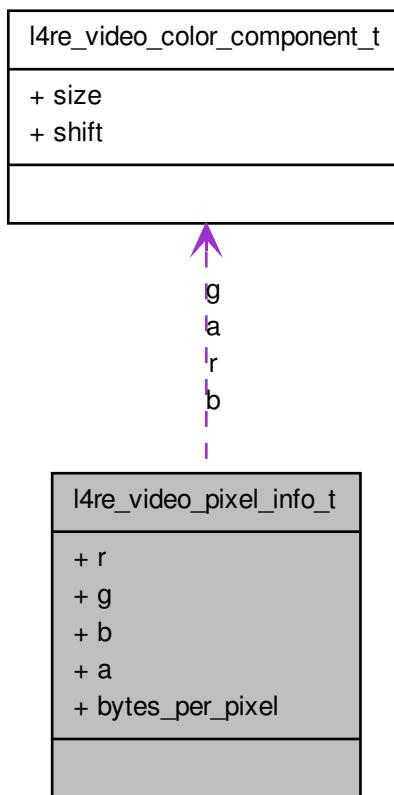
- [l4/re/c/video/goos.h](#)

11.128 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.

11.128.1 Detailed Description

Pixel_info structure.

Definition at line 39 of file [colors.h](#).

The documentation for this struct was generated from the following file:

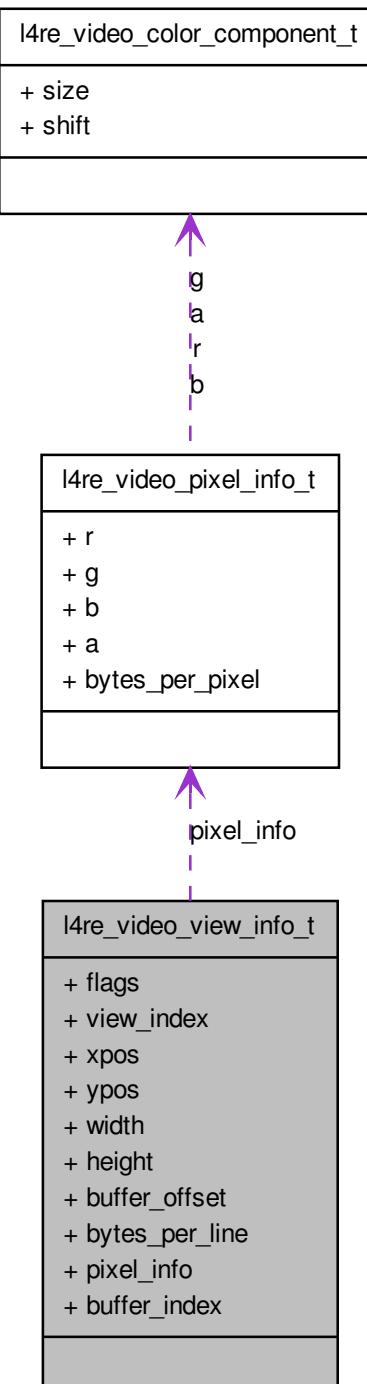
- l4/re/c/video/colors.h

11.129 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.

11.129.1 Detailed Description

View information structure.

Definition at line 59 of file `view.h`.

The documentation for this struct was generated from the following file:

- `l4/re/c/video/view.h`

11.130 l4re_video_view_t Struct Reference

C representation of a goos view.

```
#include <view.h>
```

11.130.1 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 78 of file `view.h`.

The documentation for this struct was generated from the following file:

- `l4/re/c/video/view.h`

11.131 l4util_idt_desc_t Struct Reference

IDT entry.

```
#include <idt.h>
```

Data Fields

- [l4_uint64_t b](#)

see Intel doc

- [l4_uint32_t b](#)

see Intel doc

11.131.1 Detailed Description

IDT entry.

Definition at line [33](#) of file [idt.h](#).

The documentation for this struct was generated from the following files:

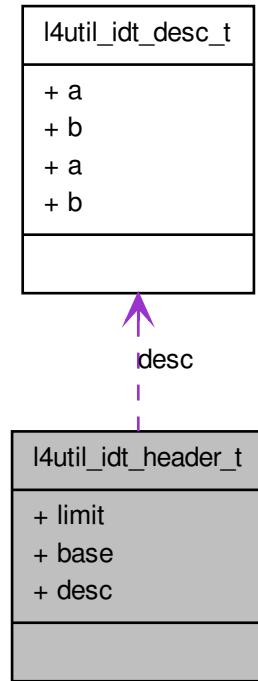
- [amd64/l4/util/idt.h](#)
- [x86/l4/util/idt.h](#)

11.132 l4util_idt_header_t Struct Reference

Header of an IDT table.

```
#include <idt.h>
```

Collaboration diagram for l4util_idt_header_t:



Data Fields

- `l4_uint16_t limit`
limit field (see Intel doc)
- `void * base`
idt base (see Intel doc)

11.132.1 Detailed Description

Header of an IDT table.

Definition at line 40 of file [idt.h](#).

The documentation for this struct was generated from the following files:

- [amd64/l4/util/idt.h](#)
- [x86/l4/util/idt.h](#)

11.133 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>
```

Data Fields

- `l4_uint64_t addr`
<Size of structure
- `l4_uint64_t size`
<Start address
- `l4_uint32_t type`
<Size of memory range

11.133.1 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 [43](#) of file [mb_info.h](#).

The documentation for this struct was generated from the following file:

- `l4/util/mb_info.h`

11.134 l4util_mb_apm_t Struct Reference

APM BIOS info.

```
#include <mb_info.h>
```

11.134.1 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`

11.135 l4util_mb_drive_t Struct Reference

Drive Info structure.

```
#include <mb_info.h>
```

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

11.135.1 Detailed Description

Drive Info structure.

Definition at line [73](#) of file `mb_info.h`.

11.135.2 Field Documentation

11.135.2.1 l4_uint8_t l4util_mb_drive_t::drive_number

<The size of this structure.

Definition at line [76](#) of file `mb_info.h`.

11.135.2.2 l4_uint8_t l4util_mb_drive_t::drive_mode

<The BIOS drive number.

Definition at line [77](#) of file `mb_info.h`.

11.135.2.3 l4_uint16_t l4util_mb_drive_t::drive_cylinders

<The access mode (see below).

Definition at line 78 of file [mb_info.h](#).

The documentation for this struct was generated from the following file:

- [l4util\(mb_info.h\)](#)

11.136 l4util_mb_info_t Struct Reference

```
#include <mb_info.h>
```

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.
- `l4_uint32_t tabsize`
(a.out) Kernel symbol table info
- `l4_uint32_t num`
(ELF) Kernel section header table

11.136.1 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 202 of file `mb_info.h`.

The documentation for this struct was generated from the following file:

- `l4/util/mb_info.h`

11.137 l4util_mb_mod_t Struct Reference

```
#include <mb_info.h>
```

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

11.137.1 Detailed Description

The structure type "mod_list" is used by the `multiboot_info` structure.

Definition at line 27 of file `mb_info.h`.

11.137.2 Field Documentation

11.137.2.1 `l4_uint32_t l4util_mb_mod_t::mod_start`

Starting address of module in memory.

Definition at line 29 of file `mb_info.h`.

11.137.2.2 `l4_uint32_t l4util_mb_mod_t::mod_end`

End address of module in memory.

Definition at line 30 of file `mb_info.h`.

The documentation for this struct was generated from the following file:

- `l4/util/mb_info.h`

11.138 `l4util_mb_vbe_ctrl_t` Struct Reference

VBE controller information.

```
#include <mb_info.h>
```

11.138.1 Detailed Description

VBE controller information.

Definition at line 104 of file `mb_info.h`.

The documentation for this struct was generated from the following file:

- l4/util/mb_info.h

11.139 l4util_mb_vbe_mode_t Struct Reference

VBE mode information.

```
#include <mb_info.h>
```

Data Fields

all VESA versions

- l4_uint16_t mode_attributes
- l4_uint8_t win_a_attributes
- l4_uint8_t win_b_attributes
- l4_uint16_t win_granularity
- l4_uint16_t win_size
- l4_uint16_t win_a_segment
- l4_uint16_t win_b_segment
- l4_uint32_t win_func
- l4_uint16_t bytes_per_scanline

>= VESA version 1.2

- l4_uint16_t x_resolution
- l4_uint16_t y_resolution
- l4_uint8_t x_char_size
- l4_uint8_t y_char_size
- l4_uint8_t number_of_planes
- l4_uint8_t bits_per_pixel
- l4_uint8_t number_of_banks
- l4_uint8_t memory_model
- l4_uint8_t bank_size
- l4_uint8_t number_of_image_pages
- l4_uint8_t reserved0

direct color

- l4_uint8_t red_mask_size
- l4_uint8_t red_field_position
- l4_uint8_t green_mask_size
- l4_uint8_t green_field_position
- l4_uint8_t blue_mask_size
- l4_uint8_t blue_field_position
- l4_uint8_t reserved_mask_size
- l4_uint8_t reserved_field_position
- l4_uint8_t direct_color_mode_info

>= VESA version 2.0

- l4_uint32_t phys_base
- l4_uint32_t reserved1
- l4_uint16_t reversed2

>= VESA version 3.0

- `l4_uint16_t linear_bytes_per_scanline`
- `l4_uint8_t banked_number_of_image_pages`
- `l4_uint8_t linear_number_of_image_pages`
- `l4_uint8_t linear_red_mask_size`
- `l4_uint8_t linear_red_field_position`
- `l4_uint8_t linear_green_mask_size`
- `l4_uint8_t linear_green_field_position`
- `l4_uint8_t linear_blue_mask_size`
- `l4_uint8_t linear_blue_field_position`
- `l4_uint8_t linear_reserved_mask_size`
- `l4_uint8_t linear_reserved_field_position`
- `l4_uint32_t max_pixel_clock`
- `l4_uint8_t reserved3 [189+1]`

11.139.1 Detailed Description

VBE mode information.

Definition at line 122 of file `mb_info.h`.

The documentation for this struct was generated from the following file:

- `l4/util/mb_info.h`

11.140 `cxx::List< D, Alloc >` Class Template Reference

Doubly linked list, with internal allocation.

Data Structures

- class `Iter`

Iterator.

Public Member Functions

- `void push_back (D const &d) throw ()`
Add element at the end of the list.
- `void push_front (D const &d) throw ()`
Add element at the beginning of the list.
- `void remove (Iter const &i) throw ()`
Remove element pointed to by the iterator.
- `unsigned long size () const throw ()`
Get the length of the list.
- `D const & operator[] (unsigned long idx) const throw ()`
Random access.

- `D & operator[]` (unsigned long idx) throw ()

Random access.

- `Iter items ()` throw ()

Get iterator for the list elements.

11.140.1 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 335 of file [list](#).

11.140.2 Member Function Documentation

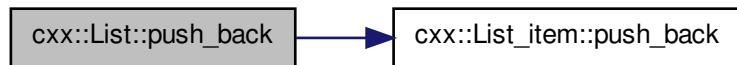
11.140.2.1 template<typename D , template< typename A > class Alloc = New_allocator> void cxx::List< D, Alloc >::push_back (D const & d) throw () [inline]

Add element at the end of the list.

Definition at line 381 of file [list](#).

References [cxx::List_item::push_back\(\)](#).

Here is the call graph for this function:



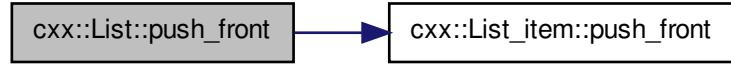
11.140.2.2 template<typename D , template< typename A > class Alloc = New_allocator> void cxx::List< D, Alloc >::push_front (D const & d) throw () [inline]

Add element at the beginning of the list.

Definition at line 390 of file [list](#).

References [cxx::List_item::push_front\(\)](#).

Here is the call graph for this function:



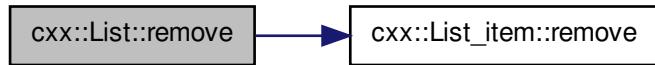
11.140.2.3 template<typename D , template< typename A > class Alloc = New_allocator> void cxx::List< D, Alloc >::remove (Iter const & i) throw () [inline]

Remove element pointed to by the iterator.

Definition at line 399 of file [list](#).

References [cxx::List_item::remove\(\)](#).

Here is the call graph for this function:



11.140.2.4 template<typename D , template< typename A > class Alloc = New_allocator> unsigned long cxx::List< D, Alloc >::size () const throw () [inline]

Get the length of the list.

Definition at line 403 of file [list](#).

11.140.2.5 template<typename D , template< typename A > class Alloc = New_allocator> D const& cxx::List< D, Alloc >::operator[] (unsigned long idx) const throw () [inline]

Random access.

Complexity is O(n).

Definition at line 406 of file [list](#).

11.140.2.6 template<typename D , template< typename A > class Alloc = New_allocator> D& cxx::List< D, Alloc >::operator[] (unsigned long *idx*) throw () [inline]

Random access.

Complexity is O(n).

Definition at line 410 of file [list](#).

11.140.2.7 template<typename D , template< typename A > class Alloc = New_allocator> Iter cxx::List< D, Alloc >::items () throw () [inline]

Get iterator for the list elements.

Definition at line 414 of file [list](#).

The documentation for this class was generated from the following file:

- l4/cxx/list

11.141 cxx::List_alloc Class Reference

Standard list-based allocator.

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 \(unsigned long size, unsigned align\)](#)
Alloc a memory block.
- [unsigned long avail \(\)](#)
Get the amount of available memory.

11.141.1 Detailed Description

Standard list-based allocator.

Definition at line 29 of file [list_alloc](#).

11.141.2 Constructor & Destructor Documentation

11.141.2.1 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 48 of file [list_alloc](#).

11.141.3 Member Function Documentation

11.141.3.1 `void cxx::List_alloc::free (void * block, unsigned long size, bool initial_free = false) [inline]`

Return a free memory block to the allocator.

Parameters

block pointer to memory block

size size of memory block

initial_free Set to true for putting fresh memory to the allocator. This will enforce alignment on that memory.

Definition at line 199 of file [list_alloc](#).

11.141.3.2 `void * cxx::List_alloc::alloc (unsigned long size, unsigned align) [inline]`

Alloc a memory block.

Parameters

size Size of the memory block

align Alignment constraint

Returns

Pointer to memory block

Definition at line 237 of file [list_alloc](#).

11.141.3.3 `unsigned long cxx::List_alloc::avail () [inline]`

Get the amount of available memory.

Returns

Available memory in bytes

Definition at line 308 of file [list_alloc](#).

The documentation for this class was generated from the following file:

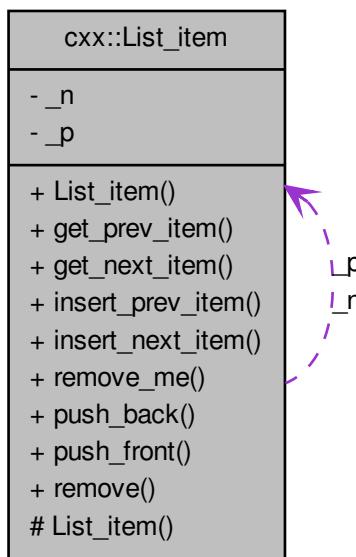
- [l4/cxx/list_alloc](#)

11.142 cxx::List_item Class Reference

Basic list item.

Inherited by cxx::List< D, Alloc >::E, and cxx::T_list_item< T >.

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 throw \(\)](#)

Get previous item.

- [List_item * get_next_item \(\) const throw \(\)](#)

Get next item.

- [void insert_prev_item \(List_item *p\) throw \(\)](#)

Insert item p before this item.

- `void insert_next_item (List_item *p) throw ()`

Insert item p after this item.

- `void remove_me () throw ()`

Remove this item from the list.

Static Public Member Functions

- `template<typename C, typename N >`
`static C * push_back (C *head, N *p) throw ()`

Append item to a list.

- `template<typename C, typename N >`
`static C * push_front (C *head, N *p) throw ()`

Prepend item to a list.

- `template<typename C, typename N >`
`static C * remove (C *head, N *p) throw ()`

Remove item from a list.

11.142.1 Detailed Description

Basic list item. Basic item that can be member of a doubly linked, cyclic list.

Definition at line 37 of file [list](#).

11.142.2 Member Function Documentation

11.142.2.1 `List_item* cxx::List_item::get_prev_item () const throw () [inline]`

Get previous item.

Definition at line 174 of file [list](#).

11.142.2.2 `List_item* cxx::List_item::get_next_item () const throw () [inline]`

Get next item.

Definition at line 177 of file [list](#).

11.142.2.3 `void cxx::List_item::insert_prev_item (List_item * p) throw () [inline]`

Insert item p before this item.

Definition at line 180 of file [list](#).

11.142.2.4 void cxx::List_item::insert_next_item (List_item * *p*) throw () [inline]

Insert item *p* after this item.

Definition at line 190 of file [list](#).

11.142.2.5 void cxx::List_item::remove_me () throw () [inline]

Remove this item from the list.

Definition at line 199 of file [list](#).

Referenced by [cxx::List_item::Iter::remove_me\(\)](#).

Here is the caller graph for this function:

**11.142.2.6 template<typename C, typename N> C * cxx::List_item::push_back (C * *head*, N * *p*) throw () [inline, static]**

Append item to a list.

Convinience function for empty-head corner case.

Parameters

h pointer to the current list head.

p pointer to new item.

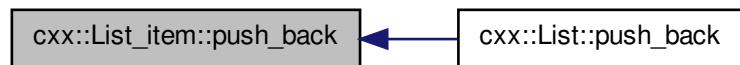
Returns

the pointer to the new head.

Definition at line 249 of file [list](#).

Referenced by [cxx::List< D, Alloc >::push_back\(\)](#).

Here is the caller graph for this function:



11.142.2.7 template<typename C, typename N> C * cxx::List_item::push_front (C * *head*, N * *p*) throw () [inline, static]

Prepend item to a list.

Convinience function for empty-head corner case.

Parameters

head pointer to the current list head.

p pointer to new item.

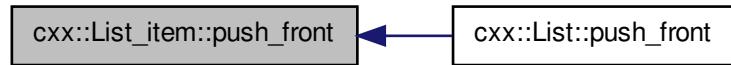
Returns

the pointer to the new head.

Definition at line 260 of file [list](#).

Referenced by [cxx::List< D, Alloc >::push_front\(\)](#).

Here is the caller graph for this function:



11.142.2.8 template<typename C, typename N> C * cxx::List_item::remove (C * *head*, N * *p*) throw () [inline, static]

Remove item from a list.

Convinience function for remove-head corner case.

Parameters

head pointer to the current list head.

p pointer to the item to remove.

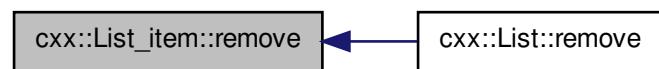
Returns

the pointer to the new head.

Definition at line 270 of file [list](#).

Referenced by [cxx::List< D, Alloc >::remove\(\)](#).

Here is the caller graph for this function:



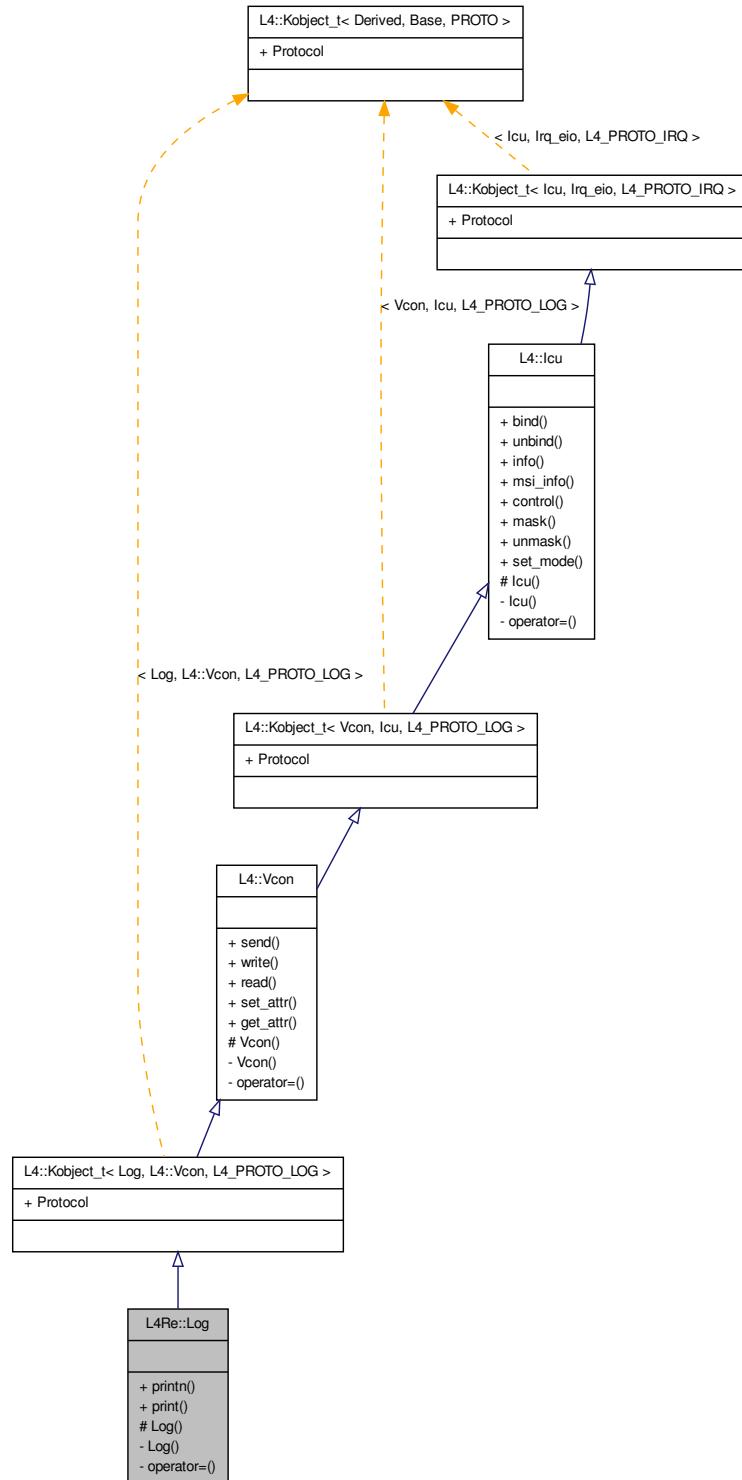
The documentation for this class was generated from the following file:

- l4/cxx/list

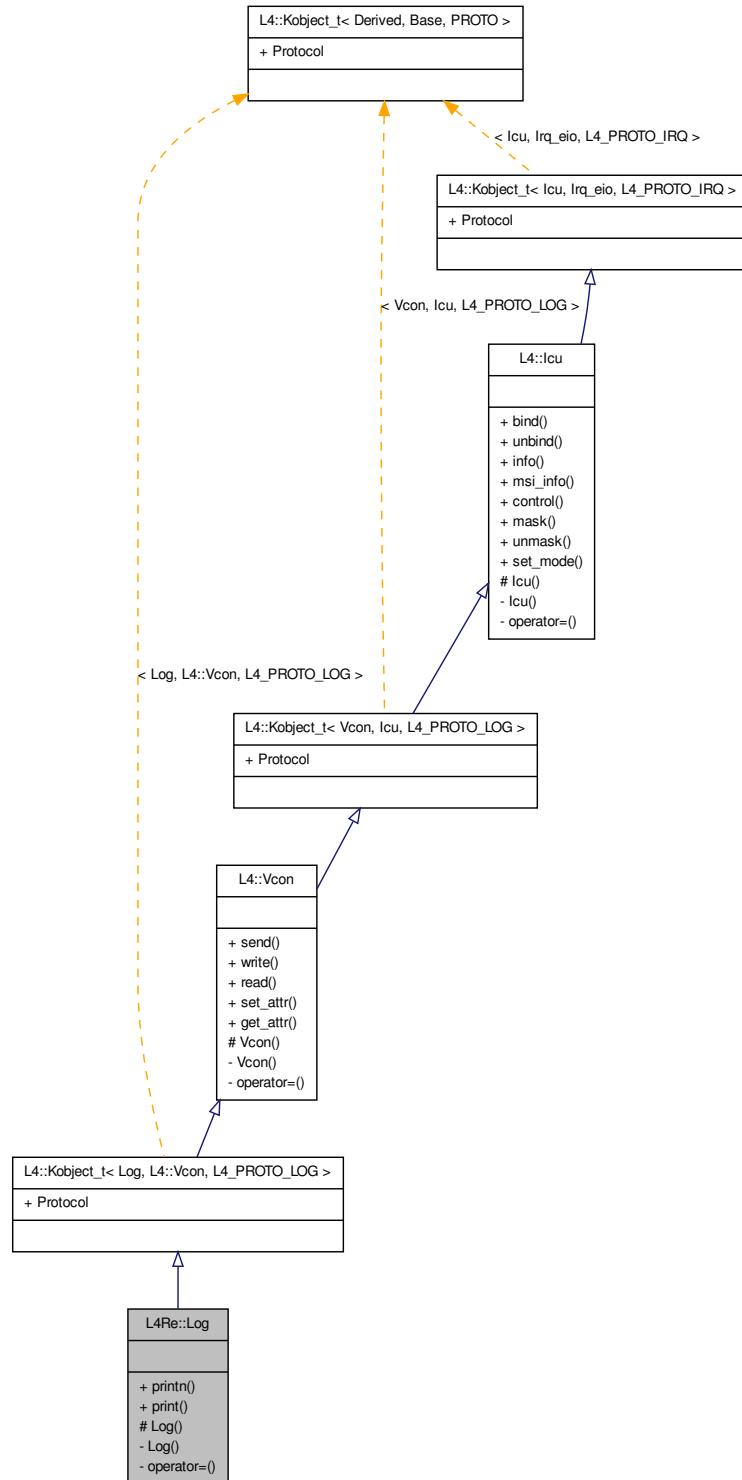
11.143 L4Re::Log Class Reference

[Log](#) interface class.

Inheritance diagram for L4Re::Log:



Collaboration diagram for L4Re::Log:



Public Member Functions

- void `printn` (char const **string*, int *len*) const throw ()
Print string with length len, NULL characters don't matter.
- void `print` (char const **string*) const throw ()
Print NULL-terminated string.

11.143.1 Detailed Description

[Log](#) interface class.

Definition at line 44 of file [log](#).

11.143.2 Member Function Documentation

11.143.2.1 void L4Re::Log::printn (char const * *string*, int *len*) const throw ()

Print string with length len, NULL characters don't matter.

Parameters

- string* string to print
len length of string

11.143.2.2 void L4Re::Log::print (char const * *string*) const throw ()

Print NULL-terminated string.

Parameters

- string* string to print

The documentation for this class was generated from the following file:

- [l4/re/log](#)

11.144 L4::Factory::Lstr Struct Reference

Special type to add a pascal string into the factory create stream.

Data Fields

- char const * `s`
The character buffer.
- int `len`
The number of characters in the buffer.

11.144.1 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 61 of file [factory](#).

The documentation for this struct was generated from the following file:

- 14/sys/factory

11.145 cxx::Lt_functor< Obj > Struct Template Reference

Generic comparator class that defaults to the less-than operator.

11.145.1 Detailed Description

```
template<typename Obj> struct cxx::Lt_functor< Obj >
```

Generic comparator class that defaults to the less-than operator.

Definition at line 29 of file [std_ops](#).

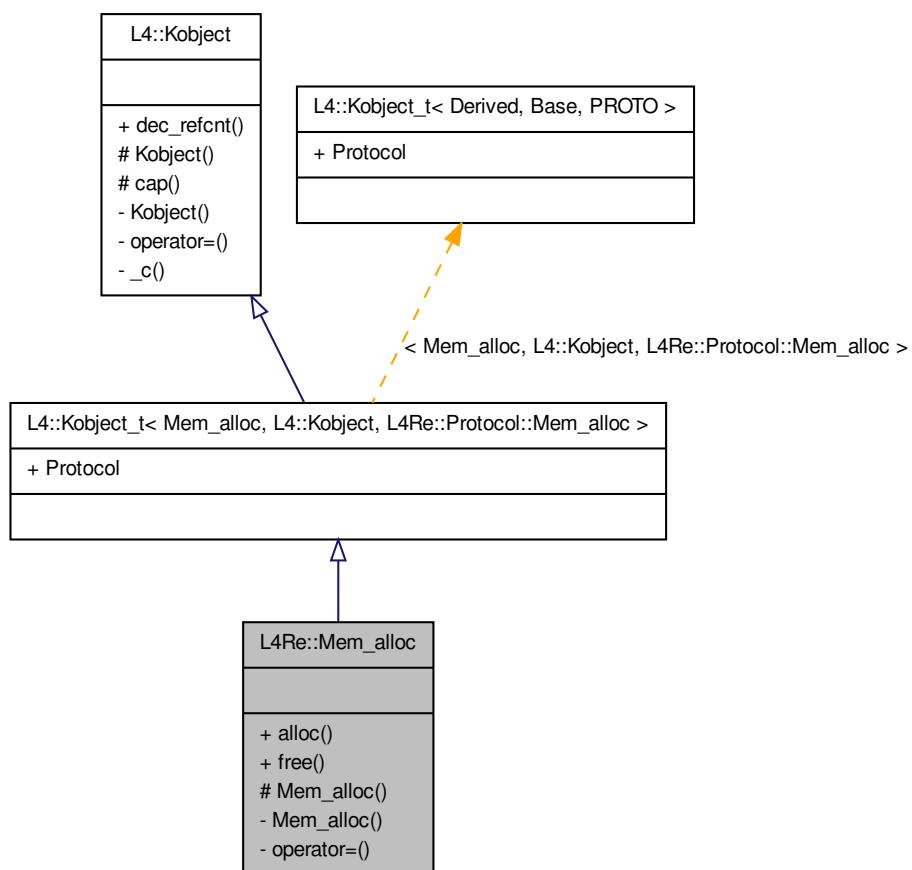
The documentation for this struct was generated from the following file:

- 14/cxx/std_ops

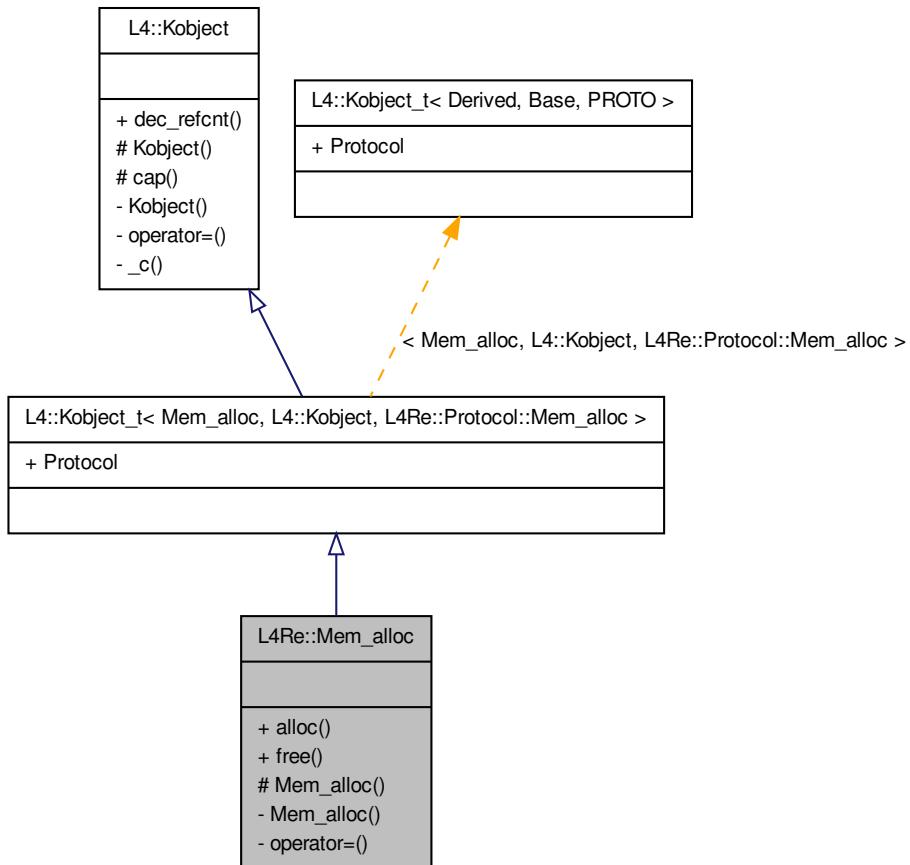
11.146 L4Re::Mem_alloc Class Reference

Memory allocator.

Inheritance diagram for L4Re::Mem_alloc:



Collaboration diagram for L4Re::Mem_alloc:



Public Types

- enum `Mem_alloc_flags` { `Continuous` = 0x01, `Pinned` = 0x02, `Super_pages` = 0x04 }
- Flags for the allocator.*

Public Member Functions

- long `alloc` (unsigned long `size`, L4::Cap< `Dataspace` > `mem`, unsigned long `flags`=0) const throw ()

Allocate anonymous memory.
- long `free` (L4::Cap< `Dataspace` > `mem`) const throw ()

Free data space.

11.146.1 Detailed Description

Memory allocator. Memory-allocator interface, for more information see [Memory-allocator API](#) .

Definition at line 69 of file [mem_alloc](#).

11.146.2 Member Enumeration Documentation

11.146.2.1 enum L4Re::Mem_alloc::Mem_alloc_flags

Flags for the allocator.

Enumerator:

Continuous Allocate physically contiguous data space, if supported by the allocator.

Pinned Allocate pinned data space, if supported by the allocator.

Super_pages Allocate super pages, if supported by the allocator.

Definition at line 78 of file [mem_alloc](#).

11.146.3 Member Function Documentation

11.146.3.1 long L4Re::Mem_alloc::alloc (*unsigned long size*, *L4::Cap< Dataspace > mem*, *unsigned long flags = 0*) const throw ()

Allocate anonymous memory.

Parameters

size Size to be requested in bytes (granularity is (super)pages and the size is rounded up to this granularity).

mem Object capability for the data space to be allocated.

flags Flags, see [Mem_alloc_flags](#), default none

Returns

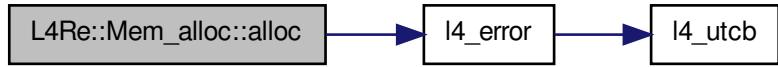
0 on success, <0 on error

- [-L4_ENOMEM](#)
- IPC errors

Definition at line 35 of file [mem_alloc_impl.h](#).

References [l4_error\(\)](#).

Here is the call graph for this function:



11.146.3.2 long L4Re::Mem_alloc::free (L4::Cap< Dataspace > mem) const throw ()

Free data space.

Parameters

mem Data space that contains the memory.

Returns

0 on success, <0 on error

- [-L4_EINVAL](#)
- IPC errors

Definition at line 45 of file [mem_alloc_impl.h](#).

The documentation for this class was generated from the following files:

- [l4/re/mem_alloc](#)
- [l4/re/impl/mem_alloc_impl.h](#)

11.147 L4::Kip::Mem_desc Class Reference

Memory descriptors stored in the kernel interface page.

Public Types

- enum [Mem_type](#)

Memory types.

Public Member Functions

- [Mem_desc](#) (unsigned long start, unsigned long end, [Mem_type](#) t, unsigned char st=0, bool virt=false)
throw ()

Initialize memory descriptor.

- `unsigned long start () const throw ()`
Return start address of memory descriptor.
- `unsigned long end () const throw ()`
Return end address of memory descriptor.
- `unsigned long size () const throw ()`
Return size of region described by the memory descriptor.
- `Mem_type type () const throw ()`
Return type of the memory descriptor.
- `unsigned char sub_type () const throw ()`
Return sub-type of the memory descriptor.
- `unsigned is_virtual () const throw ()`
Return whether the memory descriptor describes a virtual or physical region.
- `void set (unsigned long start, unsigned long end, Mem_type t, unsigned char st=0, bool virt=false) throw ()`
Set values of a memory descriptor.

Static Public Member Functions

- `static Mem_desc * first (void *kip) throw ()`
Get first memory descriptor.
- `static unsigned long count (void const *kip) throw ()`
Return number of memory descriptors stored in the kernel info page.
- `static void count (void *kip, unsigned count) throw ()`
Set number of memory descriptors.

11.147.1 Detailed Description

Memory descriptors stored in the kernel interface page. `#include <l4/sys/kip>`
 Definition at line [51](#) of file `kip`.

11.147.2 Constructor & Destructor Documentation

11.147.2.1 L4::Kip::Mem_desc::Mem_desc (`unsigned long start, unsigned long end, Mem_type t, unsigned char st = 0, bool virt = false`) `throw () [inline]`

Initialize memory descriptor.

Parameters

start Start address
end End address
t Memory type
st Memory subtype, defaults to 0
virt True for virtual memory, false for physical memory, defaults to physical

Definition at line 125 of file [kip](#).

11.147.3 Member Function Documentation

11.147.3.1 static Mem_desc* L4::Kip::Mem_desc::first (void * *kip*) throw () [inline, static]

Get first memory descriptor.

Parameters

kip Pointer to the kernel info page

Returns

First memory descriptor stored in the kernel info page

Definition at line 83 of file [kip](#).

11.147.3.2 static unsigned long L4::Kip::Mem_desc::count (void const * *kip*) throw () [inline, static]

Return number of memory descriptors stored in the kernel info page.

Parameters

kip Pointer to the kernel info page

Returns

Number of memory descriptors in the kernel info page.

Definition at line 100 of file [kip](#).

11.147.3.3 static void L4::Kip::Mem_desc::count (void * *kip*, unsigned *count*) throw () [inline, static]

Set number of memory descriptors.

Parameters

kip Pointer to the kernel info page
count Number of memory descriptors

Definition at line 110 of file [kip](#).

11.147.3.4 unsigned long L4::Kip::Mem_desc::start () const throw () [inline]

Return start address of memory descriptor.

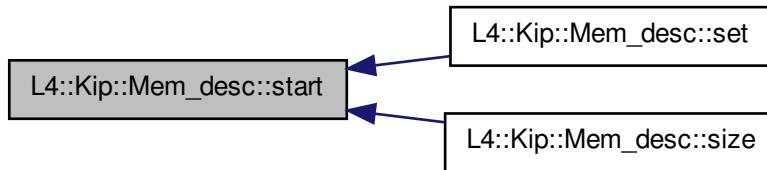
Returns

Start address of memory descriptor

Definition at line 135 of file [kip](#).

Referenced by [set\(\)](#), and [size\(\)](#).

Here is the caller graph for this function:

**11.147.3.5 unsigned long L4::Kip::Mem_desc::end () const throw () [inline]**

Return end address of memory descriptor.

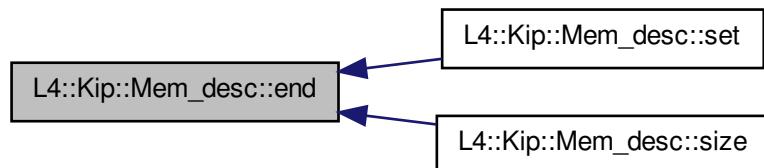
Returns

End address of memory descriptor

Definition at line 141 of file [kip](#).

Referenced by [set\(\)](#), and [size\(\)](#).

Here is the caller graph for this function:



11.147.3.6 unsigned long L4::Kip::Mem_desc::size () const throw () [inline]

Return size of region described by the memory descriptor.

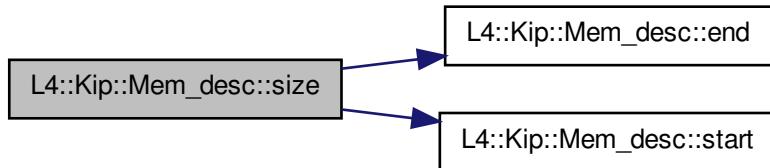
Returns

Size of the region described by the memory descriptor

Definition at line 147 of file [kip](#).

References [end\(\)](#), and [start\(\)](#).

Here is the call graph for this function:

**11.147.3.7 Mem_type L4::Kip::Mem_desc::type () const throw () [inline]**

Return type of the memory descriptor.

Returns

Type of the memory descriptor

Definition at line 153 of file [kip](#).

11.147.3.8 unsigned char L4::Kip::Mem_desc::sub_type () const throw () [inline]

Return sub-type of the memory descriptor.

Returns

Sub-type of the memory descriptor

Definition at line 159 of file [kip](#).

11.147.3.9 unsigned L4::Kip::Mem_desc::is_virtual () const throw () [inline]

Return whether the memory descriptor describes a virtual or physical region.

Returns

True for virtual region, false for physical region.

Definition at line 166 of file [kip](#).

11.147.3.10 `void L4::Kip::Mem_desc::set (unsigned long start, unsigned long end, Mem_type t, unsigned char st = 0, bool virt = false) throw () [inline]`

Set values of a memory descriptor.

Parameters

start Start address

end End address

t Memory type

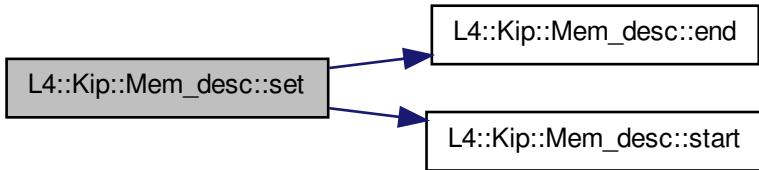
st Sub-type, defaults to 0

virt Virtual or physical memory region, defaults to physical

Definition at line 176 of file [kip](#).

References [end\(\)](#), and [start\(\)](#).

Here is the call graph for this function:



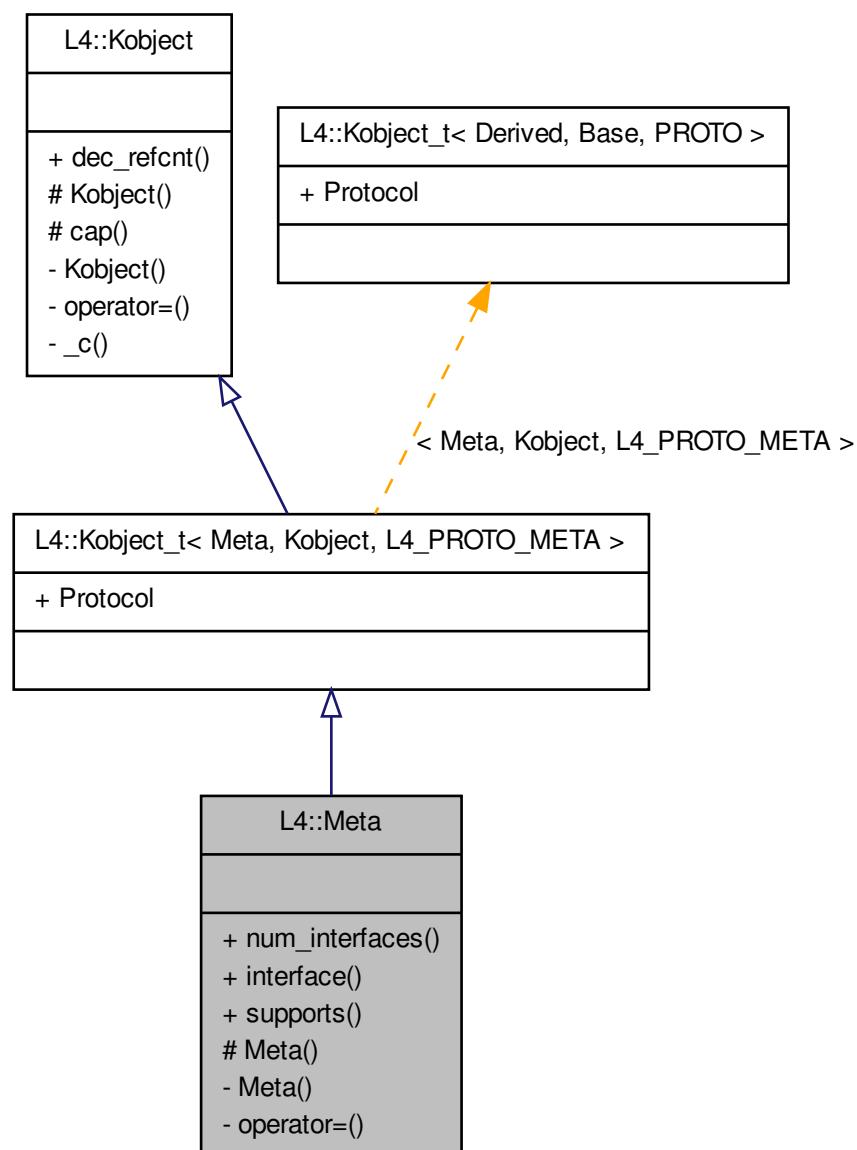
The documentation for this class was generated from the following file:

- [14/sys/kip](#)

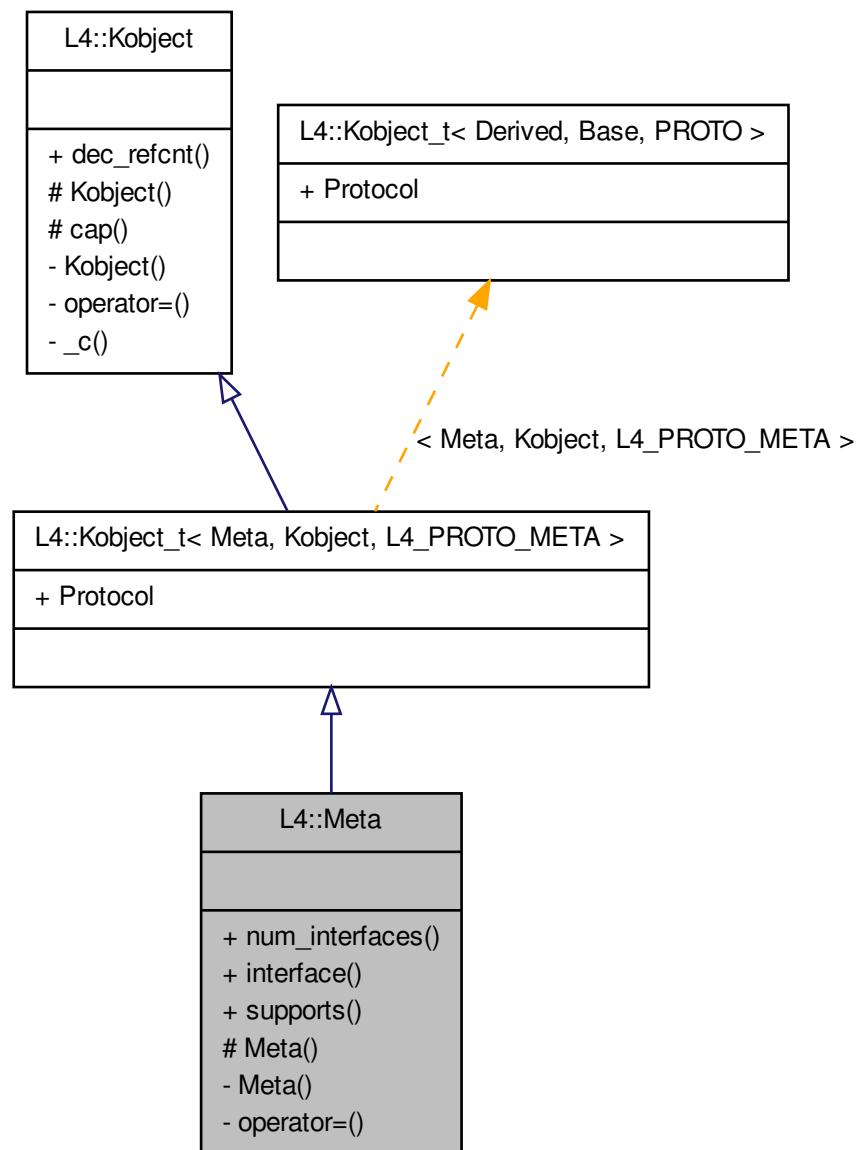
11.148 L4::Meta Class Reference

[Meta](#) interface that shall be implemented by each [L4Re](#) object and gives access to the dynamic type information for [L4Re](#) objects.

Inheritance diagram for L4::Meta:



Collaboration diagram for L4::Meta:



Public Member Functions

- `l4_msntag_t num_interfaces (l4_utcb_t *utcb=l4_utcb()) throw ()`
Get the number of interfaces implemented by this object.
- `l4_msntag_t interface (int idx, l4_utcb_t *u=l4_utcb()) throw ()`

Get the protocol number that must be used for the interface with the index *idx*.

- **`l4_mshtag_t` supports (long protocol, `l4_utcb_t` **u*=`l4_utcb()`) throw ()**

Figure out if the object supports the given protocol (number).

11.148.1 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 41 of file `meta`.

11.148.2 Member Function Documentation

11.148.2.1 `l4_mshtag_t L4::Meta::num_interfaces (l4_utcb_t * utcb = l4_utcb()) throw () [inline]`

Get the number of interfaces implemented by this object.

Parameters

`utcb` is the utcb to use for sending the message.

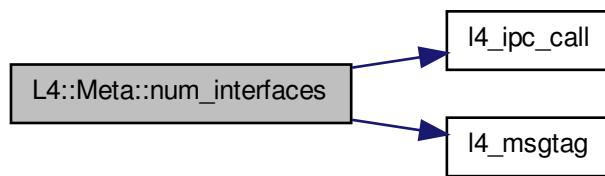
Returns

The message tag for the operation, the label (`l4_mshtag_t::label()`) is set to the number of interfaces if successful, or to -error when an error occurred.

Definition at line 89 of file `meta`.

References `l4_ipc_call()`, `L4_IPC_NEVER`, `l4_mshtag()`, and `l4_msg_regs_t::mr`.

Here is the call graph for this function:



11.148.2.2 `l4_mshtag_t L4::Meta::interface (int idx, l4_utcb_t * u = l4_utcb()) throw () [inline]`

Get the protocol number that must be used for the interface with the index *idx*.

Parameters

idx is the index of the interface to get the protocol number for. *idx* must be ≥ 0 and $<$ the return value of [l4_num_interfaces\(\)](#).

utcb is the utcb to use for sending the message.

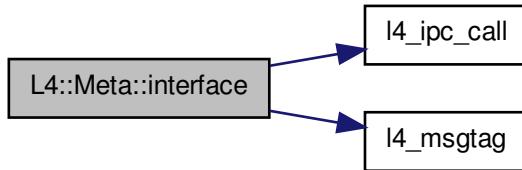
Returns

The message tag for the operation, the label ([l4_mshtag_t::label\(\)](#)) is set to the protocol number of interface *idx*.

Definition at line 98 of file [meta](#).

References [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), [l4_mshtag\(\)](#), and [l4_msg_regs_t::mr](#).

Here is the call graph for this function:



11.148.2.3 l4_mshtag_t L4::Meta::supports (long protocol, l4_utcb_t * u = l4_utcb()) throw () [inline]

Figure out if the object supports the given *protocol* (number).

Parameters

protocol is the protocol number to check for.

utcb is the utcb to use for sending the message.

Returns

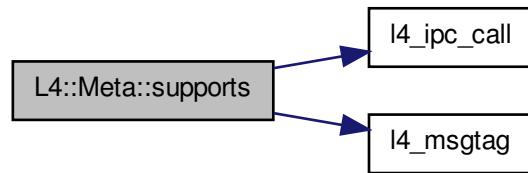
The message tag for the operation, the label ([l4_mshtag_t::label\(\)](#)) is set to 1 if *protocol* is supported to 0 if not.

This method is intended to be used for statically assigned protocol numbers.

Definition at line 108 of file [meta](#).

References [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), [l4_mshtag\(\)](#), and [l4_msg_regs_t::mr](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

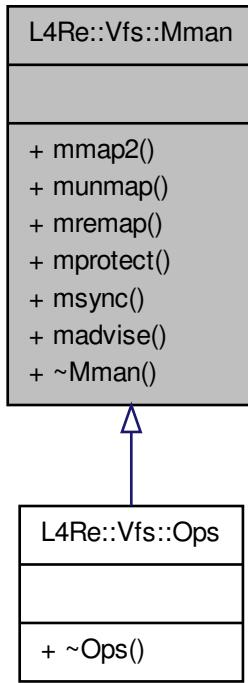
- l4/sys/meta

11.149 L4Re::Vfs::Mman Class Reference

Interface for the POSIX memory management.

```
#include <vfs.h>
```

Inheritance 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)=0 throw ()

Backend for the mmap2 system call.
- virtual int **munmap** (void *start, size_t len)=0 throw ()

Backend for the munmap system call.
- virtual int **mremap** (void *old, size_t old_sz, size_t new_sz, int flags, void **new_adr)=0 throw ()

Backend for the mremap system call.
- virtual int **mprotect** (const void *a, size_t sz, int prot)=0 throw ()

Backend for the mprotect system call.
- virtual int **msync** (void *addr, size_t len, int flags)=0 throw ()

Backend for the msync system call.
- virtual int **madvise** (void *addr, size_t len, int advice)=0 throw ()

Backend for the madvice system call.

11.149.1 Detailed Description

Interface for the POSIX memory management.

Note

This interface exists usually as a singleton as superclass of [L4Re::Vfs::Ops](#).

An implementation for this interface is in [14/l4re_vfs/impl/vfs_impl.h](#) and used by the l4re_vfs library or by the VFS implementation in lndo.

Definition at line [785](#) of file [vfs.h](#).

The documentation for this class was generated from the following file:

- [14/l4re_vfs/vfs.h](#)

11.150 L4::Thread::Modify_senders Class Reference

Wrapper class for modifying 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) throw ()`
Add a rule.

11.150.1 Detailed Description

Wrapper class for modifying senders. Use the `add()` function to add modification rules, and use `modify_senders()` to commit. Do not use the UTCB inbetween as it is used by `add()` and `modify_senders()`.

Definition at line [220](#) of file [thread](#).

11.150.2 Member Function Documentation

11.150.2.1 `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) throw () [inline]`

Add a rule.

Parameters

`match_mask` Bitmask of bits to match the label.

`match` Bitmask that must be equal to the label after applying `match_mask`.

`del_bits` Bits to be deleted from the label.

`add_bits` Bits to be added to the label.

Returns

0 on sucess, <0 on error

Only the first match is applied.

See also

[l4_thread_modify_sender_add\(\)](#)

Definition at line [249](#) of file [thread](#).

References [L4_UTCB_GENERIC_DATA_SIZE](#), and [l4_msg_regs_t::mr](#).

The documentation for this class was generated from the following file:

- [l4/sys/thread](#)

11.151 L4::Ipc::Msg_ptr< T > Class Template Reference

Pointer to an element of type T in an [Ipc::Istream](#).

Public Member Functions

- [Msg_ptr \(T *&p\)](#)

Create a [Msg_ptr](#) object that set pointer p to point into the message buffer.

11.151.1 Detailed Description

template<typename T> class L4::Ipc::Msg_ptr< T >

Pointer to an element of type T in an [Ipc::Istream](#). This wrapper can be used to extract an element of type T from an [Ipc::Istream](#), whereas the data is not copied out, but a pointer into the message buffer itself is returned. With this 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 [169](#) of file [ipc_stream](#).

11.151.2 Constructor & Destructor Documentation

11.151.2.1 template<typename T> L4::Ipc::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 179 of file [ipc_stream](#).

The documentation for this class was generated from the following file:

- l4/cxx/ipc_stream

11.152 L4Re::Util::Names::Name Class Reference

[Name](#) class.

Inherits [cxx::String](#).

11.152.1 Detailed Description

[Name](#) class.

Definition at line 36 of file [name_space_svr](#).

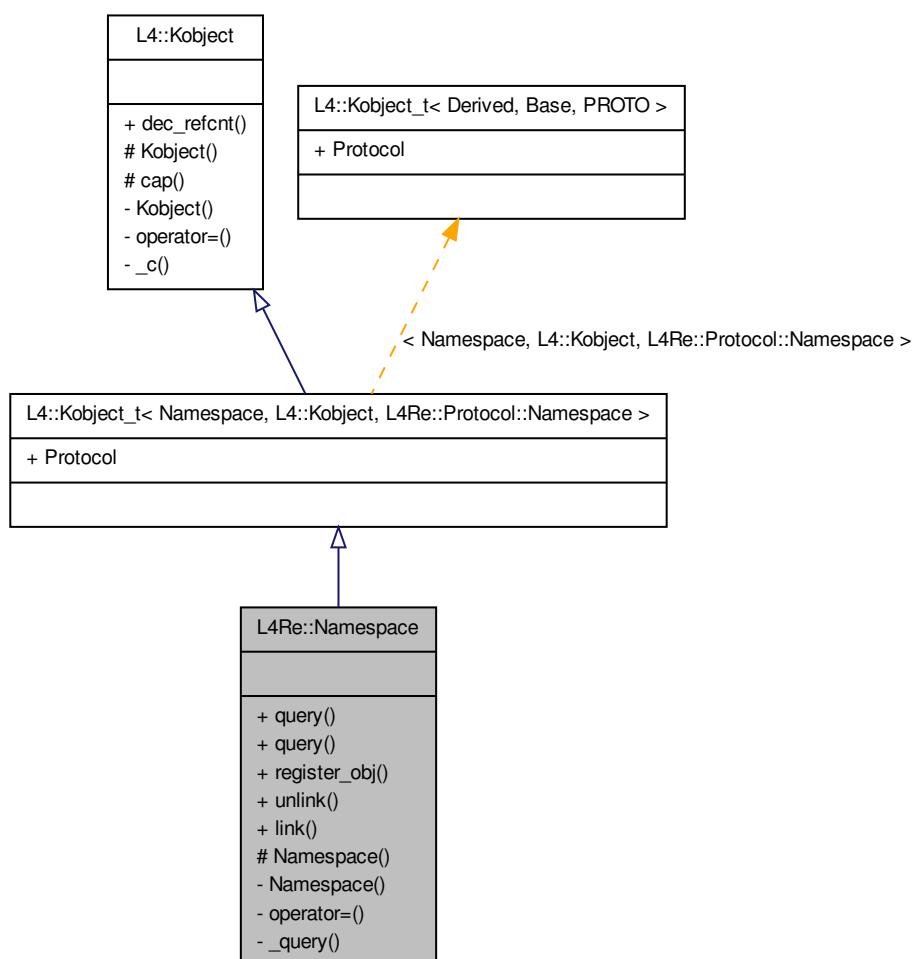
The documentation for this class was generated from the following file:

- l4/re/util/name_space_svr

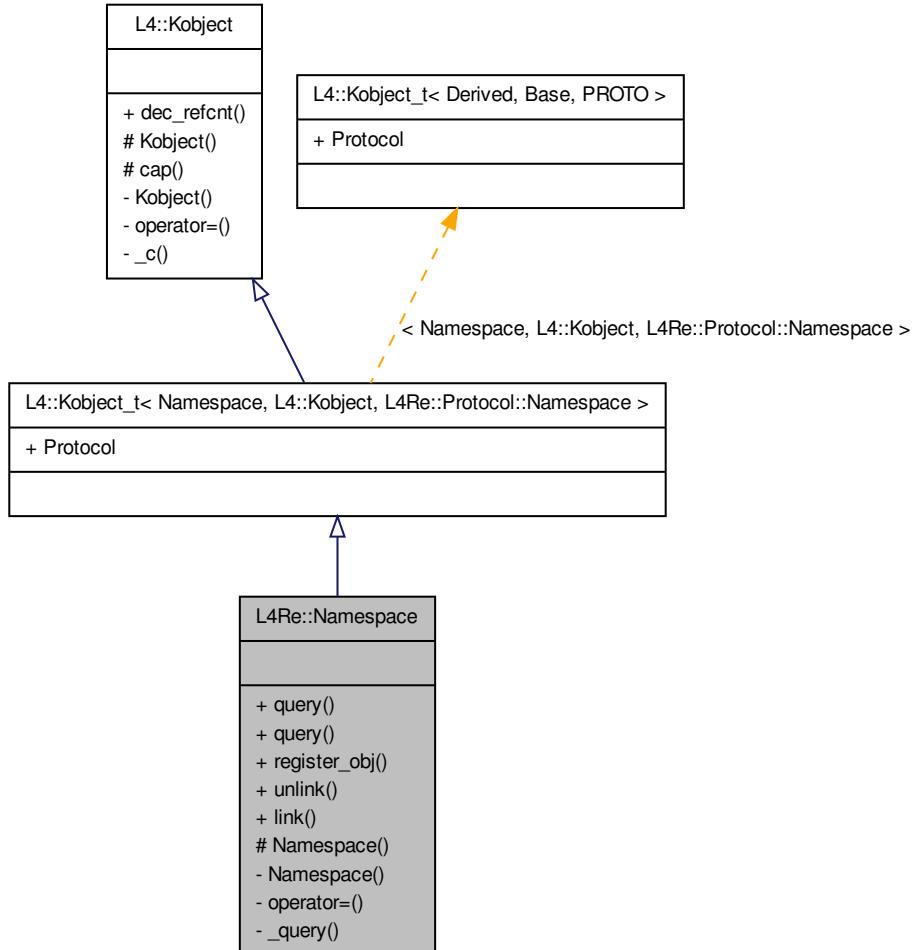
11.153 L4Re::Namespace Class Reference

Name-space interface.

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 , `Strong` = L4_CAP_FPAGE_S }
- Flags for registering name spaces.*

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 throw ()
Query a name.

- 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 throw ()

Query a name.

- long `register_obj` (char const *name, L4::Cap< void > const &obj, unsigned flags=Rw) const throw ()

Register an object with a name.

11.153.1 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 57 of file [namespace](#).

11.153.2 Member Enumeration Documentation

11.153.2.1 enum L4Re::Namespace::Register_flags

Flags for registering name spaces.

Enumerator:

Ro Read-only.

Rw Read-write.

Strong Strong.

Definition at line 66 of file [namespace](#).

11.153.3 Member Function Documentation

11.153.3.1 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 throw ()

Query a name.

Parameters

name String to query

cap Capability slot to put object into.

timeout Timeout of query in milliseconds.

Return values

local_id Local id.

Returns

- <0 on failure, see [l4_error_code_t](#).
- [-L4_ENOENT](#)
 - IPC errors == 0 if name could be fully resolved > 0 if name could not be fully resolved

Definition at line 109 of file [namespace_impl.h](#).

11.153.3.2 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 throw ()

Query a name.

Parameters

- name* String to query
len Length of the string to query.
cap Capability slot to put object into.
timeout Timeout of query in milliseconds.

Return values

local_id Local id.

Returns

- <0 on failure, see [l4_error_code_t](#).
- [-L4_ENOENT](#)
 - IPC errors == 0 if name could be fully resolved > 0 if name could not be fully resolved

Definition at line 75 of file [namespace_impl.h](#).

11.153.3.3 long L4Re::Namespace::register_obj (char const * *name*, L4::Cap< void > const & *obj*, unsigned *flags* = [Rw](#)) const throw ()

Register an object with a name.

Parameters

- name* String to register.
obj Object to register.
flags Flags to use, see [Register_flags](#), default is rw.

Returns

0 on success, <0 on failure, see [l4_error_code_t](#).

- [-L4_EEXIST](#)
- [-L4_EPERM](#)
- [-L4_ENOMEM](#)

- [-L4_EINVAL](#)
- IPC errors

Definition at line 115 of file [namespace_impl.h](#).

The documentation for this class was generated from the following files:

- [l4/re/namespace](#)
- [l4/re/impl/namespace_impl.h](#)

11.154 `cxx::New_allocator< _Type >` Class Template Reference

Standard allocator based on `operator new ()`.

11.154.1 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 60 of file [std_alloc](#).

The documentation for this class was generated from the following file:

- [l4/cxx/std_alloc](#)

11.155 `L4::Factory::Nil` Struct Reference

Special type to add a void argument into the factory create stream.

11.155.1 Detailed Description

Special type to add a void argument into the factory create stream.

Definition at line 53 of file [factory](#).

The documentation for this struct was generated from the following file:

- [l4/sys/factory](#)

11.156 `cxx::Avl_set< Item, Compare, Alloc >::Node` Class Reference

A smart pointer to a tree item.

Public Member Functions

- `Node ()`
Default construction for NIL pointer.
- `Node & operator= (Node const & o)`
Default assignment.
- `Item const & operator* ()`
Dereference the pointer.
- `Item const * operator-> ()`
Dereferenced member access.
- `bool valid () const`
Validity check.
- `operator Item const * ()`
Cast to a real item pointer.

11.156.1 Detailed Description

```
template<typename Item, class Compare = Lt_functor<Item>, template< typename A > class Alloc = New_allocator> class cxx::Avl_set< Item, Compare, Alloc >::Node
```

A smart pointer to a tree item.

Definition at line 148 of file [avl_set](#).

11.156.2 Member Function Documentation

11.156.2.1 template<typename Item, class Compare = Lt_functor<Item>, template< typename A > class Alloc = New_allocator> bool cxx::Avl_set< Item, Compare, Alloc >::Node::valid () const [inline]

Validity check.

Returns

false if the pointer is NIL, true if valid.

Definition at line 172 of file [avl_set](#).

The documentation for this class was generated from the following file:

- [l4/cxx/avl_set](#)

11.157 cxx::Nothrow Class Reference

Helper type to distinguish the oeprator new version that does not throw exceptions.

11.157.1 Detailed Description

Helper type to distinguish the oeprator `new` version that does not throw exceptions.

Definition at line 30 of file [std_alloc](#).

The documentation for this class was generated from the following file:

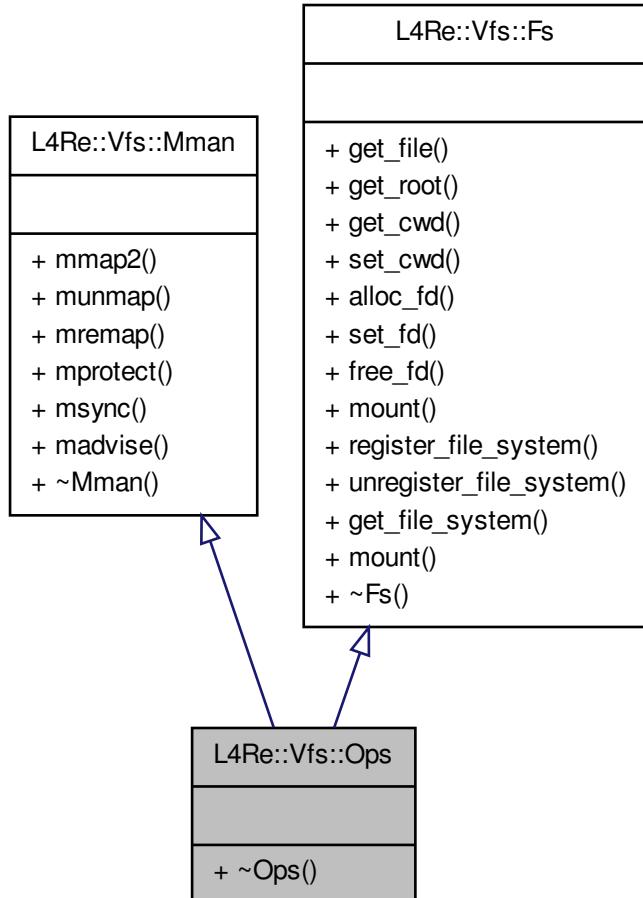
- [l4/cxx/std_alloc](#)

11.158 L4Re::Vfs::Ops Class Reference

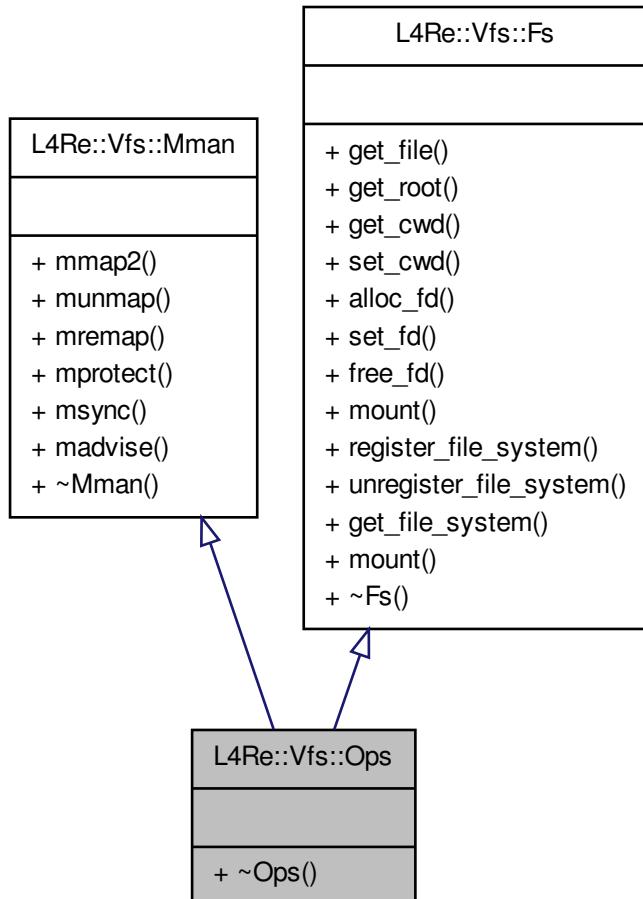
Interface for the POSIX backends for an application.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::Ops:



Collaboration diagram for L4Re::Vfs::Ops:



11.158.1 Detailed Description

Interface for the POSIX backends for 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 1016 of file [vfs.h](#).

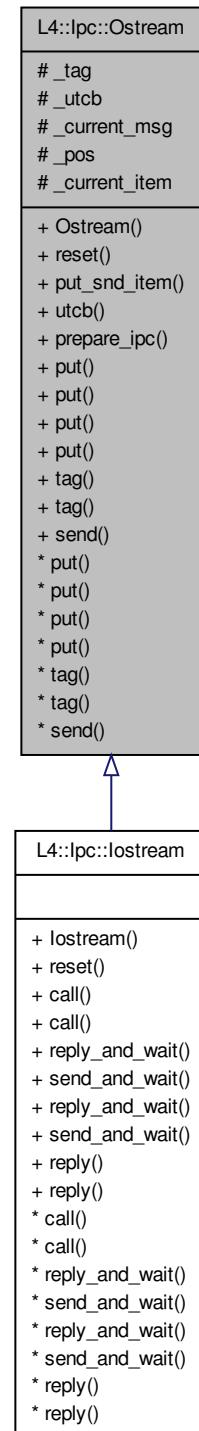
The documentation for this class was generated from the following file:

- [l4re_vfs/vfs.h](#)

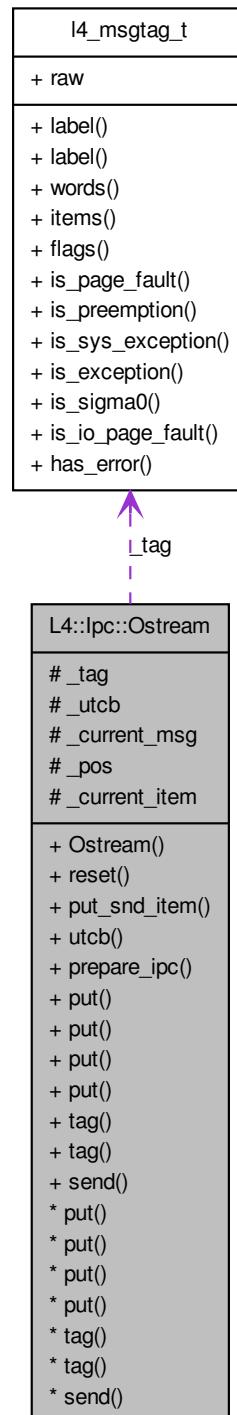
11.159 L4::Ipc::Ostream Class Reference

Output stream for IPC marshalling.

Inheritance diagram for L4::Ipc::Ostream:



Collaboration diagram for L4::Ipc::Ostream:



Public Member Functions

- **Ostream (l4_utcb_t *utcb)**
Create an IPC output stream using the given message buffer msg.
- **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.

See [IPC stream operators](#) .

- template<typename T >
void 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.
- **int put (Varg const &va)**
- template<typename T >
int put (Varg_t< T > const &va)
- **l4_mshtag_t tag () const**
Extract the [L4](#) message tag from the stream.
- **l4_mshtag_t & tag ()**
Extract a reference to the [L4](#) message tag from the stream.

IPC operations.

- **l4_mshtag_t send (l4_cap_idx_t dst, long proto=0, unsigned flags=0)**
Send the message via IPC to the given receiver.

11.159.1 Detailed Description

Output stream for IPC marshalling. [Ipc::Ostream](#) is part of the dynamic IPC marshalling infrastructure, as well as [Ipc::Istream](#) and [Ipc::Iostream](#).

[Ipc::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 [Ipc::Buf_cp_out](#)) and indirect strings (see [Msg_out_buffer](#) and [Msg_io_buffer](#)).

Definition at line 842 of file [ipc_stream](#).

11.159.2 Member Function Documentation

11.159.2.1 `template<typename T > void L4::Ipc::Ostream::put (T * buf, unsigned long size) [inline]`

Put an array with *size* elements of type *T* into the stream.

Parameters

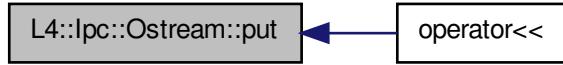
buf A pointer to the array to insert into the buffer.

size The number of elements in the array.

Definition at line 880 of file [ipc_stream](#).

Referenced by [operator<<\(\)](#).

Here is the caller graph for this function:



11.159.2.2 `template<typename T > bool L4::Ipc::Ostream::put (T const & v) [inline]`

Insert an element of type *T* into the stream.

Parameters

v The element to insert.

Definition at line 896 of file [ipc_stream](#).

11.159.2.3 `l4_mshtag_t L4::Ipc::Ostream::tag () const [inline]`

Extract the [L4](#) message tag from the stream.

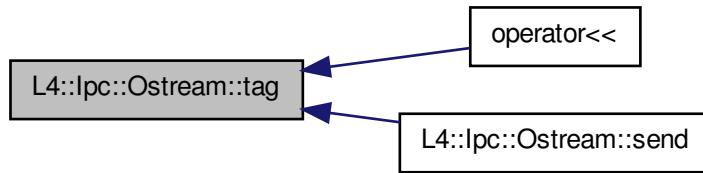
Returns

the extracted [L4](#) message tag.

Definition at line 923 of file [ipc_stream](#).

Referenced by [operator<<\(\)](#), and [send\(\)](#).

Here is the caller graph for this function:



11.159.2.4 l4_mshtag_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 929 of file [ipc_stream](#).

11.159.2.5 l4_mshtag_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.

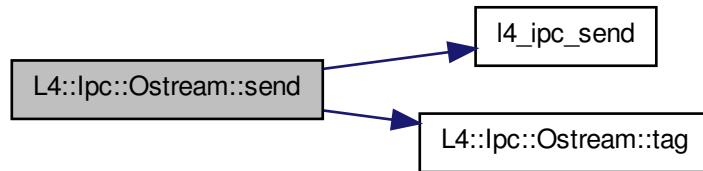
Parameters

dst The destination for the message.

Definition at line 1149 of file [ipc_stream](#).

References [L4_IPC_NEVER](#), [l4_ipc_send\(\)](#), [L4_MSGTAG_FLAGS](#), and [tag\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

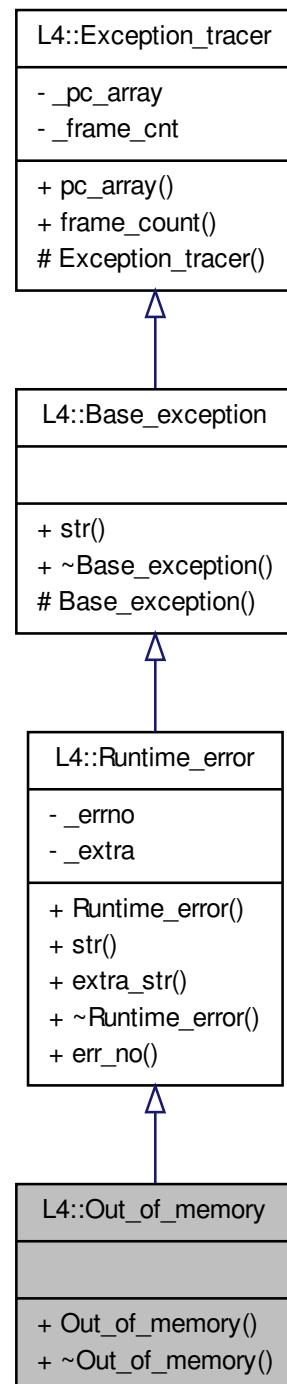
- l4/cxx/ ipc_stream

11.160 L4::Out_of_memory Class Reference

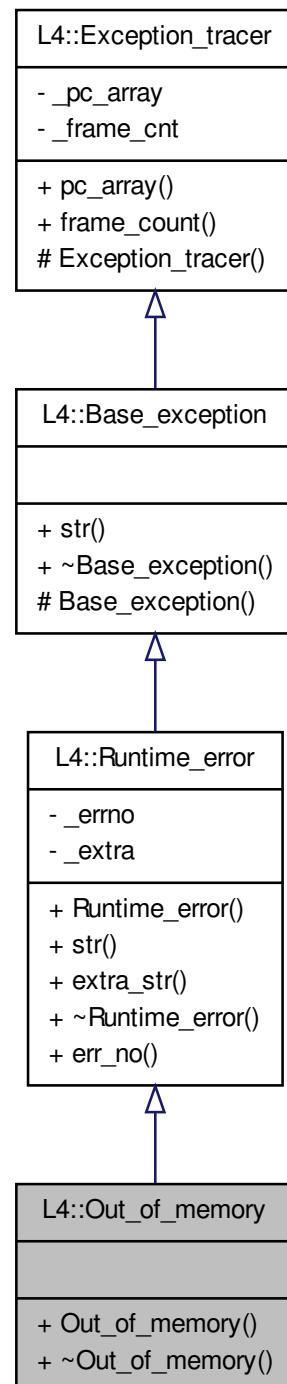
Exception signalling insufficient memory.

```
#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="") throw ()
Create an out-of-memory exception.
- `~Out_of_memory` () throw ()
Destruction.

11.160.1 Detailed Description

Exception signalling insufficient memory.

Definition at line 167 of file `exceptions`.

The documentation for this class was generated from the following file:

- 14/cxx/exceptions

11.161 `cxx::Pair<First, Second>` Struct Template Reference

`Pair` of two values.

Public Types

- `typedef First First_type`
Type of first value.
- `typedef Second Second_type`
Type of second value.

Public Member Functions

- `Pair` (First const &`first`, Second const &`second`)
Create a pair from the two values.
- `Pair` ()
Default construction.

Data Fields

- First `first`
First value.
- Second `second`
Second value.

11.161.1 Detailed Description

template<typename First, typename Second> struct cxx::Pair< First, Second >

Pair of two values. Standard container for a pair of values.

Parameters

First Type of the first value.

Second Type of the second value.

Definition at line 36 of file [pair](#).

11.161.2 Constructor & Destructor Documentation

11.161.2.1 template<typename First, typename Second> cxx::Pair< First, Second >::Pair (First const & *first*, Second const & *second*) [inline]

Create a pair from the two values.

Parameters

first The first value.

second The second value.

Definition at line 53 of file [pair](#).

The documentation for this struct was generated from the following file:

- [14/cxx/pair](#)

11.162 cxx::Pair_first_compare< Cmp, Typ > Class Template Reference

Comparison functor for [Pair](#).

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.

11.162.1 Detailed Description

template<typename Cmp, typename Typ> class cxx::Pair_first_compare< Cmp, Typ >

Comparison functor for [Pair](#).

Parameters

Cmp Comparison functor for the first value of the pair.

Typ 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](#).

11.162.2 Constructor & Destructor Documentation

11.162.2.1 `template<typename Cmp , typename Typ > cxx::Pair_first_compare< Cmp, Typ >::Pair_first_compare (Cmp const & cmp = Cmp ()) [inline]`

Construction.

Parameters

cmp The comparison functor used for the first value.

Definition at line [84](#) of file [pair](#).

11.162.3 Member Function Documentation

11.162.3.1 `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

l The lefthand value.

r The righthand value.

Definition at line [91](#) of file [pair](#).

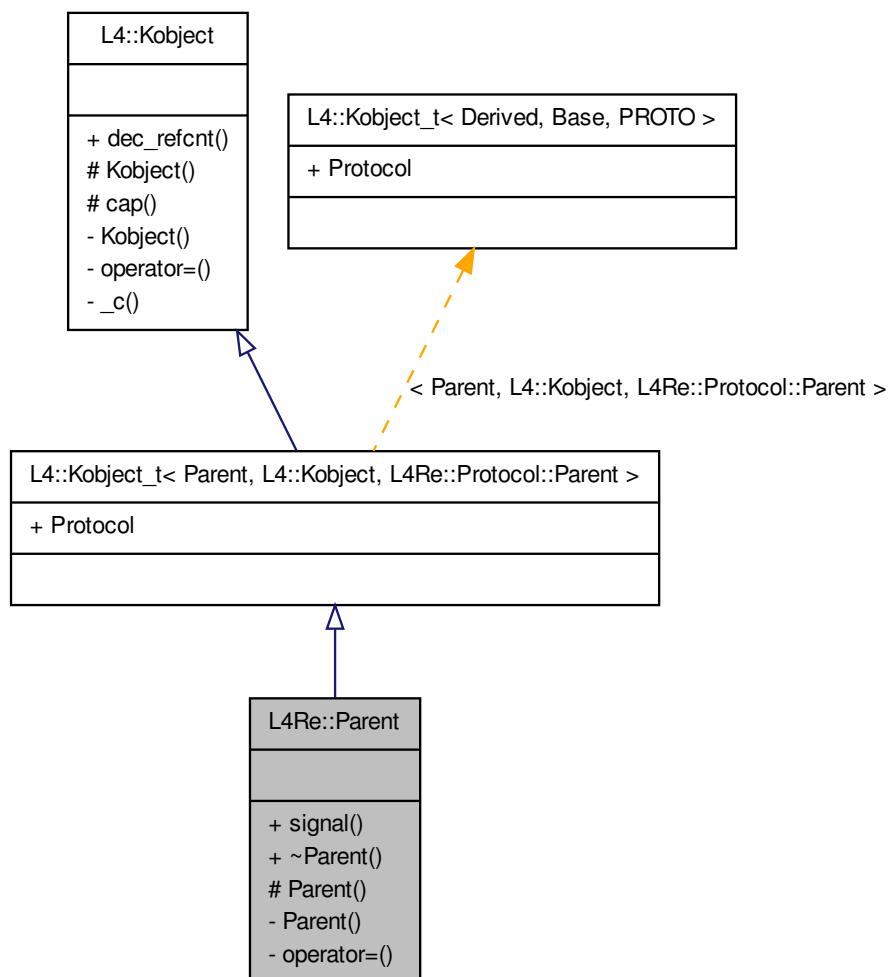
The documentation for this class was generated from the following file:

- [14/cxx/pair](#)

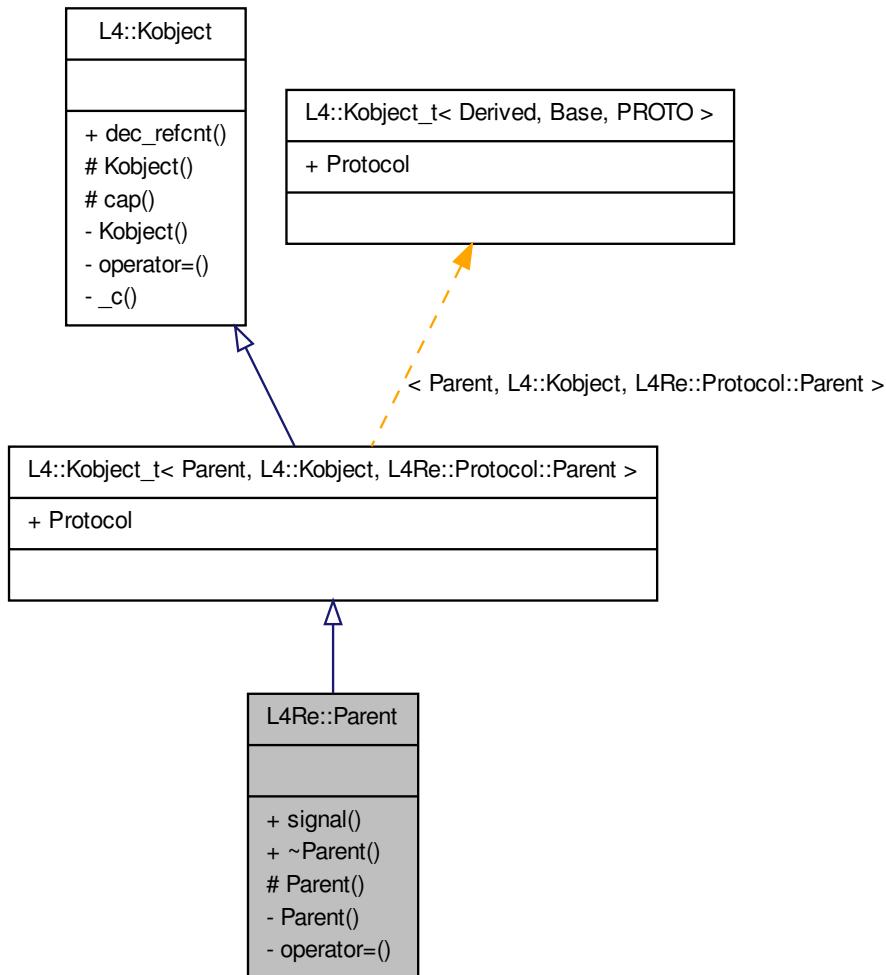
11.163 L4Re::Parent Class Reference

[Parent](#) interface.

Inheritance diagram for L4Re::Parent:



Collaboration diagram for L4Re::Parent:



Public Member Functions

- long [signal](#) (unsigned long sig, unsigned long val) const throw ()
Send a signal to the parent.

11.163.1 Detailed Description

[Parent](#) interface.

See also

[Parent API](#) for more details about the purpose.

Definition at line 50 of file [parent](#).

11.163.2 Member Function Documentation

11.163.2.1 long L4Re::Parent::signal (`unsigned long sig, unsigned long val`) const throw ()

Send a signal to the parent.

Parameters

sig Signal to send

val Value of the signal

Returns

0 on success, <0 on error

- [-L4_ENOREPLY](#)
- IPC errors

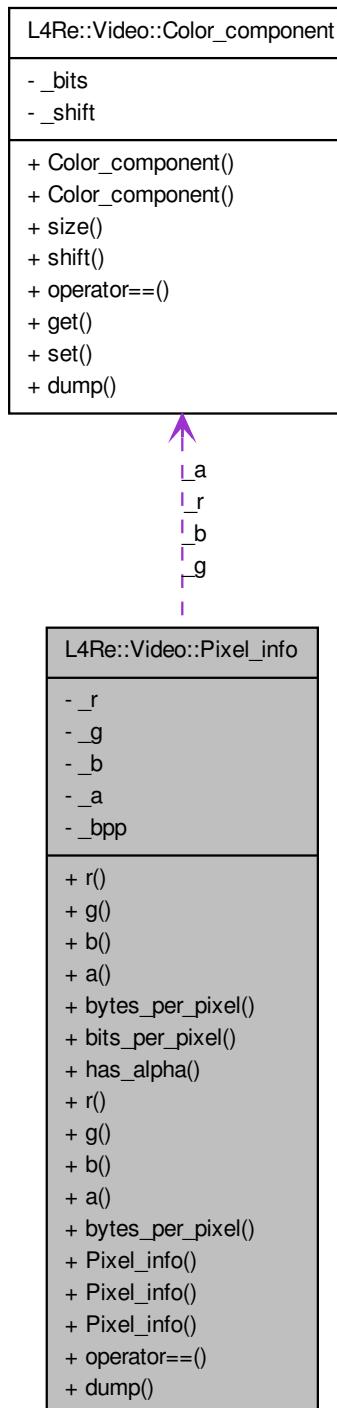
The documentation for this class was generated from the following file:

- [l4/re/parent](#)

11.164 L4Re::Video::Pixel_info Class Reference

Pixel information.

Collaboration diagram for L4Re::Video::Pixel_info:



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.
- `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 equazality of the color sapce.

- template<typename STREAM >
STREAM & **dump** (STREAM &s) const

Dump information on the pixel to a stream.

11.164.1 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 108 of file [colors](#).

11.164.2 Constructor & Destructor Documentation

11.164.2.1 L4Re::Video::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

- bpp* Size of pixel in bytes.
- r* Red component size.
- rs* Red component shift.
- g* Green component size.
- gs* Green component shift.
- b* Blue component size.
- bs* Blue component shift.
- a* Alpha component size, defaults to 0.
- as* Alpha component shift, defaults to 0.

Definition at line 205 of file [colors](#).

11.164.2.2 template<typename VBI > L4Re::Video::Pixel_info::Pixel_info (VBI const * *vbi*) [inline, explicit]

Convenience constructor.

Parameters

- vbi* Suitable information structure. Convenience constructor to create the pixel info from a VESA Framebuffer Info.

Definition at line 217 of file [colors](#).

11.164.3 Member Function Documentation

11.164.3.1 **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 119 of file [colors](#).

11.164.3.2 **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 125 of file [colors](#).

11.164.3.3 **Color_component const& L4Re::Video::Pixel_info::b () const [inline]**

Return the blue color component of the pixel.

Returns

Blue color component.

Definition at line 131 of file [colors](#).

11.164.3.4 **Color_component const& L4Re::Video::Pixel_info::a () const [inline]**

Return the alpha color component of the pixel.

Returns

Alpha color component.

Definition at line 137 of file [colors](#).

11.164.3.5 **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 143 of file [colors](#).

11.164.3.6 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 149 of file [colors](#).

11.164.3.7 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 156 of file [colors](#).

11.164.3.8 void L4Re::Video::Pixel_info::r (Color_component const & c) [inline]

Set the red color component of the pixel.

Parameters

c Red color component.

Definition at line 162 of file [colors](#).

11.164.3.9 void L4Re::Video::Pixel_info::g (Color_component const & c) [inline]

Set the green color component of the pixel.

Parameters

c Green color component.

Definition at line 168 of file [colors](#).

11.164.3.10 void L4Re::Video::Pixel_info::b (Color_component const & c) [inline]

Set the blue color component of the pixel.

Parameters

c Blue color component.

Definition at line 174 of file [colors](#).

11.164.3.11 void L4Re::Video::Pixel_info::a (Color_component const & *c*) [inline]

Set the alpha color component of the pixel.

Parameters

c Alpha color component.

Definition at line 180 of file [colors](#).

11.164.3.12 void L4Re::Video::Pixel_info::bytes_per_pixel (unsigned char *bpp*) [inline]

Set the size of the pixel in bytes.

Parameters

bpp Size of pixel in bytes.

Definition at line 186 of file [colors](#).

11.164.3.13 bool L4Re::Video::Pixel_info::operator== (Pixel_info const & *o*) const [inline]

Compare for complete equality of the color space.

Parameters

o A [Pixel_info](#) to compare to.

Returns

true if the both [Pixel_info](#)'s are equal, false if not.

Definition at line 229 of file [colors](#).

11.164.3.14 template<typename STREAM> STREAM& L4Re::Video::Pixel_info::dump (STREAM & *s*) const [inline]

Dump information on the pixel to a stream.

Parameters

s Stream

Returns

The stream

Definition at line 240 of file [colors](#).

The documentation for this class was generated from the following file:

- [l4/re/video/colors](#)

11.165 L4Re::Util::Ref_cap< T > Struct Template Reference

Automatic capability that implements automatic free and unmap of the capability selector.

11.165.1 Detailed Description

template<typename T> struct L4Re::Util::Ref_cap< T >

Automatic capability that implements automatic free and unmap of the capability selector.

Parameters

T the 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>::Ca 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 223 of file [cap_alloc](#).

The documentation for this struct was generated from the following file:

- [l4/re/util/cap_alloc](#)

11.166 L4Re::Util::Ref_del_cap< T > Struct Template Reference

Automatic capability that implements automatic free and unmap+delete of the capability selector.

11.166.1 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.

Parameters

T the 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>::Ca 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 [263](#) of file [cap_alloc](#).

The documentation for this struct was generated from the following file:

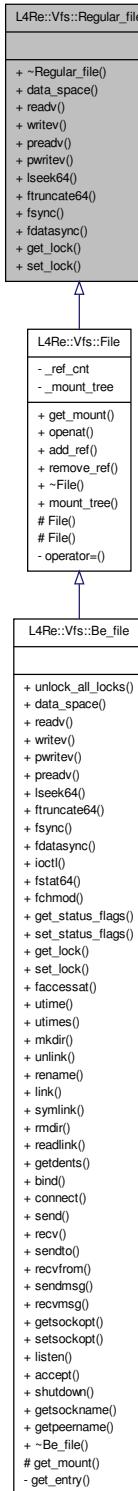
- l4/re/util/cap_alloc

11.167 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:



Public Member Functions

- virtual `L4::Cap< L4Re::Dataspace > data_space () const =0 throw ()`
Get an L4Re::Dataspace object for the file.
- virtual `ssize_t readv (const struct iovec *, int iovcnt)=0 throw ()`
Read one or more blocks of data from the file.
- virtual `ssize_t writev (const struct iovec *, int iovcnt)=0 throw ()`
Write one or more blocks of data to the file.
- virtual `off64_t lseek64 (off64_t, int)=0 throw ()`
Change the file pointer.
- virtual `int ftruncate64 (off64_t pos)=0 throw ()`
Truncate the file at the given position.
- virtual `int fsync () const =0 throw ()`
Sync the data and meta data to persistent storage.
- virtual `int fdatasync () const =0 throw ()`
Sync the data to persistent storage.
- virtual `int get_lock (struct flock64 *lock)=0 throw ()`
Test if the given lock can be placed in the file.
- virtual `int set_lock (struct flock64 *lock, bool wait)=0 throw ()`
Acquire or release the given lock on the file.

11.167.1 Detailed Description

Interface for a POSIX file that provides regular file semantics. Real objects use always the combined `L4Re::Vfs::File` interface.

Definition at line 262 of file `vfs.h`.

11.167.2 Member Function Documentation

11.167.2.1 virtual L4::Cap<L4Re::Dataspace> L4Re::Vfs::Regular_file::data_space () const throw () [pure virtual]

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 functions returns an invalid capability.

Returns

A capability to an L4Re::Dataspace, that represents the files contents in an L4Re way.

11.167.2.2 virtual ssize_t L4Re::Vfs::Regular_file::readv (const struct iovec *, int iovcnt) throw () [pure virtual]

Read one or more blocks of data from the file.

This function acts as backend for POSIX read and ready calls and reads data starting for the f_pos pointer of that open file. The file pointer is advanced according to the number of read bytes.

Returns

The number of bytes read from the file, or <0 on error.

11.167.2.3 virtual ssize_t L4Re::Vfs::Regular_file::writev (const struct iovec *, int iovcnt) throw () [pure virtual]

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.

11.167.2.4 virtual off64_t L4Re::Vfs::Regular_file::lseek64 (off64_t, int) throw () [pure virtual]

Change the file pointer.

This is the backend for POSIX seek, lseek and friends.

Returns

The new file position, or <0 on error.

11.167.2.5 virtual int L4Re::Vfs::Regular_file::ftruncate64 (off64_t pos) throw () [pure virtual]

Truncate the file at the given position.

This function is the backend for truncate and friends.

Parameters

pos The offset at which the file shall be truncated.

Returns

0 on success, or <0 on error.

11.167.2.6 virtual int L4Re::Vfs::Regular_file::fsync () const throw () [pure virtual]

Sync the data and meta data to persistent storage.

This is the backend for POSIX fsync.

11.167.2.7 virtual int L4Re::Vfs::Regular_file::fdatasync () const throw () [pure virtual]

Sync the data to persistent storage.

This is the backend for POSIX fdatasync.

11.167.2.8 virtual int L4Re::Vfs::Regular_file::get_lock (struct flock64 * lock) throw () [pure virtual]

Test if the given lock can be placed in the file.

This function is used as backend for fcntl F_GETLK commands.

Parameters

lock The lock that shall be placed on the file. The *l_type* member will contain F_UNLCK if the lock could be placed.

Returns

0 on success, <0 on error.

11.167.2.9 virtual int L4Re::Vfs::Regular_file::set_lock (struct flock64 * lock, bool wait) throw () [pure virtual]

Acquire or release the given lock on the file.

This function is used as backend for fcntl F_SETLK and F_SETLKW commands.

Parameters

lock The lock that shall be placed on the file.

wait If true, then block if there is a conflicting lock on the file.

Returns

0 on success, <0 on error.

The documentation for this class was generated from the following file:

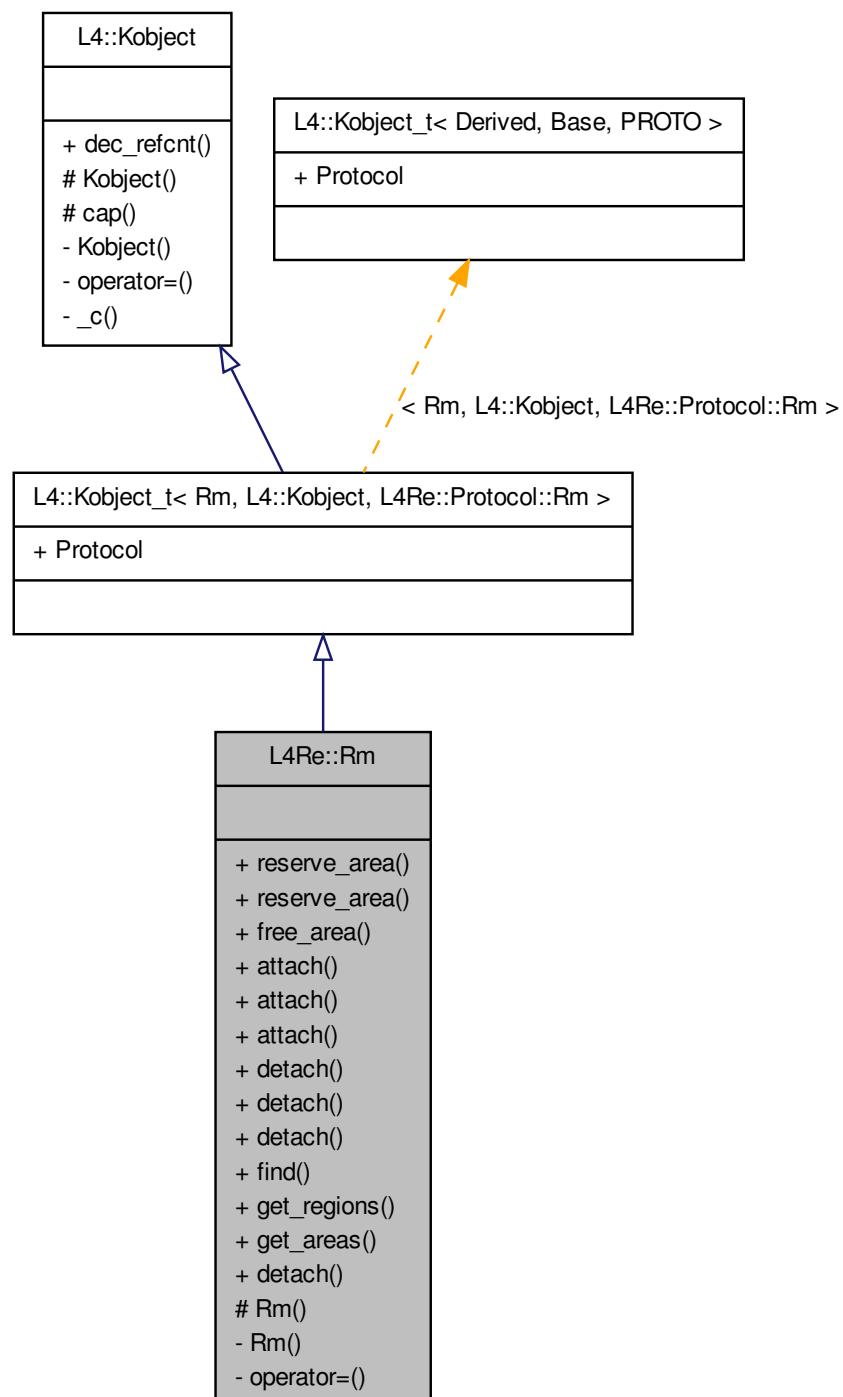
- l4/l4re_vfs/vfs.h

11.168 L4Re::Rm Class Reference

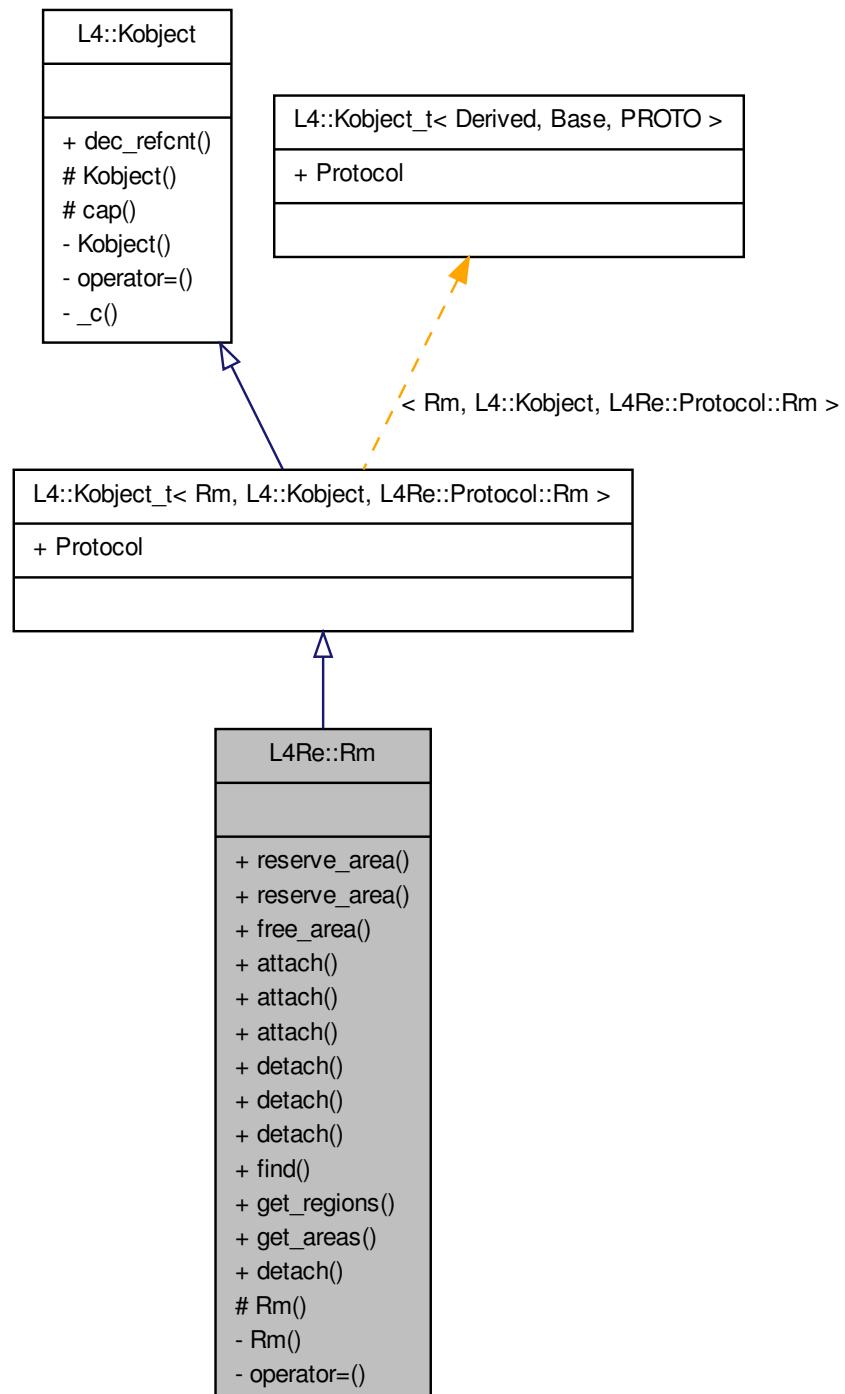
Region map.

```
#include <l4/re/rm>
```

Inheritance diagram for L4Re::Rm:



Collaboration diagram for L4Re::Rm:



Public Types

- enum `Detach_result` { `Detached_ds` = 0, `Kept_ds` = 1, `Split_ds` = 2, `Detach_again` = 4 }
Result values for detach operation.
- enum `Region_flags` {
`Read_only` = 0x01, `Detach_free` = 0x02, `Pager` = 0x04, `Reserved` = 0x08,
`Region_flags` = 0x0f }
Flags for regions.
- enum `Attach_flags` { `Search_addr` = 0x20, `In_area` = 0x40, `Eager_map` = 0x80, `Attach_flags` = 0xf0 }
Flags for attach operation.
- 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, unsigned flags=0, unsigned char align=L4_-PAGESHIFT) const throw ()
Reserve the given area in the region map.
- template<typename T >
long `reserve_area` (T **start, unsigned long size, unsigned flags=0, unsigned char align=L4_-PAGESHIFT) const throw ()
Reserve the given area in the region map.
- long `free_area` (`l4_addr_t` addr) const throw ()
Free an area from the region map.
- long `attach` (`l4_addr_t` *start, unsigned long size, unsigned long flags, `L4::Cap< Dataspace >` mem, `l4_addr_t` offs=0, unsigned char align=L4_PAGESHIFT) const throw ()
Attach a data space to a region.
- template<typename T >
long `attach` (T **start, unsigned long size, unsigned long flags, `L4::Cap< Dataspace >` mem, `l4_addr_t` offs=0, unsigned char align=L4_PAGESHIFT) const throw ()
Attach a dataspace to a region.
- int `detach` (`l4_addr_t` addr, `L4::Cap< Dataspace >` *mem, `L4::Cap< L4::Task >` const &task=This_task) const throw ()
Detach a region from the address space.
- int `detach` (void *addr, `L4::Cap< Dataspace >` *mem, `L4::Cap< L4::Task >` const &task=This_task) const throw ()
Detach 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 throw ()

Detach all regions of the specified interval.

- int `find` (`l4_addr_t` *addr, unsigned long *size, `l4_addr_t` *offset, unsigned *flags, `L4::Cap< Dataspace >` *m) throw ()

Find a region given an address and size.

11.168.1 Detailed Description

Region map.

Definition at line [69](#) of file `rm`.

11.168.2 Member Enumeration Documentation

11.168.2.1 enum L4Re::Rm::Detach_result

Result values for detach operation.

Enumerator:

Detached_ds Detached data space.

Kept_ds Kept data space.

Split_ds Splitted data space, and done.

Detach_again Detached data space, more to do.

Definition at line [76](#) of file `rm`.

11.168.2.2 enum L4Re::Rm::Region_flags

Flags for regions.

Enumerator:

Read_only Region is read-only.

Detach_free Free the portion of the data space after detach.

Pager Region has a pager.

Reserved Region is reserved (blocked).

Region_flags Mask of all region flags.

Definition at line [87](#) of file `rm`.

11.168.2.3 enum L4Re::Rm::Attach_flags

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.
- Attach_flags* Mask of all attach flags.

Definition at line 99 of file [rm](#).

11.168.2.4 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 [Detach_free](#).

Definition at line 109 of file [rm](#).

11.168.3 Member Function Documentation**11.168.3.1 long L4Re::Rm::reserve_area (l4_addr_t * start, unsigned long size, unsigned flags = 0, unsigned char align = L4_PAGESHIFT) const throw ()**

Reserve the given area in the region map.

Parameters

- start* The virtual start address of the area to reserve.
- size* The size of the area to reserve (in bytes).
- flags* Flags for the reserved area (see [Region_flags](#) and [Attach_flags](#)).
- align* Alignment of area if searched as bits (log2 value).

Return values

- start* Start of address.

Returns

0 on success, <0 on error

- [-L4_EADDRNOTAVAIL](#)
- IPC errors

This function reserves an area within the virtual address space implemented by the region map. There are two kinds of areas available:

- Reserved areas (*flags* = [Reserved](#)), where no data spaces can be attached
- Special purpose areas (*flags* = 0), where data spaces can be attached to the area via the [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* = `Search_addr`), the space between *start* and the end of the virtual address space is searched.

Definition at line 30 of file [rm_impl.h](#).

11.168.3.2 template<typename T > long L4Re::Rm::reserve_area (T ** *start*, unsigned long *size*, unsigned *flags* = 0, unsigned char *align* = `L4_PAGESHIFT`) const throw () [inline]

Reserve the given area in the region map.

Parameters

start The virtual start address of the area to reserve.

size The size of the area to reserve (in bytes).

flags Flags for the reserved area (see [Region_flags](#) and [Attach_flags](#)).

align Alignment of area if searched as bits (log2 value).

Return values

start Start of address.

Returns

0 on success, <0 on error

- [-L4_EADDRNOTAVAIL](#)
- IPC errors

For more information, please refer to the analogous function

See also

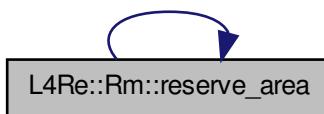
[L4Re::Rm::reserve_area](#).

Definition at line 241 of file [rm](#).

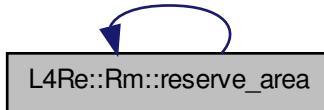
References [reserve_area\(\)](#).

Referenced by [reserve_area\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.168.3.3 long L4Re::Rm::free_area (l4_addr_t *addr*) const throw ()

Free an area from the region map.

Parameters

addr An address within the area to free.

Returns

0 on success, <0 on error

- [-L4_ENOENT](#)
- 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.

Definition at line 44 of file [rm_impl.h](#).

11.168.3.4 long L4Re::Rm::attach (l4_addr_t * *start*, unsigned long *size*, unsigned long *flags*, L4::Cap< Dataspace > *mem*, l4_addr_t *offs* = 0, unsigned char *align* = L4_PAGESHIFT) const throw ()

Attach a data space to a region.

Parameters

start Virtual start address

size Size of the data space to attach (in bytes)

flags Flags, see [Attach_flags](#) and [Region_flags](#)

mem Data space

offs Offset into the data space to use

align Alignment of the virtual region, log2-size, default: a page ([L4_PAGESHIFT](#)), Only meaningful if the [Search_addr](#) flag is used.

Return values

start Start of region if [Search_addr](#) was used.

Returns

0 on success, <0 on error

- [-L4_ENOENT](#)
- [-L4_EPERM](#)
- [-L4_EINVAL](#)
- [-L4_EADDRNOTAVAIL](#)
- 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 [find](#)).

Definition at line 52 of file [rm_impl.h](#).

11.168.3.5 `template<typename T> long L4Re::Rm::attach (T ** start, unsigned long size, unsigned long flags, L4::Cap< Dataspace > * mem, l4_addr_t offs = 0, unsigned char align = L4_PAGESHIFT) const throw () [inline]`

Attach a dataspace to a region.

See also

[attach](#)

Definition at line 301 of file [rm](#).

11.168.3.6 `int L4Re::Rm::detach (l4_addr_t addr, L4::Cap< Dataspace > * mem, L4::Cap< L4::Task > const & task = This_task) const throw () [inline]`

Detach a region from the address space.

Parameters

addr Virtual address of region, any address within the region is valid.

Return values

mem [Dataspace](#) that is affected. Give 0 if not interested.

Parameters

task If given, task specifies the task where the pages are unmapped. Give 0 for none. Default is current task.

Returns

[Detach_result](#) on success, <0 on error

- [-L4_ENOENT](#)
- 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 [451](#) of file [rm](#).

11.168.3.7 int L4Re::Rm::detach (void * *addr*, L4::Cap< Dataspace > * *mem*, L4::Cap< L4::Task > const & *task* = *This_task*) const throw () [inline]

Detach a region from the address space.

Parameters

addr Virtual address of region, any address within the region is valid.

Return values

mem [Dataspace](#) that is affected. Give 0 if not interested.

Parameters

task If given, task specifies the task where the pages are unmapped. Give 0 for none. Default is current task.

Returns

[Detach_result](#) on success, <0 on error

- [-L4_ENOENT](#)
- IPC errors

Frees a region in the virtual address space given by *addr* (void pointer type). The corresponding part of the address space is now available again.

Definition at line [456](#) of file [rm](#).

11.168.3.8 int L4Re::Rm::detach (l4_addr_t *start*, unsigned long *size*, L4::Cap< Dataspace > * *mem*, L4::Cap< L4::Task > const & *task*) const throw () [inline]

Detach all regions of the specified interval.

Parameters

start Start of area to detach, must be within region.

size Size of of area to detach (in bytes).

Return values

mem Dataspace that is affected. Give 0 if not interested.

Parameters

task Specifies the task where the pages are unmapped. Give 0 for none.

Returns

Detach_result on success, <0 on error

- -L4_ENOENT
- 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 461 of file [rm](#).

**11.168.3.9 int L4Re::Rm::find (l4_addr_t * *addr*, unsigned long * *size*, l4_addr_t * *offset*,
unsigned * *flags*, L4::Cap< Dataspace > * *m*) throw ()**

Find a region given an address and size.

Parameters

addr Address to look for

size Size of the area to look for (in bytes).

Return values

addr Start address of the found region.

size Size of the found region (in bytes).

offset Offset at the beginning of the region within the associated dataspace.

flags Region flags, see [Region_flags](#).

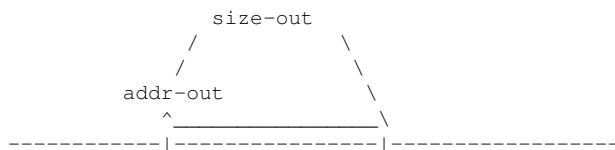
m Associated dataspace or paging service.

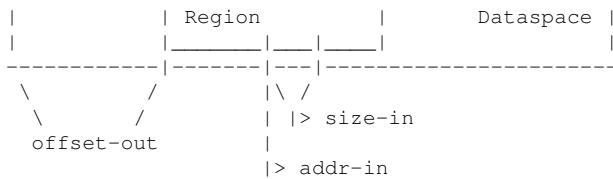
Returns

0 on success, <0 on error

- -L4_EPERM: not allowed
- -L4_ENOENT: not found
- IPC errors

This function returns the properties of the region that contains the area described by the *addr* and *size* parameter.



**Note**

The value of the size input parameter should be 1 to assure that a region can be determined unambiguously.

The documentation for this class was generated from the following files:

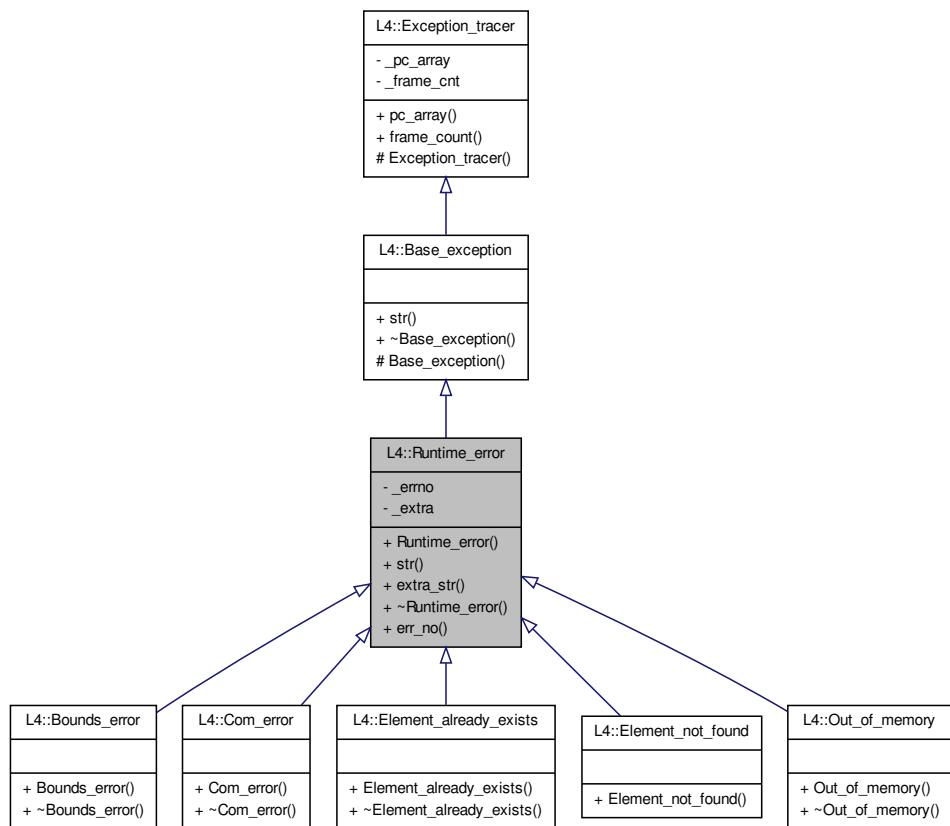
- l4/re/rm
- l4/re/impl/rm_impl.h

11.169 L4::Runtime_error Class Reference

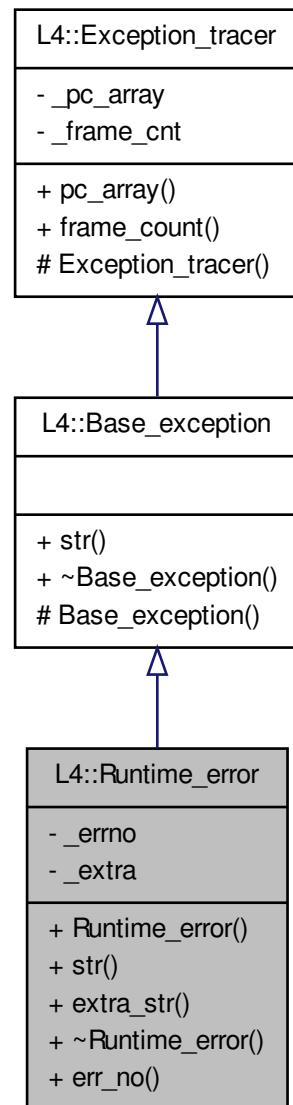
Exception for an abstract runtime error.

```
#include <l4/cxx/exceptions>
```

Inheritance diagram for L4::Runtime_error:



Collaboration diagram for L4::Runtime_error:



Public Member Functions

- `char const * str () const throw ()`

Should return a human readable string for the exception.

11.169.1 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 [136](#) of file [exceptions](#).

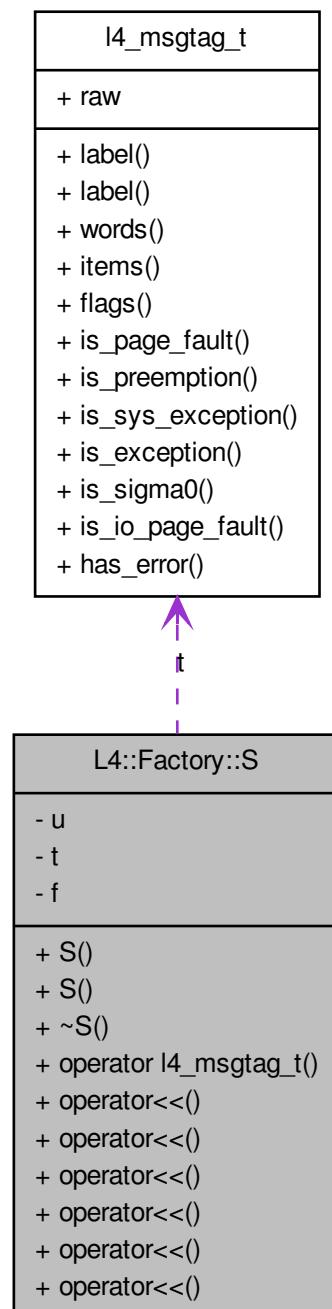
The documentation for this class was generated from the following file:

- [l4/cxx/exceptions](#)

11.170 L4::Factory::S Class Reference

Stream class for the [create\(\)](#) argument stream.

Collaboration diagram for L4::Factory::S:



Public Member Functions

- **S (S const &o)**
create a copy.
- **S (l4_cap_idx_t f, long obj, L4::Cap< L4::Kobject > target, l4_utcb_t *utcb) throw ()**
create a stream for a specific `create()` call.
- **~S ()**
Commit the operation in the destructor to have a cool syntax for `create()`.
- **operator l4_mshtag_t ()**
Explicitely commits the operation and returns the result.
- **S & operator<< (l4_mword_t i)**
Put a single l4_mword_t as next argument.
- **S & operator<< (l4_umword_t i)**
Put a single l4_umword_t as next argument.
- **S & operator<< (char const *s)**
Add a zero-terminated string as next argument.
- **S & operator<< (Lstr const &s)**
Add a pascal string as next argument.
- **S & operator<< (Nil)**
Add an empty argument.
- **S & operator<< (l4_fpage_t d)**
Add a flex page as next argument.

11.170.1 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 82 of file `factory`.

11.170.2 Constructor & Destructor Documentation

11.170.2.1 L4::Factory::S::S (l4_cap_idx_t f, long obj, L4::Cap< L4::Kobject > target, l4_utcb_t * utcb) throw () [inline]

create a stream for a specific `create()` call.

Parameters

f is the capability for the factory object (`L4::Factory`).

obj is the protocol ID to describe the type of the object that shall be created.

target is the capability selector for the new object.

utc is the UTCB that shall be used for the operation.

Definition at line 105 of file [factory](#).

11.170.3 Member Function Documentation

11.170.3.1 L4::Factory::S::operator l4_msntag_t() [inline]

Explicitly commits the operation and returns the result.

Returns

The result of the [create\(\)](#) operation.

Definition at line 124 of file [factory](#).

References [l4_msntag_t::raw](#).

11.170.3.2 S & L4::Factory::S::operator<< (l4_mword_t i) [inline]

Put a single l4_mword_t as next argument.

Parameters

i is the value to add as next argument.

Definition at line 135 of file [factory](#).

11.170.3.3 S & L4::Factory::S::operator<< (l4_umword_t i) [inline]

Put a single l4_umword_t as next argument.

Parameters

i is the value to add as next argument.

Definition at line 145 of file [factory](#).

11.170.3.4 S & L4::Factory::S::operator<< (char const * s) [inline]

Add a zero-terminated string as next argument.

Parameters

s is the string to add as next argument.

Definition at line 155 of file [factory](#).

11.170.3.5 S& L4::Factory::S::operator<<(Lstr const & s) [inline]

Add a pascal string as next argument.

Parameters

s is the string to add as next argument.

Definition at line 165 of file [factory](#).

References [L4::Factory::Lstr::len](#), and [L4::Factory::Lstr::s](#).

11.170.3.6 S& L4::Factory::S::operator<<(14_fpage_t d) [inline]

Add a flex page as next argument.

Parameters

d is the flex page to add (there will be no map operation).

Definition at line 184 of file [factory](#).

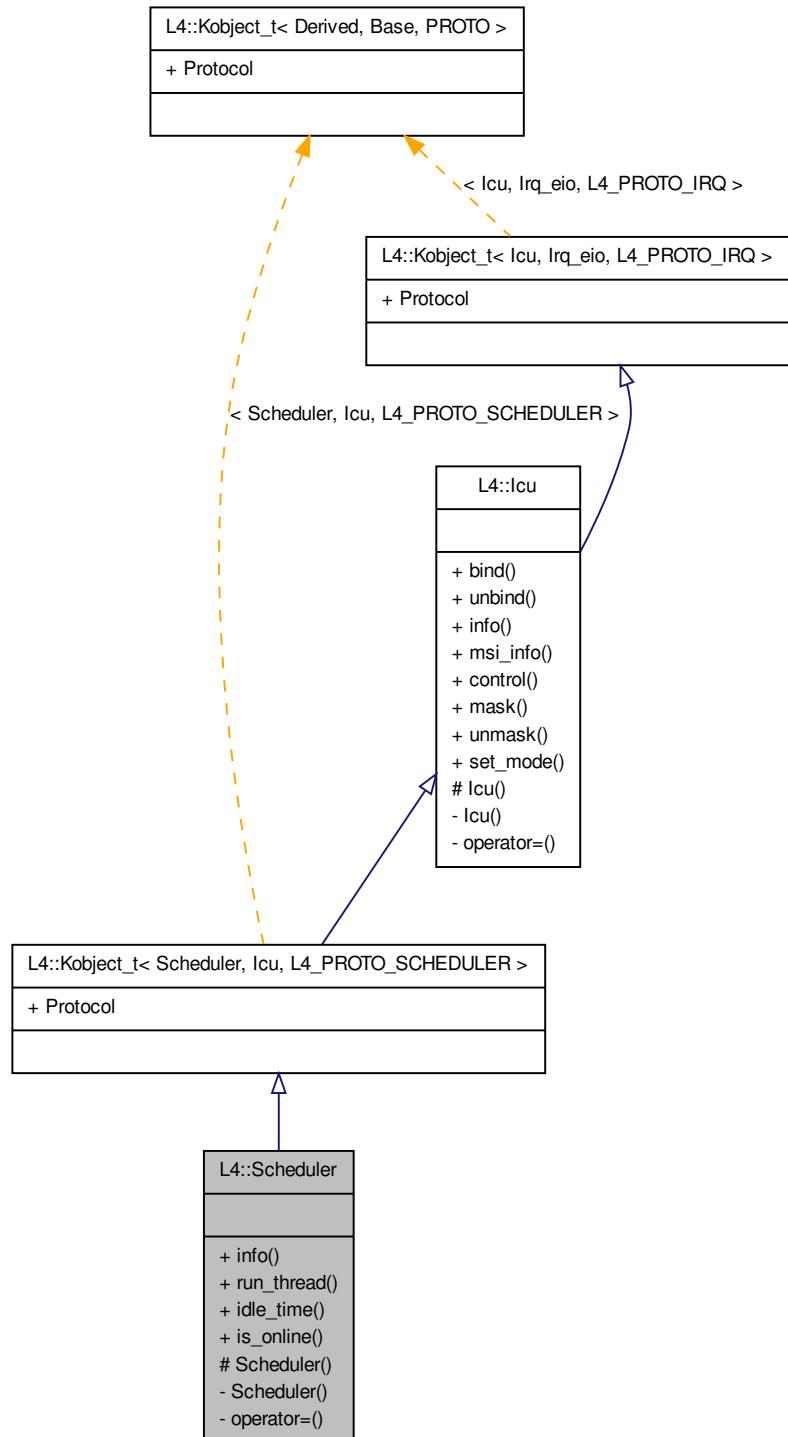
The documentation for this class was generated from the following file:

- 14/sys/factory

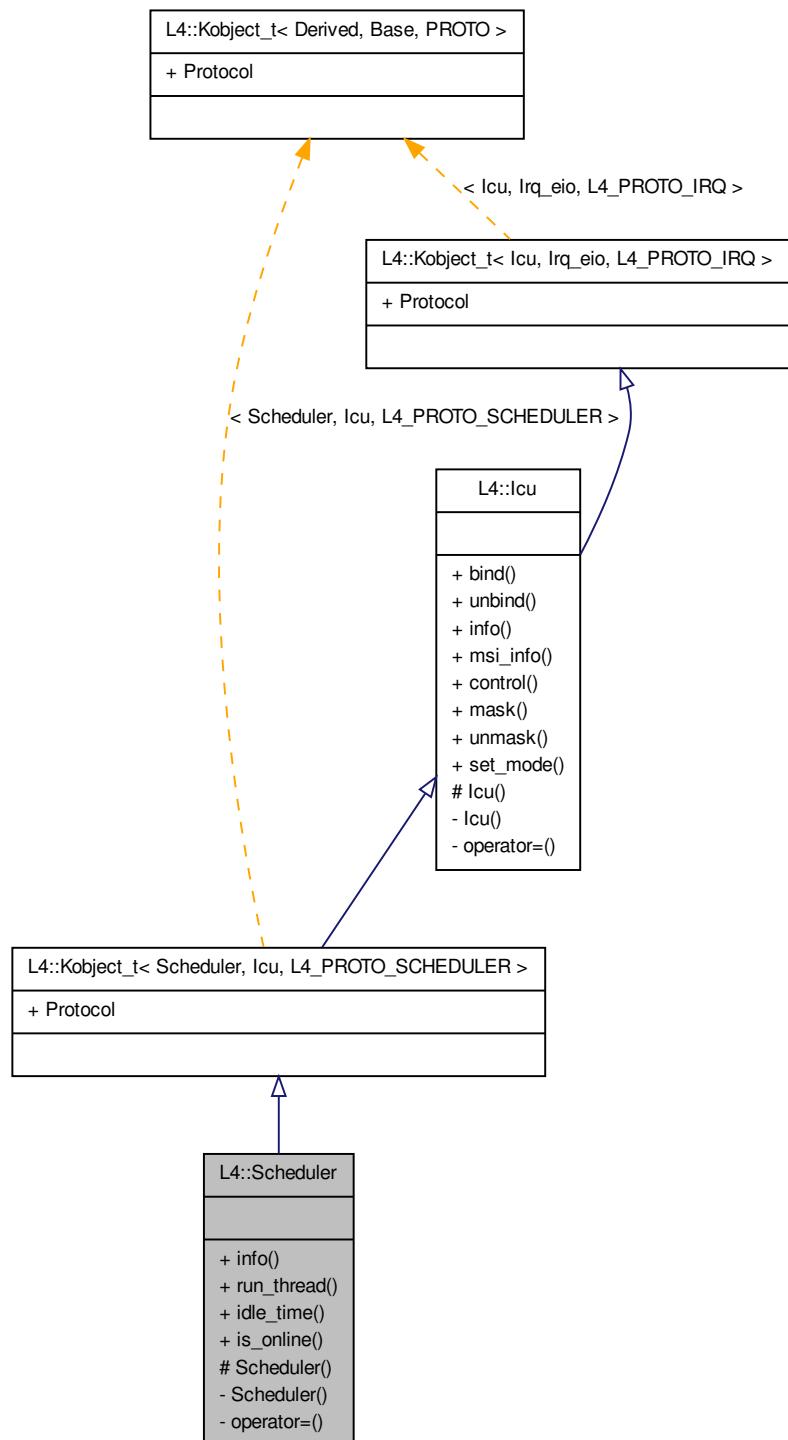
11.171 L4::Scheduler Class Reference

[Scheduler](#) object.

Inheritance diagram for L4::Scheduler:



Collaboration diagram for L4::Scheduler:



Public Member Functions

- `l4_mshtag_t info (l4_umword_t *cpu_max, l4_sched_cpu_set_t *cpus, l4_utcb_t *utcb=l4_utcb()) const throw ()`
- `l4_mshtag_t run_thread (Cap< Thread > const &thread, l4_sched_param_t const &sp, l4_utcb_t *utcb=l4_utcb()) const throw ()`
- `l4_mshtag_t idle_time (l4_sched_cpu_set_t const &cpus, l4_utcb_t *utcb=l4_utcb()) const throw ()`
- `bool is_online (l4_umword_t cpu, l4_utcb_t *utcb=l4_utcb()) const throw ()`

11.171.1 Detailed Description

`Scheduler` object. `#include <l4/sys/scheduler>`

See also

[Scheduler](#) for an overview description.

Definition at line 39 of file `scheduler`.

11.171.2 Member Function Documentation

11.171.2.1 `l4_mshtag_t L4::Scheduler::info (l4_umword_t * cpu_max, l4_sched_cpu_set_t * cpus, l4_utcb_t * utcb = l4_utcb()) const throw () [inline]`

Get scheduler information.

Parameters

`scheduler` Scheduler object.

Return values

`cpu_max` maximum number of CPUs ever available.

Parameters

`cpus` `cpus.offset` is first CPU of interest. `cpus.granularity` (see `l4_sched_cpu_set_t`).

Return values

`cpus` `cpus.map` Bitmap of online CPUs.

Returns

0 on success, <0 error code otherwise.

Note

`scheduler` is the implicit `this` pointer.

Definition at line 36 of file `scheduler`.

**11.171.2.2 l4_mshtag_t L4::Scheduler::run_thread (Cap< Thread > const & *thread*,
 l4_sched_param_t const & *sp*, l4_utcb_t * *utcb* = **l4_utcb()**) const throw ()
 [inline]**

Run a thread on a Scheduler.

Parameters

scheduler Scheduler object.

thread Thread to run.

sp Scheduling parameters.

Returns

0 on success, <0 error code otherwise.

Note

scheduler is the implicit *this* pointer.

Definition at line 44 of file [scheduler](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



**11.171.2.3 l4_mshtag_t L4::Scheduler::idle_time (l4_sched_cpu_set_t const & *cpus*, l4_utcb_t *
utcb = **l4_utcb()**) const throw () [inline]**

Query idle time of a CPU, in μ s.

Parameters

scheduler Scheduler object.

cpus Set of CPUs to query.

The consumed time is returned as l4_kernel_clock_t at UTCB message register 0.

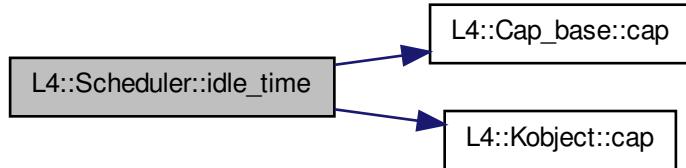
Note

scheduler is the implicit *this* pointer.

Definition at line 53 of file [scheduler](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.171.2.4 `bool L4::Scheduler::is_online (l4_umword_t cpu, l4_utcb_t * utcb = l4_utcb())
const throw () [inline]`

Query if a CPU is online.

Parameters

scheduler Scheduler object.

cpu CPU number.

Returns

true if online, false if not (or any other query error).

Note

scheduler is the implicit *this* pointer.

Definition at line 62 of file [scheduler](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

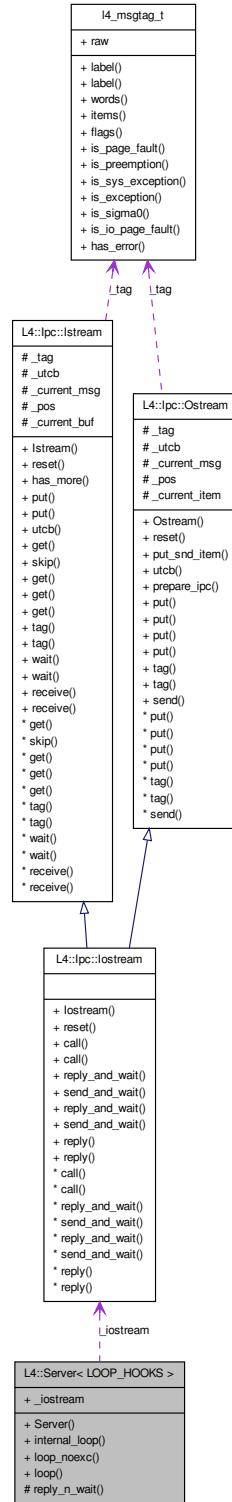
- l4/sys/scheduler

11.172 L4::Server< LOOP_HOOKS > Class Template Reference

Basic server loop for handling client requests.

Inherited by L4Re::Util::Registry_server< LOOP_HOOKS >.

Collaboration diagram for L4::Server< LOOP_HOOKS >:



Public Member Functions

- **Server (l4_utcb_t *utcbs)**
Initializes the server loop and its underlying [Ipc::Iostream](#).
- template<typename DISPATCH >
void internal_loop (DISPATCH dispatch)
The server loop.

11.172.1 Detailed Description

template<typename LOOP_HOOKS = Ipc_svr::Default_loop_hooks> class L4::Server< LOOP_HOOKS >

Basic server loop for handling client requests.

Parameters

LOOP_HOOKS the server inherits from LOOP_HOOKS and calls the hooks defined in LOOP_HOOKS in the server loop. See [Ipc_svr::Default_loop_hooks](#), [Ipc_svr::Ignore_errors](#), [Ipc_svr::Default_timeout](#), [Ipc_svr::Compound_reply](#), and [Ipc_svr::Default_setup_wait](#).

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 180 of file [ipc_server](#).

11.172.2 Constructor & Destructor Documentation

11.172.2.1 template<typename LOOP_HOOKS = Ipc_svr::Default_loop_hooks> L4::Server< LOOP_HOOKS >::Server (l4_utcb_t * *utcbs*) [inline, explicit]

Initializes the server loop and its underlying [Ipc::Iostream](#).

Parameters

utcbs The UTCB of the thread running the server loop.

Definition at line 188 of file [ipc_server](#).

11.172.3 Member Function Documentation

11.172.3.1 template<typename L > template<typename DISPATCH > void L4::Server< L >::internal_loop (DISPATCH *dispatch*) [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 240 of file [ipc_server](#).

References [l4_mshtag_t::has_error\(\)](#).

Here is the call graph for this function:



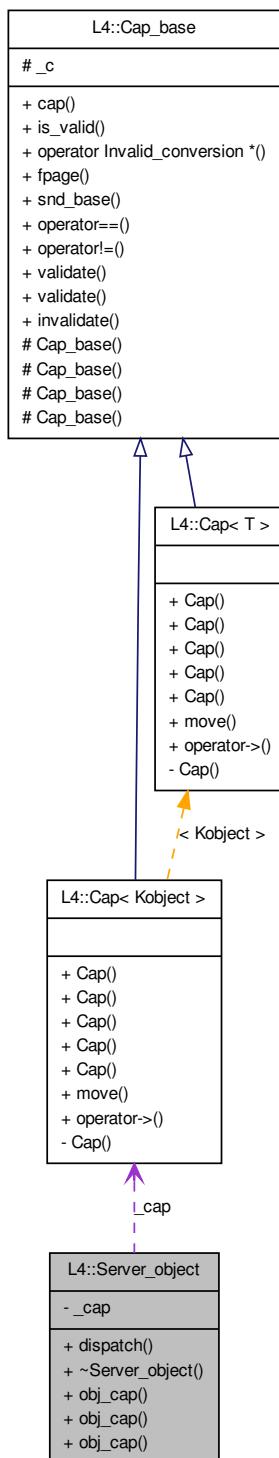
The documentation for this class was generated from the following file:

- l4/cxx/ipc_server

11.173 L4::Server_object Class Reference

Abstract server object to be used with [L4::Server](#) and [L4::Basic_registry](#).

Collaboration diagram for L4::Server_object:



Public Member Functions

- virtual int **dispatch** (unsigned long obj, [Ipc::Iostream](#) &ios)=0

The abstract handler for client requests to the object.

11.173.1 Detailed Description

Abstract server object to be used with [L4::Server](#) and [L4::Basic_registry](#). 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 272 of file [ipc_server](#).

11.173.2 Member Function Documentation

11.173.2.1 virtual int L4::Server_object::dispatch (unsigned long *obj*, [Ipc::Iostream](#) & *ios*) [pure virtual]

The abstract handler for client requests to the object.

Parameters

obj The object ID used for looking up the object.

ios The [Ipc::Iostream](#) for reading the request and writing the reply.

Returns

Reply, No_reply, or Invalid_opcode.

This function must be implemented by application specific server objects. The implementation must unmarshal 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.

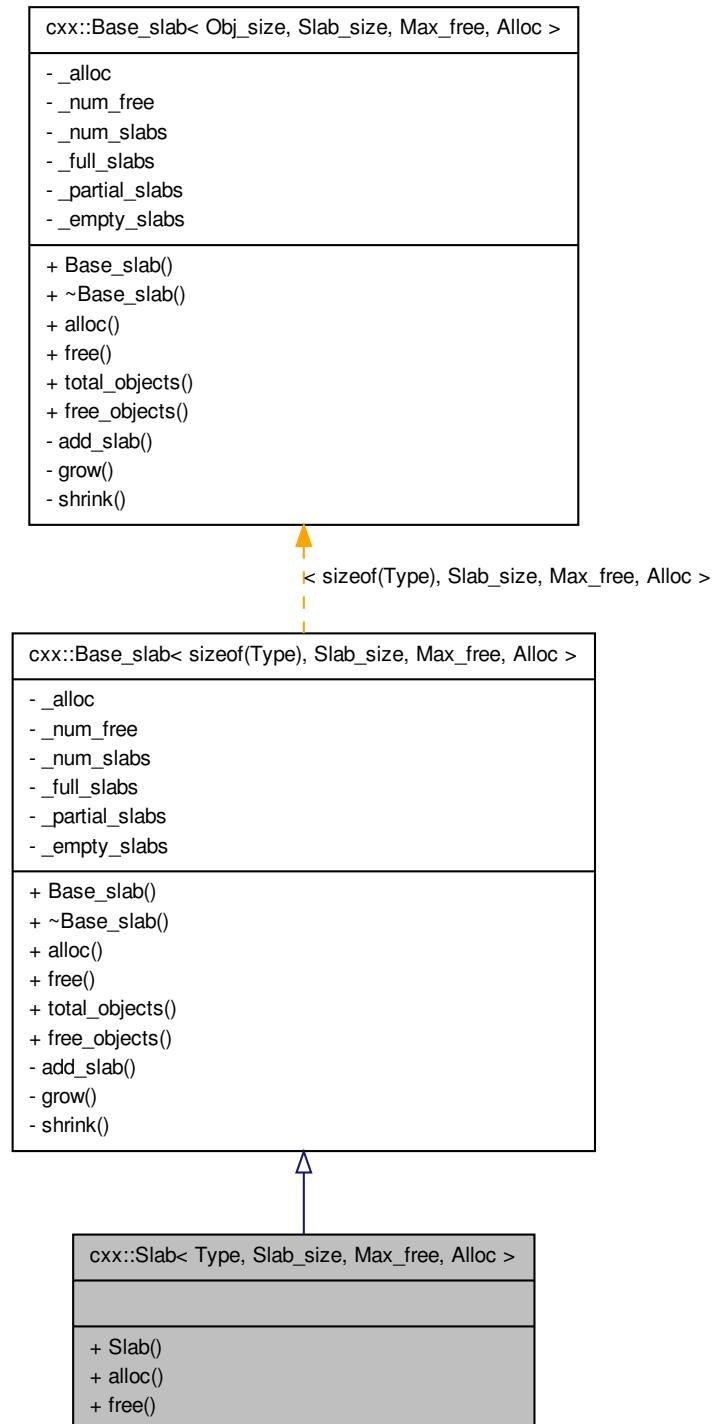
The documentation for this class was generated from the following file:

- l4/cxx/ipc_server

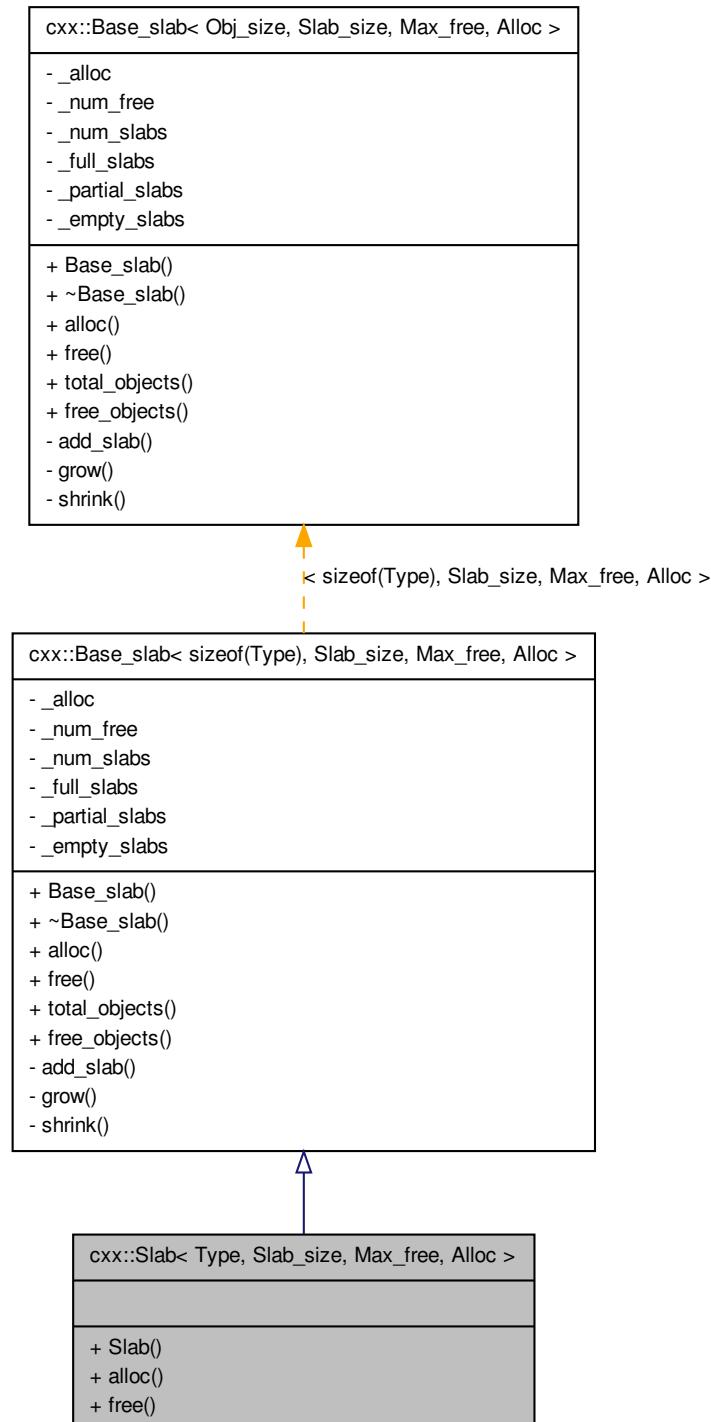
11.174 cxx::Slab< Type, Slab_size, Max_free, Alloc > Class Template Reference

[Slab](#) allocator for object of type *Type*.

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 () throw ()`
Allocate an object of type Type.
- `void free (Type *o) throw ()`
Free the object addressed by o.

11.174.1 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*.

Parameters

- Type* the type of the objects to manage.
Slab_size size of a slab cache.
Max_free the maximum number of free slab caches.
Alloc the allocator for the slab caches.

Definition at line 297 of file [slab_alloc](#).

11.174.2 Member Function Documentation

11.174.2.1 `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 () throw () [inline]`

Allocate an object of type *Type*.

Returns

A pointer to the object just allocated, or 0 on failure.

Reimplemented from `cxx::Base_slab< sizeof(Type), Slab_size, Max_free, Alloc >`.

Definition at line 314 of file [slab_alloc](#).

11.174.2.2 `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) throw () [inline]`

Free the object addressed by *o*.

Parameters

- o* The pointer to the object to free.

Precondition

The object must have been allocated with this allocator.

Definition at line 325 of file [slab_alloc](#).

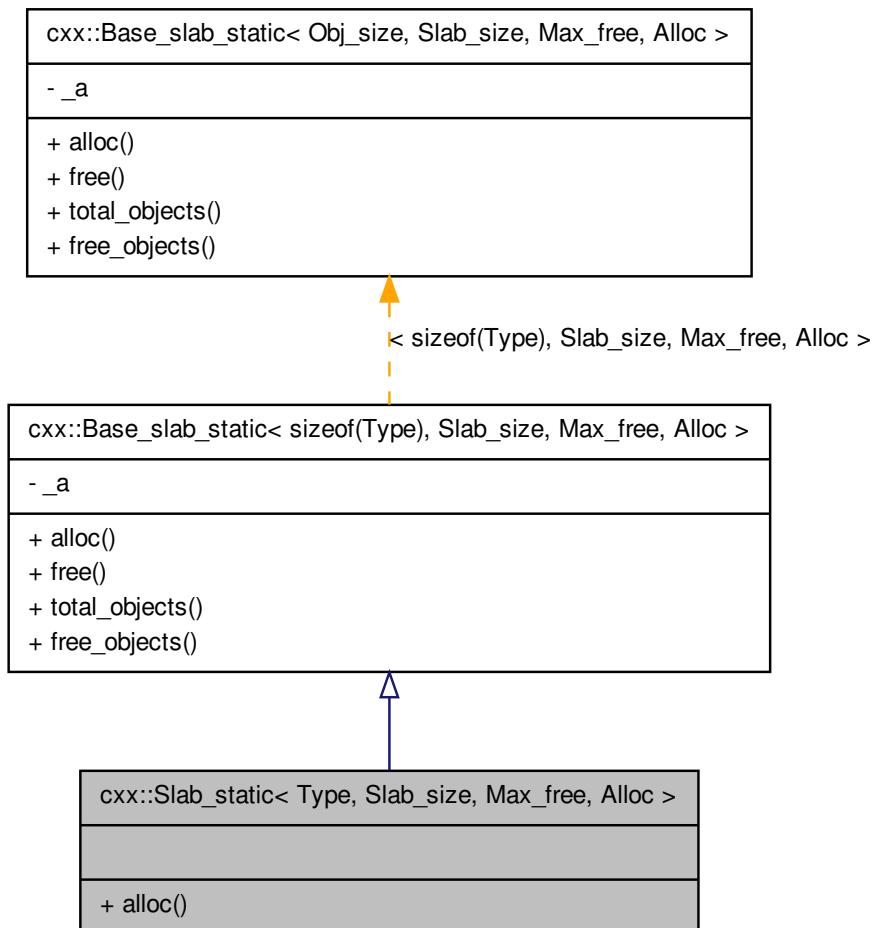
The documentation for this class was generated from the following file:

- 14/cxx/slab_alloc

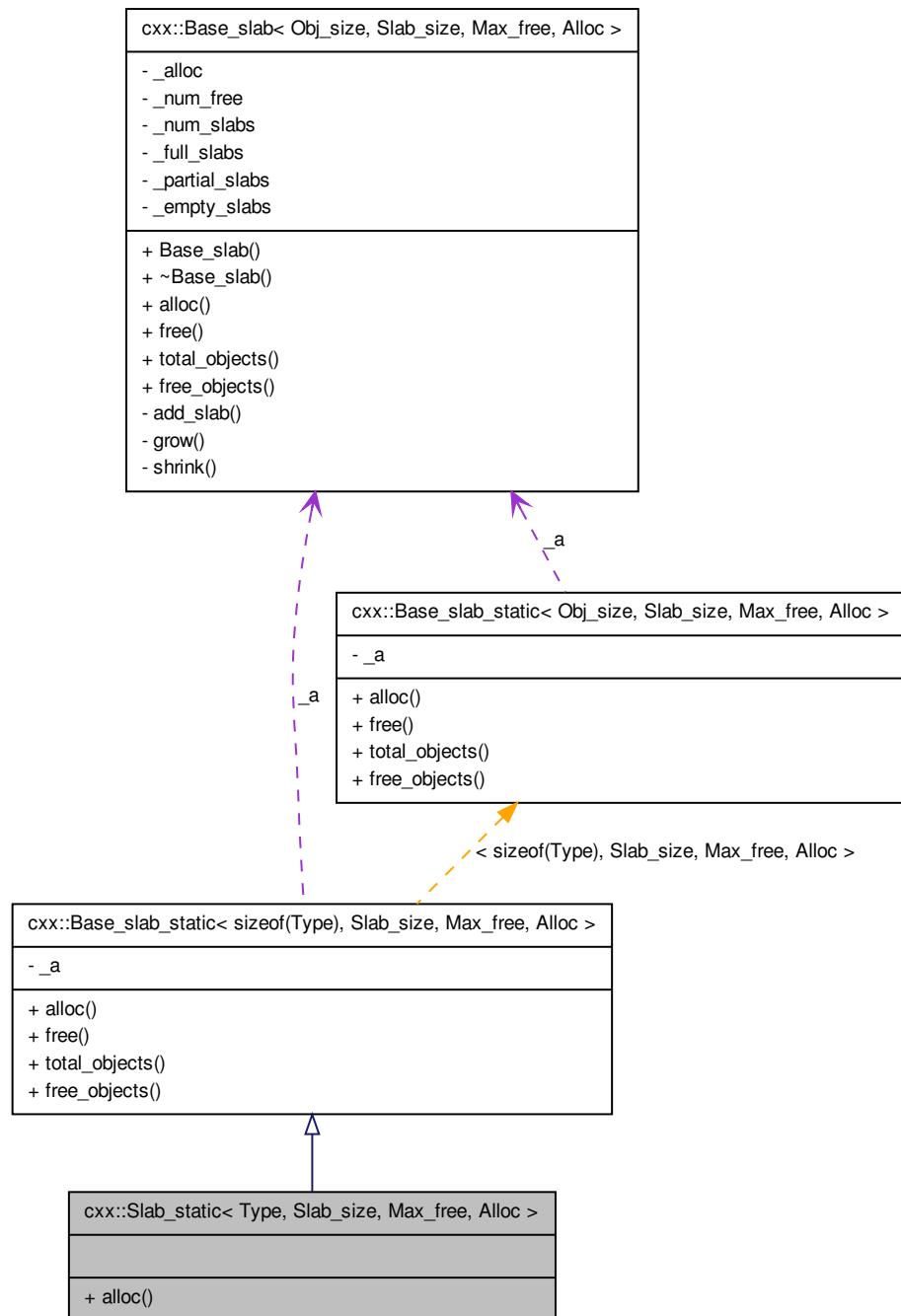
11.175 `cxx::Slab_static< Type, Slab_size, Max_free, Alloc >` Class Template Reference

Merged slab allocator (allocators for objects of the same size are merged together).

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 () throw ()`

Allocate an object of type Type.

11.175.1 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).

Parameters

Type The type of the objects to manage.

Slab_size The size of a slab cache.

Max_free The maximum number of free slab caches.

Alloc The allocator for the slab caches.

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 415 of file [slab_alloc](#).

11.175.2 Member Function Documentation

```
11.175.2.1 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 ( ) throw () [inline]
```

Allocate an object of type *Type*.

Returns

A pointer to the just allocated object, or 0 if failure.

Reimplemented from [cxx::Base_slab_static< sizeof\(Type\), Slab_size, Max_free, Alloc >](#).

Definition at line 425 of file [slab_alloc](#).

The documentation for this class was generated from the following file:

- [l4/cxx/slab_alloc](#)

11.176 L4::Ipc::Small_buf Class Reference

A receive item for receiving a single capability.

11.176.1 Detailed Description

A receive item for receiving a single capability. This class is the main abstraction for receiving capabilities via [Ipc::Istream](#). To receive a capability an instance of [Small_buf](#) that refers to an empty capability slot must be inserted into the [Ipc::Istream](#) before the receive operation.

Definition at line [256](#) of file [ipc_stream](#).

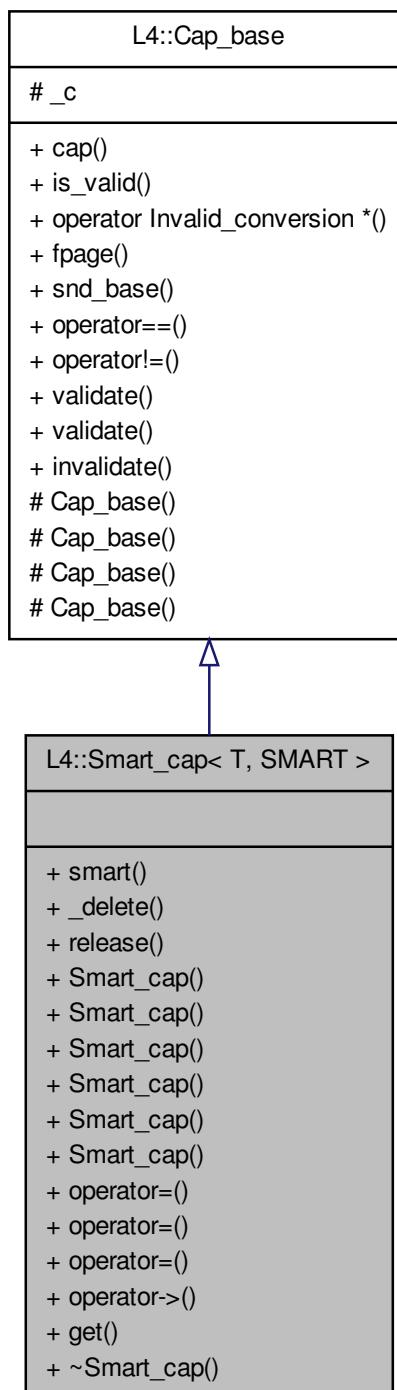
The documentation for this class was generated from the following file:

- [14/cxx/ipc_stream](#)

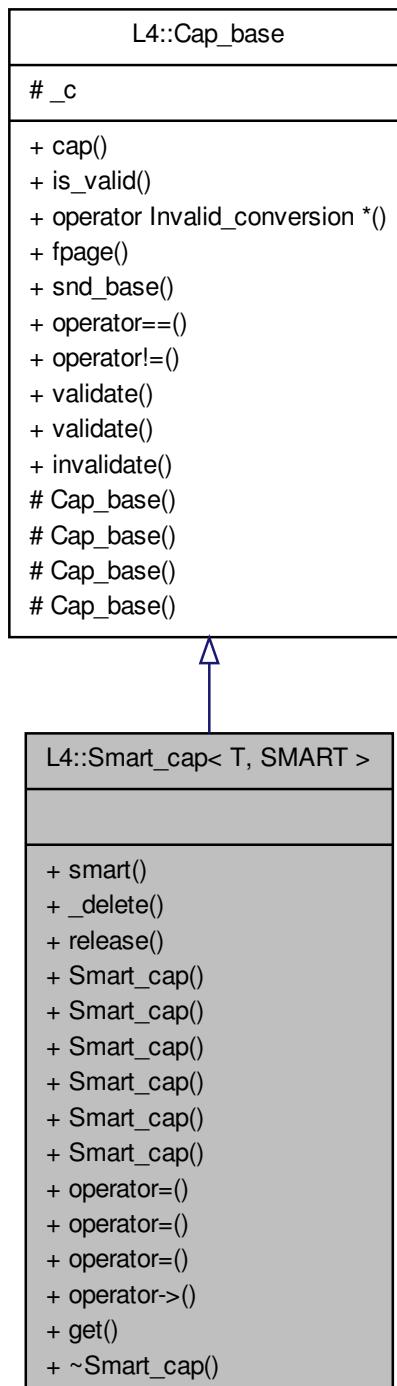
11.177 L4::Smart_cap< T, SMART > Class Template Reference

Smart capability class.

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) throw ()

Internal Constructor, use to generate a capability from a this pointer.

- Cap< T > **operator->** () const throw ()

Member access of a T.

11.177.1 Detailed Description

template<typename T, typename SMART> class L4::Smart_cap< T, SMART >

Smart capability class.

Definition at line 36 of file [smart_capability](#).

11.177.2 Constructor & Destructor Documentation

11.177.2.1 template<typename T, typename SMART> template<typename O > L4::Smart_cap< T, SMART >::Smart_cap (Cap< O > const & p) throw () [inline]

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 object that is an [L4::Kobject](#). Do *never* use this constructor for something else!

Parameters

p The *this* pointer of the [Kobject](#) or derived object

Definition at line 67 of file [smart_capability](#).

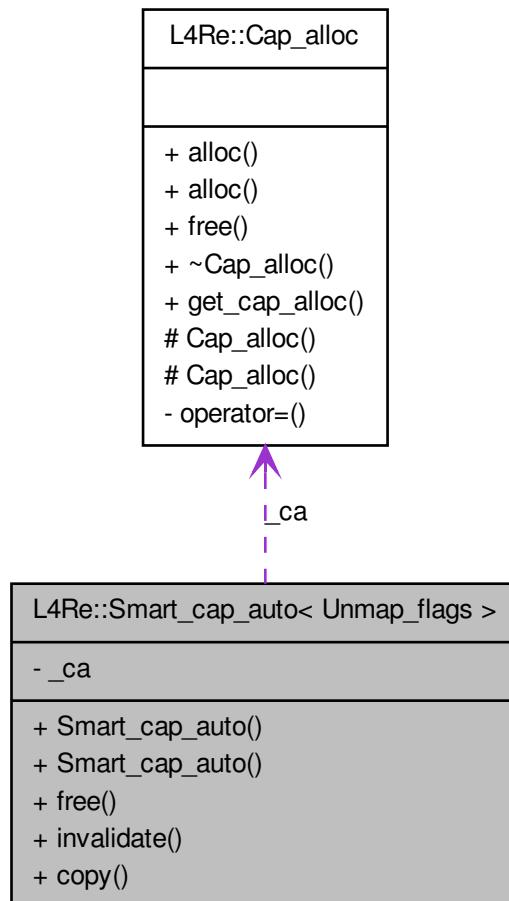
The documentation for this class was generated from the following file:

- l4/sys/smart_capability

11.178 L4Re::Smart_cap_auto< Unmap_flags > Class Template Reference

Helper for Auto_cap and Auto_del_cap.

Collaboration diagram for L4Re::Smart_cap_auto< Unmap_flags >:



11.178.1 Detailed Description

`template<unsigned long Unmap_flags = L4_FP_ALL_SPACES> class L4Re::Smart_cap_auto< Unmap_flags >`

Helper for Auto_cap and Auto_del_cap.

Definition at line 109 of file [cap_alloc](#).

The documentation for this class was generated from the following file:

- [l4/re/cap_alloc](#)

11.179 L4Re::Util::Smart_cap_auto< Unmap_flags > Class Template Reference

Helper for [Auto_cap](#) and [Auto_del_cap](#).

Static Public Member Functions

- static void [free \(L4::Cap_base &c\)](#)
free operation for L4::Smart_cap.
- static void [invalidate \(L4::Cap_base &c\)](#)
invalidate operation for L4::Smart_cap.
- static [L4::Cap_base copy \(L4::Cap_base const &src\)](#)
copy operation for L4::Smart_cap.

11.179.1 Detailed Description

template<unsigned long Unmap_flags = L4_FP_ALL_SPACES> class L4Re::Util::Smart_cap_auto< Unmap_flags >

Helper for [Auto_cap](#) and [Auto_del_cap](#).

Definition at line 54 of file [cap_alloc](#).

The documentation for this class was generated from the following file:

- [l4/re/util/cap_alloc](#)

11.180 L4Re::Util::Smart_count_cap< Unmap_flags > Class Template Reference

Helper for [Ref_cap](#) and [Ref_del_cap](#).

Static Public Member Functions

- static void [free \(L4::Cap_base &c\)](#)
free operation for L4::Smart_cap (decrement ref count and delete if 0).
- static void [invalidate \(L4::Cap_base &c\)](#)
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).

11.180.1 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 [94](#) of file [cap_alloc](#).

The documentation for this class was generated from the following file:

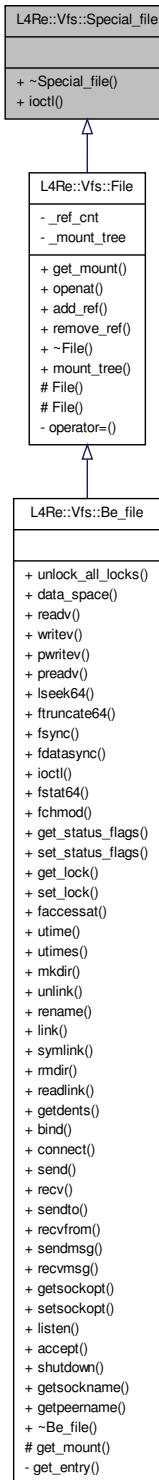
- [l4/re/util/cap_alloc](#)

11.181 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:



Public Member Functions

- virtual int **ioctl** (unsigned long cmd, va_list args)=0 throw ()

The famous IO control.

11.181.1 Detailed Description

Interface for a POSIX file that provides special file semantics. Real objects use always the combined [L4Re::Vfs::File](#) interface.

Definition at line 395 of file [vfs.h](#).

11.181.2 Member Function Documentation

11.181.2.1 virtual int L4Re::Vfs::Special_file::ioctl (**unsigned long cmd**, **va_list args**) throw () [pure virtual]

The famous IO control.

Backend for POSIX generic object invocation ioctl.

Parameters

- cmd** The ioctl command.
args The arguments for the ioctl, usually some kind of pointer.

Returns

≥ 0 on success, or <0 on error.

The documentation for this class was generated from the following file:

- [l4/l4re_vfs/vfs.h](#)

11.182 L4vcpu::State Class Reference

C++ implementation of state word in the vCPU area.

Public Member Functions

- **State** (l4vcpu_state_t v)

Initialize state.
- void **add** (unsigned bits) throw ()

Add flags.
- void **clear** (unsigned bits) throw ()

Clear flags.

- void `set` (l4vcpu_state_t *v*) throw ()
Set flags.

11.182.1 Detailed Description

C++ implementation of state word in the vCPU area.

Definition at line 30 of file `vcpu`.

11.182.2 Constructor & Destructor Documentation

11.182.2.1 L4vcpu::State::State (l4vcpu_state_t *v*) [inline, explicit]

Initialize state.

Parameters

v Initial state.

Definition at line 33 of file `vcpu`.

11.182.3 Member Function Documentation

11.182.3.1 void L4vcpu::State::add (unsigned *bits*) throw () [inline]

Add flags.

Parameters

bits Bits to add to the word.

Definition at line 40 of file `vcpu`.

11.182.3.2 void L4vcpu::State::clear (unsigned *bits*) throw () [inline]

Clear flags.

Parameters

bits Bits to clear in the word.

Definition at line 47 of file `vcpu`.

11.182.3.3 void L4vcpu::State::set (l4vcpu_state_t *v*) throw () [inline]

Set flags.

Parameters

v Set the word to the value of *v*.

Definition at line 54 of file [vcpu](#).

The documentation for this class was generated from the following file:

- l4/vcpu/vcpu

11.183 L4Re::Dataspace::Stats Struct Reference

Information about the data space.

Data Fields

- unsigned long [size](#)
size
- unsigned long [flags](#)
flags

11.183.1 Detailed Description

Information about the data space.

Definition at line 93 of file [dataspace](#).

The documentation for this struct was generated from the following file:

- l4/re/dataspace

11.184 L4::String Class Reference

A null-terminated string container class.

```
#include <string.h>
```

11.184.1 Detailed Description

A null-terminated string container class.

Definition at line 35 of file [string.h](#).

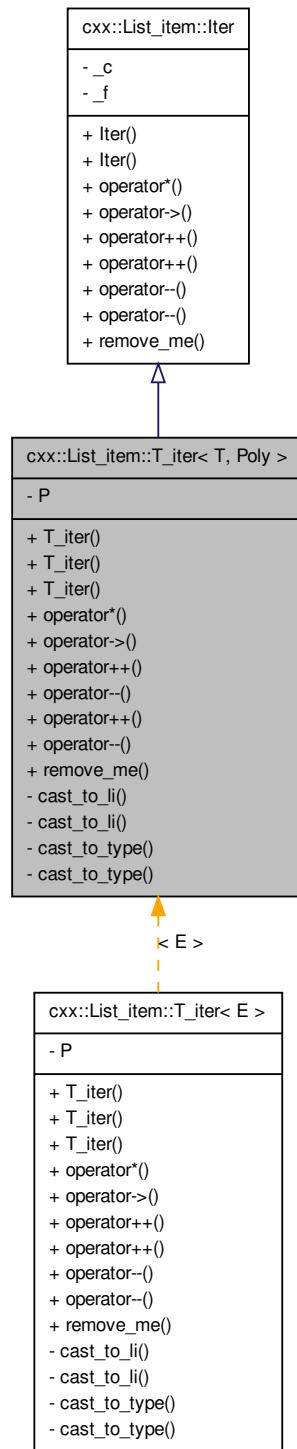
The documentation for this class was generated from the following file:

- l4/cxx/string.h

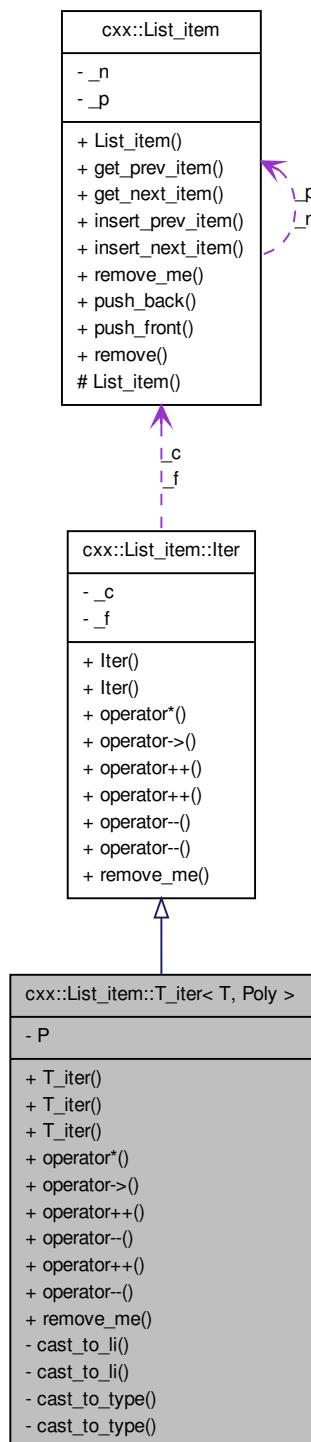
11.185 cxx::List_item::T_iter< T, Poly > Class Template Reference

Iterator for derived classes from ListItem.

Inheritance diagram for cxx::List_item::T_iter< T, Poly >:



Collaboration diagram for cxx::List_item::T_iter< T, Poly >:



Public Member Functions

- `T * remove_me () throw ()`

Remove item pointed to by iterator, and return pointer to element.

11.185.1 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 119 of file [list](#).

11.185.2 Member Function Documentation

11.185.2.1 `template<typename T, bool Poly> T * cxx::List_item::T_iter< T, Poly >::remove_me () throw () [inline]`

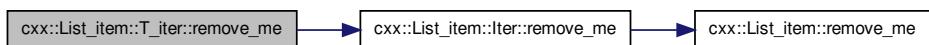
Remove item pointed to by iterator, and return pointer to element.

Reimplemented from `cxx::List_item::Iter`.

Definition at line 290 of file [list](#).

References `cxx::List_item::Iter::remove_me()`.

Here is the call graph for this function:



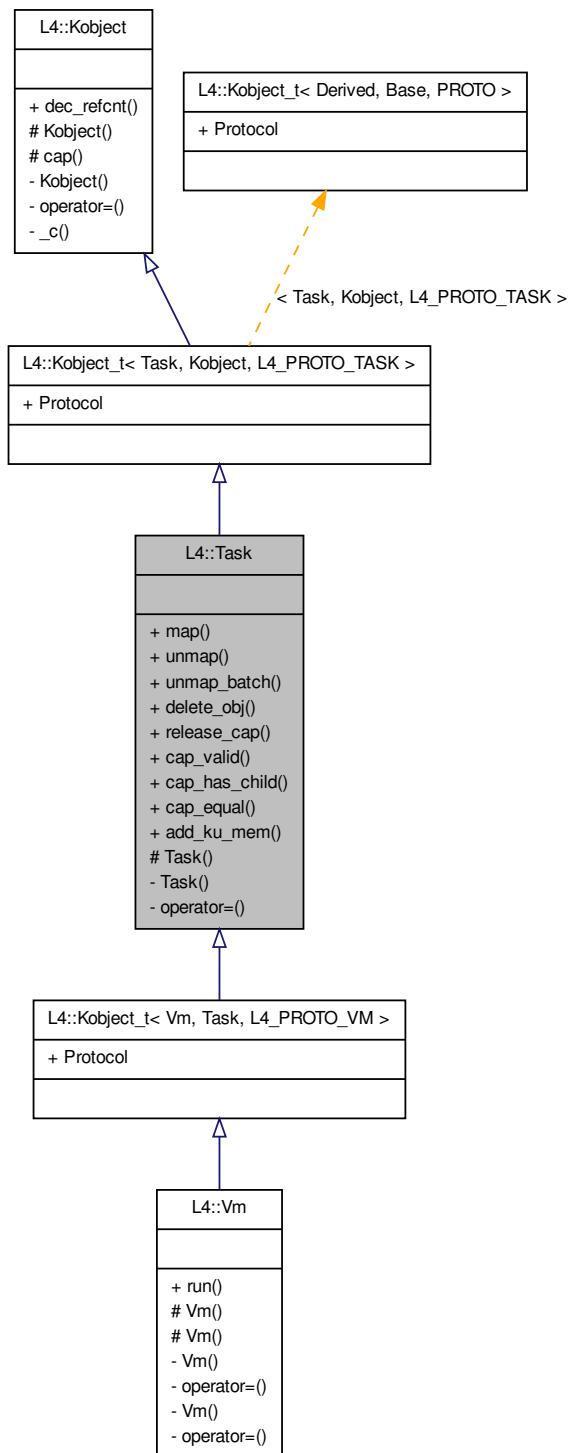
The documentation for this class was generated from the following file:

- [14/cxx/list](#)

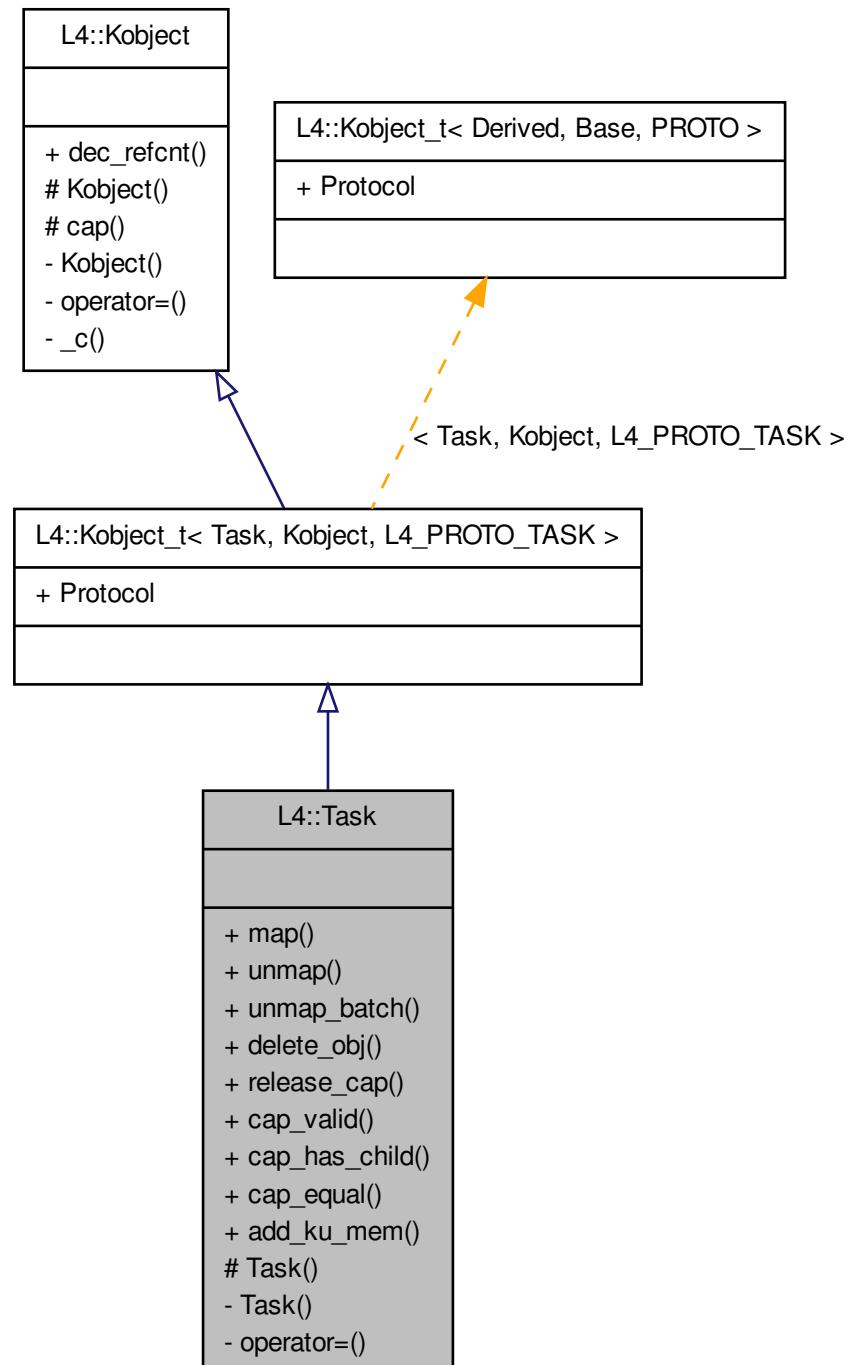
11.186 L4::Task Class Reference

An [L4 Task](#).

Inheritance diagram for L4::Task:



Collaboration diagram for L4::Task:



Public Member Functions

- `l4_mshtag_t map (Cap< Task > const &src_task, l4_fpage_t const &snd_fpage, l4_addr_t snd_base, l4_utcb_t *utcb=l4_utcb()) throw ()`
- `l4_mshtag_t unmap (l4_fpage_t const &fpage, l4_umword_t map_mask, l4_utcb_t *utcb=l4_utcb()) throw ()`
- `l4_mshtag_t unmap_batch (l4_fpage_t const *fpages, unsigned num_fpages, l4_umword_t map_mask, l4_utcb_t *utcb=l4_utcb()) throw ()`
- `l4_mshtag_t delete_obj (L4::Cap< void > obj, l4_utcb_t *utcb=l4_utcb()) throw ()`
- `l4_mshtag_t release_cap (L4::Cap< void > cap, l4_utcb_t *utcb=l4_utcb()) throw ()`
- `l4_mshtag_t cap_valid (Cap< void > const &cap, l4_utcb_t *utcb=l4_utcb()) throw ()`
- `l4_mshtag_t cap_has_child (Cap< void > const &cap, l4_utcb_t *utcb=l4_utcb()) throw ()`
- `l4_mshtag_t cap_equal (Cap< void > const &cap_a, Cap< void > const &cap_b, l4_utcb_t *utcb=l4_utcb()) throw ()`
- `l4_mshtag_t add_ku_mem (l4_fpage_t const &fpage, l4_utcb_t *utcb=l4_utcb()) throw ()`

11.186.1 Detailed Description

An L4 Task. #include <l4/sys/task>

See also

[Task](#) for an overview description.

Definition at line 41 of file [task](#).

11.186.2 Member Function Documentation

11.186.2.1 l4_mshtag_t L4::Task::map (Cap< Task > const & src_task, l4_fpage_t const & snd_fpage, l4_addr_t snd_base, l4_utcb_t * utcb = l4_utcb()) throw () [inline]

Map resources available in the source task to a destination task.

Parameters

`dst_task` Capability selector of destination task

`src_task` Capability selector of source task

`snd_fpage` Send flexpage that describes an area in the address space or object space of the source task

`snd_base` Send base that describes an offset in the receive window of the destination task.

Returns

Syscall return tag

This method allows for asynchronous rights delegation from one task to another. It can be used to share memory as well as to delegate access to objects.

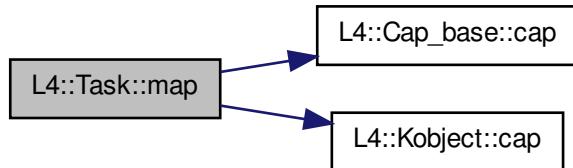
Note

`dst_task` is the implicit *this* pointer.

Definition at line 50 of file [task](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.186.2.2 `l4_mshtag_t L4::Task::unmap (l4_fpage_t const & fpage, l4_umword_t map_mask, l4_utcb_t * utcb = l4_utcb\(\)) throw () [inline]`

Revoke rights from the task.

Parameters

task Capability selector of destination task

fpage Flexpage that describes an area in the address space or object space of the destination task

map_mask Unmap mask, see [l4_unmap_flags_t](#)

Returns

Syscall return tag

This method allows to revoke rights from the destination task and from all the tasks that got the rights delegated from that task (i.e., this operation does a recursive rights revocation).

Note

Calling this function on the object space can cause a root capability of an object to be destructed, which destroys the object itself.

Note

task is the implicit *this* pointer.

Definition at line 59 of file [task](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.186.2.3 l4_mshtag_t L4::Task::unmap_batch (l4_fpage_t const * *fpages*, unsigned *num_fpages*, l4_umword_t *map_mask*, l4_utcb_t * *utcb* = *l4_utcb()*) throw () [inline]

Revoke rights from a task.

Parameters

task Capability selector of destination task

fpages An array of flexpages that describes an area in the address space or object space of the destination task each

num_fpages The size of the fpages array in elements (number of fpages sent).

map_mask Unmap mask, see [l4_unmap_flags_t](#)

Returns

Syscall return tag

This method allows to revoke rights from the destination task and from all the tasks that got the rights delegated from that task (i.e., this operation does a recursive rights revocation).

Precondition

The caller needs to take care that *num_fpages* is not bigger than *L4_UTCB_GENERIC_DATA_SIZE* - 2.

Note

Calling this function on the object space can cause a root capability of an object to be destructed, which destroys the object itself.

Note

task is the implicit *this* pointer.

Definition at line 68 of file [task](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



**11.186.2.4 `l4_mshtag_t L4::Task::delete_obj (L4::Cap< void > obj, l4_utcb_t * utcb =
 14_utcb()) throw () [inline]`**

Release capability and delete object.

Parameters

task Capability selector of destination task
obj Capability selector of object to delete

Returns

Syscall return tag

The object will be deleted if the obj has sufficient rights. No error will be reported if the rights are insufficient, however, the capability is removed in all cases.

This is operating calls [l4_task_unmap\(\)](#) with [L4_FP_DELETE_OBJ](#).

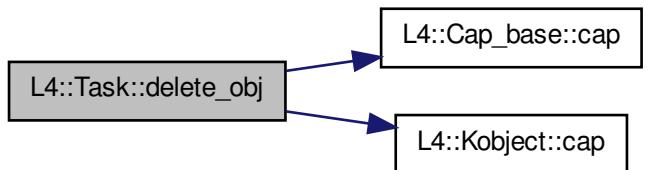
Note

task is the implicit *this* pointer.

Definition at line 78 of file [task](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.186.2.5 l4_mshtag_t L4::Task::release_cap (L4::Cap< void > *cap*, l4_utcb_t * *utcb* = *l4_utcb()*) throw () [inline]

Release capability.

Parameters

task Capability selector of destination task

cap Capability selector to release

Returns

Syscall return tag

This operation unmaps the capability from the specified task.

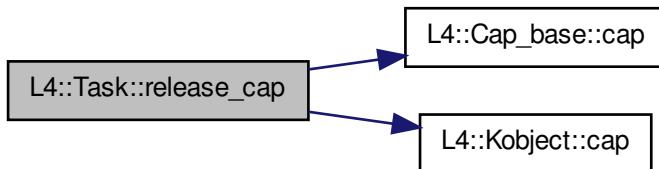
Note

task is the implicit *this* pointer.

Definition at line 86 of file [task](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.186.2.6 l4_mshtag_t L4::Task::cap_valid (Cap< void > const & *cap*, l4_utcb_t * *utcb* = *l4_utcb()*) throw () [inline]

Test whether a capability selector points to a valid capability.

Parameters

task Capability selector of the destination task to do the lookup in

cap Capability selector to look up in the destination task

Returns

label contains 1 if valid, 0 if invalid

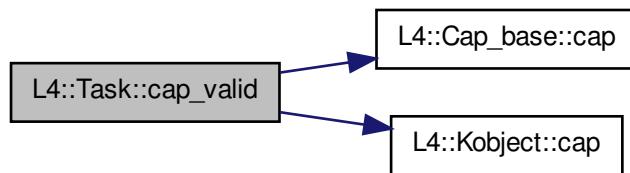
Note

task is the implicit *this* pointer.

Definition at line 94 of file [task](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.186.2.7 l4_mshtag_t L4::Task::cap_has_child (Cap< void > const & cap, l4_utcb_t * utcb = 14_utcb()) throw () [inline]

Test whether a capability has child mappings (in another task).

Parameters

task Capability selector of the destination task to do the lookup in

cap Capability selector to look up in the destination task

Returns

label contains 1 if it has at least one child, 0 if not or invalid

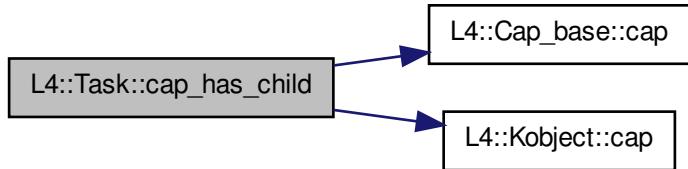
Note

task is the implicit *this* pointer.

Definition at line 102 of file [task](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.186.2.8 l4_mshtag_t L4::Task::cap_equal (Cap< void > const & cap_a, Cap< void > const & cap_b, l4_utcb_t * utcb = l4_utcb()) throw () [inline]

Test whether two capabilities point to the same object with the same rights.

Parameters

task Capability selector of the destination task to do the lookup in
cap_a Capability selector to compare
cap_b Capability selector to compare

Returns

label contains 1 if equal, 0 if not equal

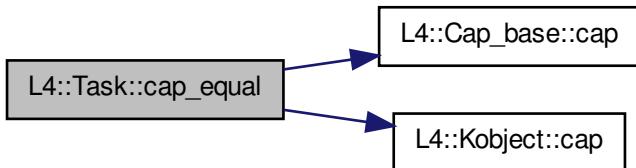
Note

task is the implicit *this* pointer.

Definition at line 110 of file [task](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.186.2.9 l4_mshtag_t L4::Task::add_ku_mem (l4_fpage_t const & *fpage*, l4_utcb_t * *utcb* = *l4_utcb()*) throw () [inline]

Add kernel-user memory.

Parameters

task Capability selector of the task to add the memory to

ku_mem Flexpage describing the virtual area the memory goes to.

Returns

Syscall return tag

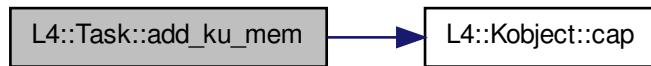
Note

task is the implicit *this* pointer.

Definition at line 119 of file [task](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



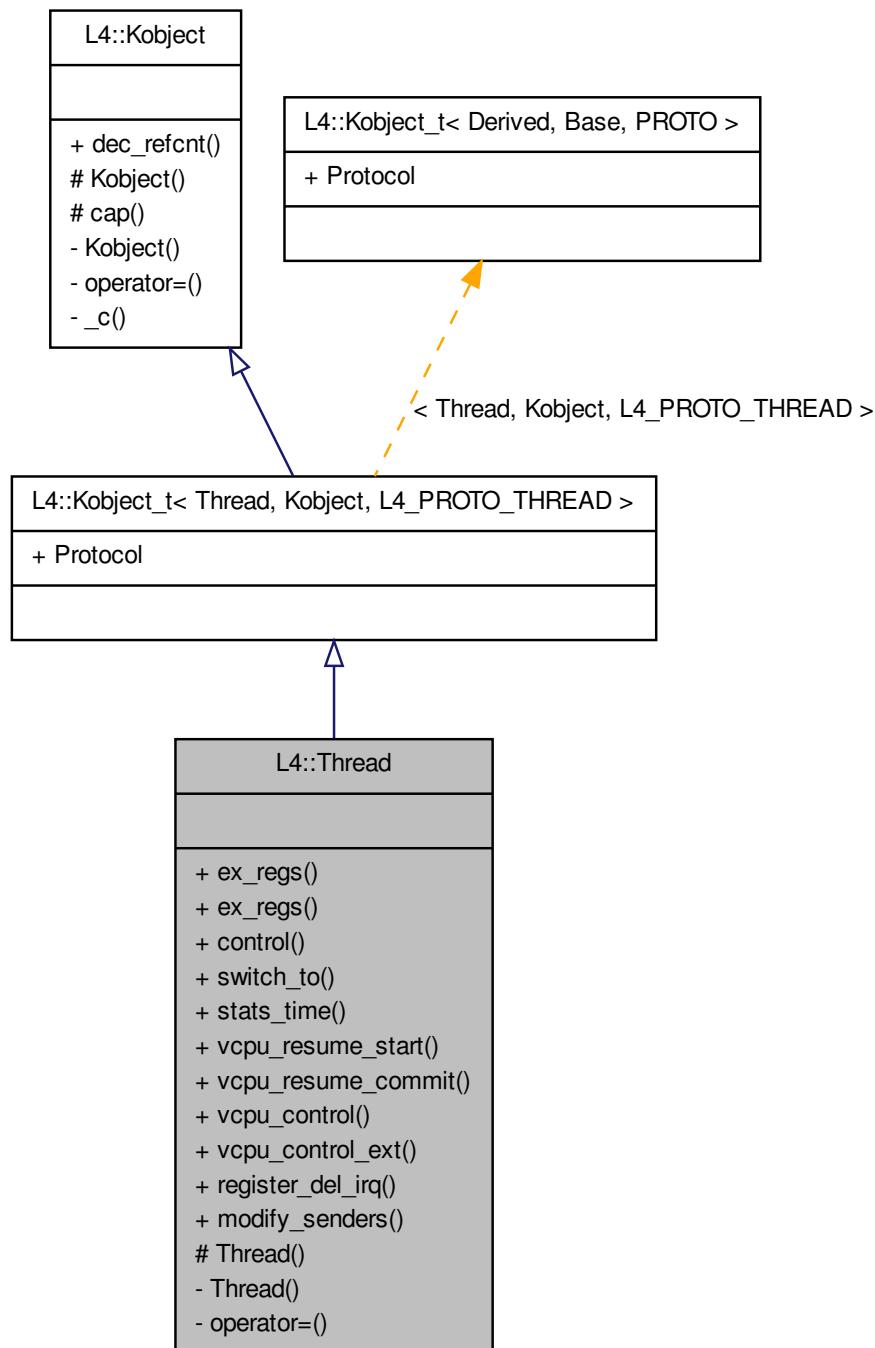
The documentation for this class was generated from the following file:

- [l4/sys/task](#)

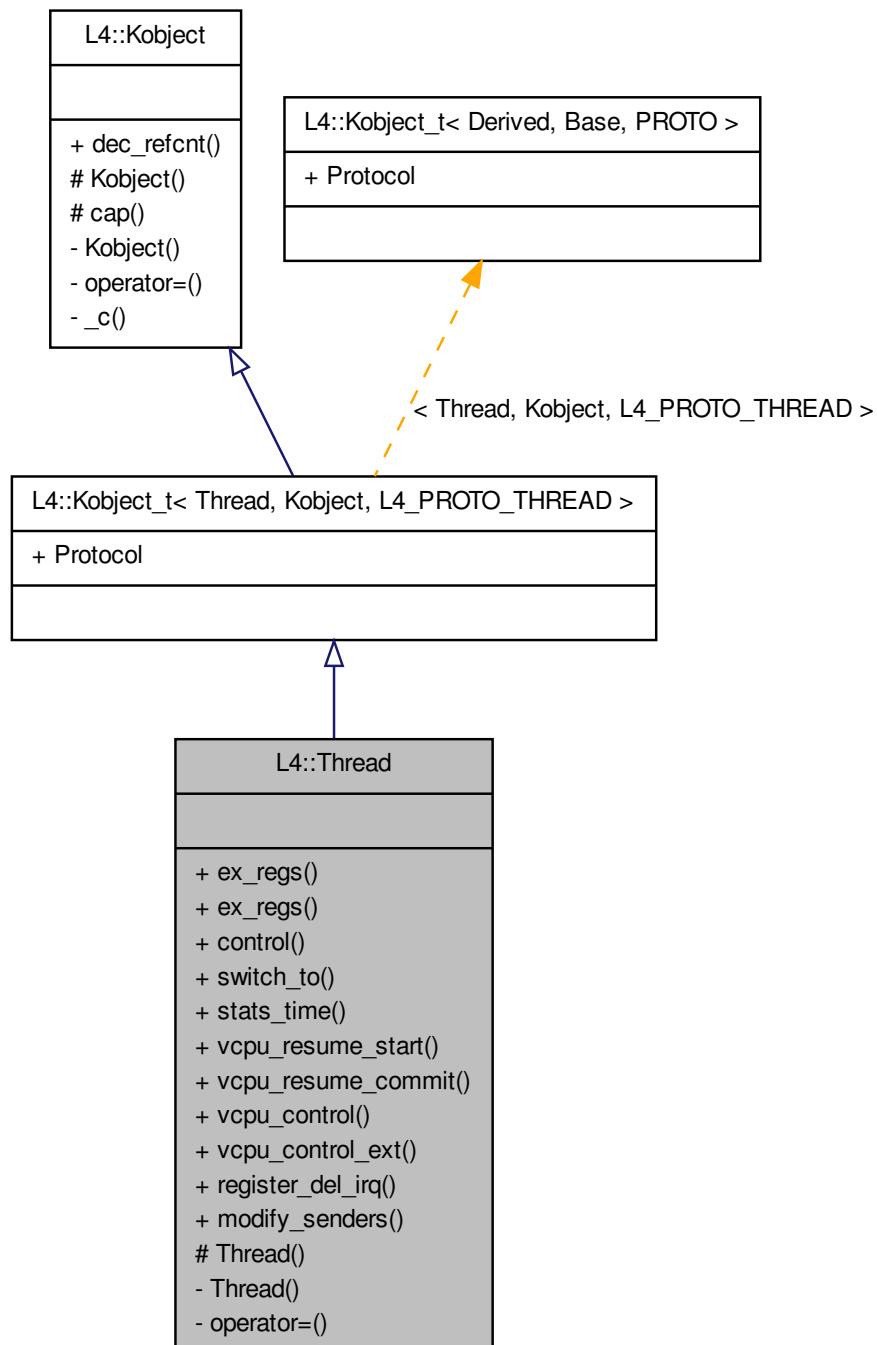
11.187 L4::Thread Class Reference

[L4](#) kernel thread.

Inheritance diagram for L4::Thread:



Collaboration diagram for L4::Thread:



Data Structures

- class [Attr](#)
Thread attributes used for control_commit().
- class [Modify_senders](#)
Wrapper class for modifying senders.

Public Member Functions

- [l4_mshtag_t ex_regs \(l4_addr_t ip, l4_addr_t sp, l4_umword_t flags, l4_utcb_t *utcb=l4_utcb\(\)\) throw \(\)](#)
*l4_mshtag_t ex_regs (l4_addr_t *ip, l4_addr_t *sp, l4_umword_t *flags, l4_utcb_t *utcb=l4_utcb()) throw ()*
- [l4_mshtag_t control \(Attr const &attr\) throw \(\)](#)
Commit the given thread-attributes object.
- [l4_mshtag_t switch_to \(l4_utcb_t *utcb=l4_utcb\(\)\) throw \(\)](#)
Switch execution to this thread.
- [l4_mshtag_t stats_time \(l4_utcb_t *utcb=l4_utcb\(\)\) throw \(\)](#)
Get consumed timed of thread in ns.
- [l4_mshtag_t vcpu_resume_start \(l4_utcb_t *utcb=l4_utcb\(\)\) throw \(\)](#)
vCPU resume, start.
- [l4_mshtag_t vcpu_resume_commit \(l4_mshtag_t tag, l4_utcb_t *utcb=l4_utcb\(\)\) throw \(\)](#)
vCPU resume, commit.
- [l4_mshtag_t vcpu_control \(l4_addr_t vcpu_state, l4_utcb_t *utcb=l4_utcb\(\)\) throw \(\)](#)
- [l4_mshtag_t vcpu_control_ext \(l4_addr_t ext_vcpu_state, l4_utcb_t *utcb=l4_utcb\(\)\) throw \(\)](#)
- [l4_mshtag_t register_del_irq \(Cap< Irq > irq, l4_utcb_t *u=l4_utcb\(\)\) throw \(\)](#)
Register an IRQ that will trigger upon deletion events.
- [l4_mshtag_t modify_senders \(Modify_senders const &todo\) throw \(\)](#)
Apply sender modifiction rules.

11.187.1 Detailed Description

[L4](#) kernel thread. #include <l4/sys/thread>

Definition at line 38 of file [thread](#).

11.187.2 Member Function Documentation

11.187.2.1 l4_mshtag_t L4::Thread::ex_regs (l4_addr_t ip, l4_addr_t sp, l4_umword_t flags, l4_utcb_t * utcb = l4_utcb()) throw () [inline]

Exchange basic thread registers.

Parameters

thread Thread to manipulate

ip New instruction pointer, use ~0UL to leave the instruction pointer unchanged

sp New stack pointer, use ~0UL to leave the stack pointer unchanged

flags Ex-reg 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*).

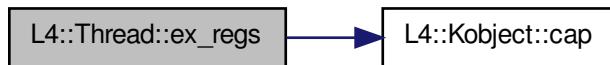
Note

the *thread* argument is the implicit *this* pointer.

Definition at line 47 of file [thread](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.187.2.2 l4_mshtag_t L4::Thread::ex_regs (l4_addr_t * ip, l4_addr_t * sp, l4_umword_t * flags, l4_utcb_t * utcb = l4_utcb()) throw () [inline]

Exchange basic thread registers and return previous values.

Parameters

[in] *thread* Thread to manipulate

[in, out] *ip* New instruction pointer, use ~0UL to leave the instruction pointer unchanged, return previous instruction pointer

[in, out] *sp* New stack pointer, use ~0UL to leave the stack pointer unchanged, returns previous stack pointer

[in, out] *flags* Ex-reg 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*).

Returned values are valid only if function returns successfully.

Note

the *thread* argument is the implicit *this* pointer.

Definition at line 56 of file [thread](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.187.2.3 `l4_mshtag_t L4::Thread::control(Attr const & attr) throw() [inline]`

Commit the given thread-attributes object.

Parameters

attr the attribute object to commit to the thread.

Definition at line 152 of file [thread](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.187.2.4 `l4_mshtag_t L4::Thread::switch_to(l4_utcb_t * utcb = l4_utcb()) throw() [inline]`

Switch execution to this thread.

Parameters

utcb the UTCB of the current thread.

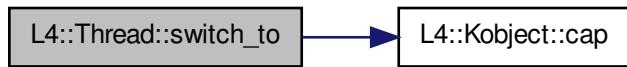
Note

The current time slice is inherited to this thread.

Definition at line 161 of file [thread](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.187.2.5 l4_mshtag_t L4::Thread::stats_time (l4_utcb_t * *utcb* = *l4_utcb()*) throw () [inline]

Get consumed timed of thread in ns.

Parameters

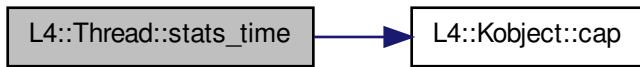
utcb the UTCB of the current thread.

The consumed time is return as l4_kernel_clock_t at UTCB message register 0.

Definition at line 171 of file [thread](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.187.2.6 l4_mshtag_t L4::Thread::vcpu_resume_start (l4_utcb_t * *utcb* = *l4_utcb()*) throw () [inline]

vCPU resume, start.

See also

[l4_thread_vcpu_resume_start](#)

Definition at line 179 of file [thread](#).

11.187.2.7 l4_mshtag_t L4::Thread::vcpu_resume_commit (l4_mshtag_t tag, l4_utcb_t * utcb = [l4_utcb\(\)](#))throw () [inline]

vCPU resume, commit.

See also

[l4_thread_vcpu_resume_commit](#)

Definition at line 187 of file [thread](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.187.2.8 l4_mshtag_t L4::Thread::vcpu_control (l4_addr_t vcpu_state, l4_utcb_t * utcb = [l4_utcb\(\)](#))throw () [inline]

Enable or disable the vCPU feature for the thread.

Parameters

thread The thread for which the vCPU feature shall be enabled or disabled.

vcpu_state 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.

Returns

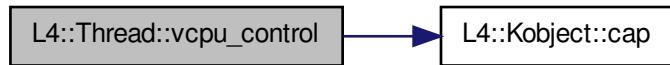
Systemcall result message tag.

This function enables the vCPU feature of the *thread* if *vcpu_state* is set to a valid kernel-user-memory address, or disables the vCPU feature if *vcpu_state* is 0.

Definition at line 194 of file [thread](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.187.2.9 `l4_mshtag_t L4::Thread::vcpu_control_ext (l4_addr_t ext_vcpu_state, l4_utcb_t * utcb = l4_utcb()) throw () [inline]`

Enable or disable the extended vCPU feature for the thread.

Parameters

thread The thread for which the extended vCPU feature shall be enabled or disabled.

vcpu_state 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.

Returns

Systemcall result message 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* if *vcpu_state* is set to a valid kernel-user-memory address, or disables the vCPU feature if *vcpu_state* is 0.

Definition at line 201 of file [thread](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.187.2.10 `l4_mshtag_t L4::Thread::register_del_irq (Cap< Irq > irq, l4_utcb_t * u = l4_utcb()) throw () [inline]`

Register an IRQ that will trigger upon deletion events.

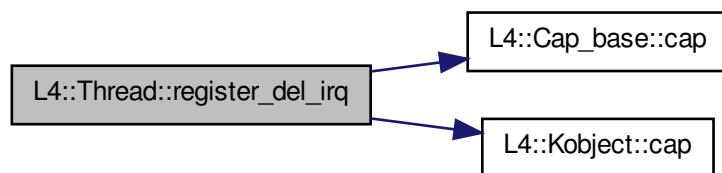
See also

[l4_thread_register_del_irq](#)

Definition at line 210 of file [thread](#).

References [L4::Cap_base::cap\(\)](#), and [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.187.2.11 **l4_mshtag_t L4::Thread::modify_senders (*Modify_senders const & todo*) throw () [inline]**

Apply sender modification rules.

Parameters

todo Prepared sender modification rules.

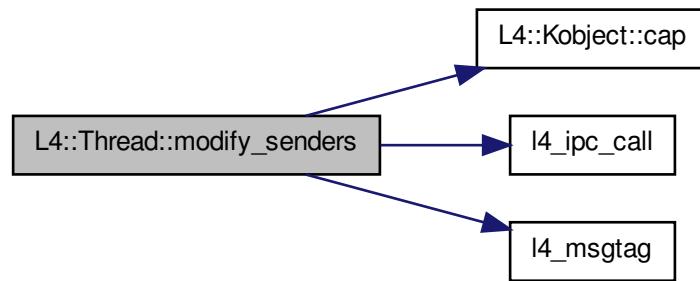
Returns

System call return tag.

Definition at line 270 of file [thread](#).

References [L4::Kobject::cap\(\)](#), [l4_ipc_call\(\)](#), [L4_IPC_NEVER](#), [l4_mshtag\(\)](#), and [L4_PROTO_THREAD](#).

Here is the call graph for this function:



The documentation for this class was generated from the following file:

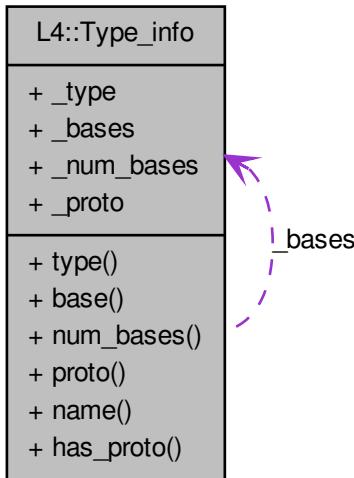
- l4/sys/thread

11.188 L4::Type_info Struct Reference

Dynamic Type Information for [L4Re](#) Interfaces.

```
#include <__typeinfo.h>
```

Collaboration diagram for L4::Type_info:



11.188.1 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 50 of file [__typeinfo.h](#).

The documentation for this struct was generated from the following file:

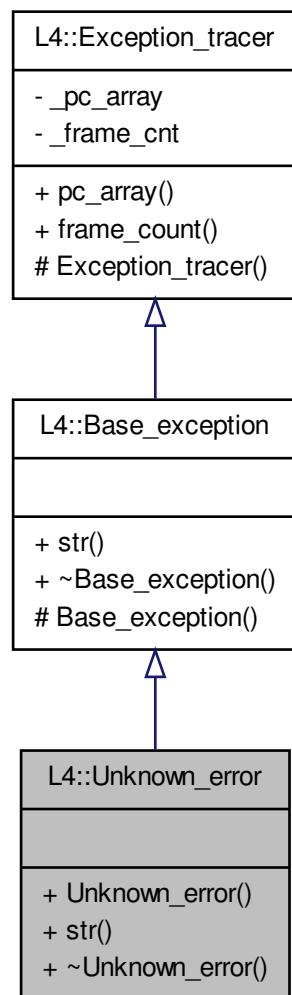
- [l4/sys/__typeinfo.h](#)

11.189 L4::Unknown_error Class Reference

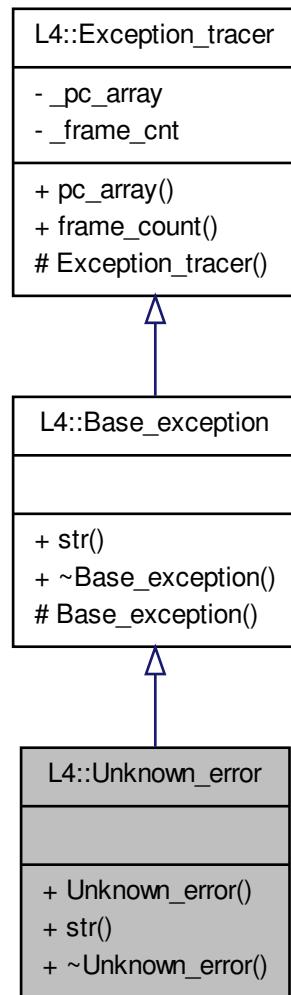
Exception for an unknown condition.

```
#include <l4/cxx/exceptions>
```

Inheritance diagram for L4::Unknown_error:



Collaboration diagram for L4::Unknown_error:



Public Member Functions

- `char const * str () const throw ()`
Should return a human readable string for the exception.

11.189.1 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 198 of file [exceptions](#).

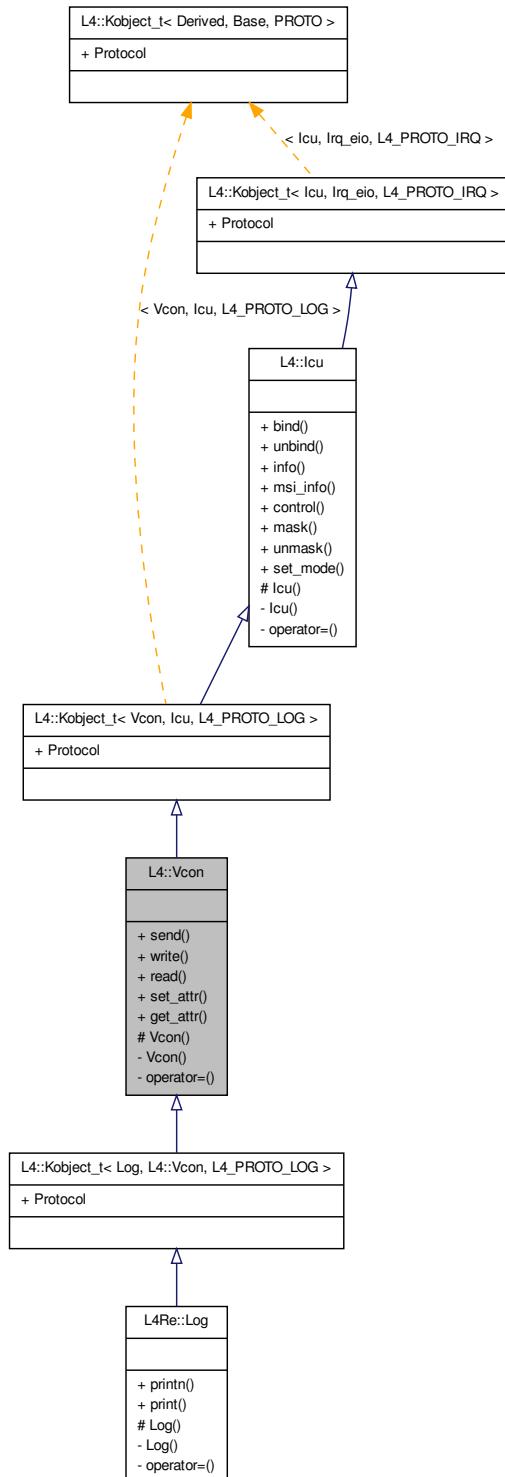
The documentation for this class was generated from the following file:

- [l4/cxx/exceptions](#)

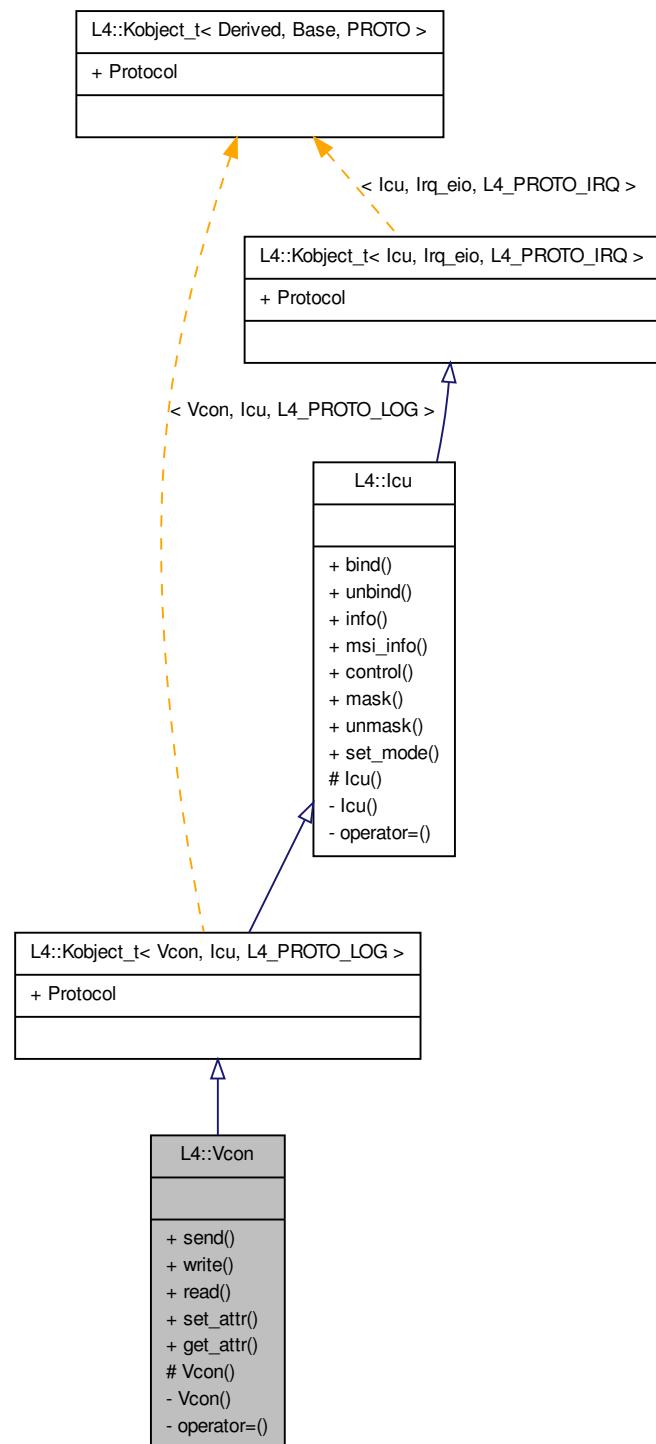
11.190 L4::Vcon Class Reference

C++ [L4](#) [Vcon](#).

Inheritance diagram for L4::Vcon:



Collaboration diagram for L4::Vcon:



Public Member Functions

- `l4_mshtag_t send (char const *buf, int size, l4_utcb_t *utcb=l4_utcb()) const throw ()`
- `long write (char const *buf, int size, l4_utcb_t *utcb=l4_utcb()) const throw ()`
- `int read (char *buf, int size, l4_utcb_t *utcb=l4_utcb()) const throw ()`
- `l4_mshtag_t set_attr (l4_vcon_attr_t const *attr, l4_utcb_t *utcb=l4_utcb()) const throw ()`
- `l4_mshtag_t get_attr (l4_vcon_attr_t *attr, l4_utcb_t *utcb=l4_utcb()) const throw ()`

11.190.1 Detailed Description

C++ [L4 Vcon](#). #include <l4/sys/vcon>

See also

[Virtual Console](#) for an overview and C bindings.

Definition at line 41 of file [vcon](#).

11.190.2 Member Function Documentation

11.190.2.1 l4_mshtag_t L4::Vcon::send (`char const * buf, int size, l4_utcb_t * utcb = l4_utcb()`) const throw () [inline]

Send data to virtual console.

Parameters

`vcon` Vcon object.

`buf` Pointer to data buffer.

`size` Size of buffer in bytes.

Returns

Syscall return tag

Note

Size must not exceed L4_VCON_WRITE_SIZE.

Note

`vcon` is the implicit *this* pointer.

Definition at line 52 of file [vcon](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.190.2.2 `long L4::Vcon::write (char const * buf, int size, l4_utcb_t * utcb = l4_utcb()) const throw () [inline]`

Write data to virtual console.

Parameters

vcon Vcon object.
buf Pointer to data buffer.
size Size of buffer in bytes.

Returns

Number of bytes written to the virtual console.

Note

vcon is the implicit *this* pointer.

Definition at line 60 of file [vcon](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.190.2.3 `int L4::Vcon::read (char * buf, int size, l4_utcb_t * utcb = l4_utcb()) const throw () [inline]`

Read data from virtual console.

Parameters

vcon Vcon object.
buf Pointer to data buffer.
size Size of buffer in bytes.

Returns

Negative error code on error, > size if more to read, size bytes are in the buffer, <= size bytes read

Note

vcon is the implicit *this* pointer.

Definition at line 68 of file [vcon](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.190.2.4 l4_mshtag_t L4::Vcon::set_attr (l4_vcon_attr_t const * attr, l4_utcb_t * utcb = 14_utcb()) const throw() [inline]

Set attributes of a Vcon.

Parameters

vcon Vcon object.
attr Attribute structure.

Returns

Syscall return tag

Note

vcon is the implicit *this* pointer.

Definition at line 76 of file [vcon](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



11.190.2.5 l4_mshtag_t L4::Vcon::get_attr (l4_vcon_attr_t * attr, l4_utcb_t * utcb = 14_utcb()) const throw () [inline]

Get attributes of a Vcon.

Parameters

vcon Vcon object.

Return values

attr Attribute structure.

Returns

Syscall return tag

Note

vcon is the implicit *this* pointer.

Definition at line 84 of file [vcon](#).

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/vcon](#)

11.191 L4Re::Util::Vcon_svr< SVR > Class Template Reference

Console server template class.

Public Member Functions

- int `dispatch (l4_umword_t obj, L4::Ipc::Iostream &ios)`

Server dispatch function.

11.191.1 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 return a value bigger than the given size to indicate that there's more data 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 function is given in bytes.

Definition at line 43 of file [vcon_svr](#).

11.191.2 Member Function Documentation

11.191.2.1 `template<typename SVR > int L4Re::Util::Vcon_svr< SVR >::dispatch (l4_umword_t obj, L4::Ipc::Iostream & ios)`

Server dispatch function.

Parameters

obj Server object ID to work on.

ios Input/Output stream.

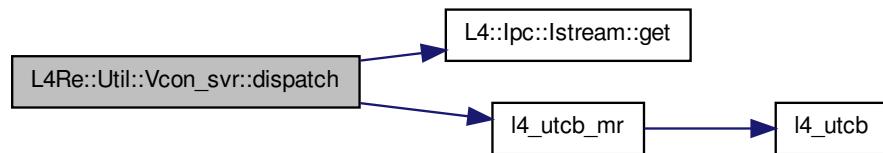
Returns

error code.

Definition at line 74 of file [vcon_svr](#).

References [L4::Ipc::Istream::get\(\)](#), [L4_EBADPROTO](#), [L4_EOK](#), [L4_UTCB_GENERIC_DATA_SIZE](#), [l4_utcb_mr\(\)](#), [L4_VCON_GET_ATTR_OP](#), [L4_VCON_SET_ATTR_OP](#), [L4_VCON_WRITE_OP](#), and [l4_msg_regs_t::mr](#).

Here is the call graph for this function:



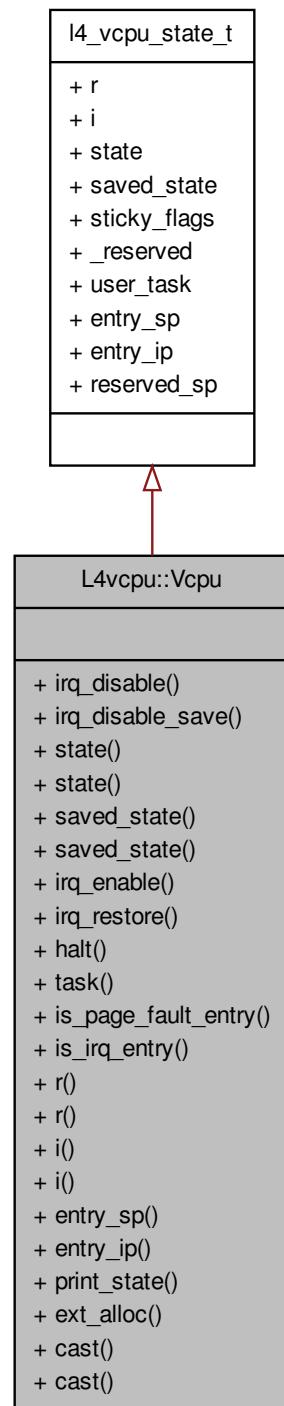
The documentation for this class was generated from the following file:

- `l4/re/util/vcon_svr`

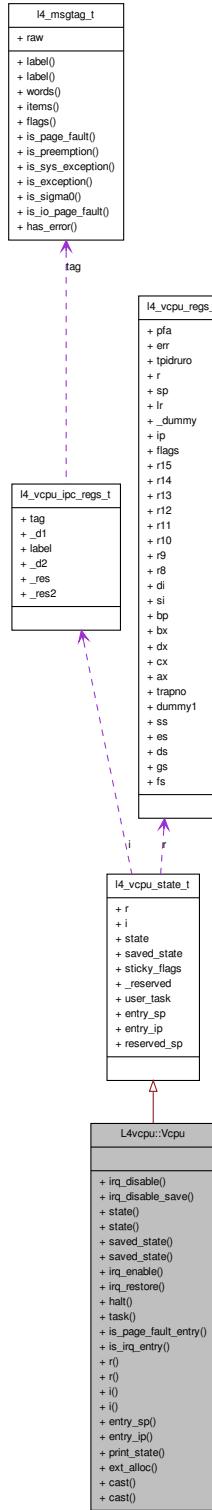
11.192 L4vcpu::Vcpu Class Reference

C++ implementation of the vCPU save state area.

Inheritance diagram for L4vcpu::Vcpu:



Collaboration diagram for L4vcpu::Vcpu:



Public Types

- **typedef l4vcpu_irq_state_t** **Irq_state**
IRQ status type.

Public Member Functions

- **void irq_disable ()** **throw ()**
Disable the vCPU for event delivery.
- **Irq_state 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.
- **void irq_enable (l4_utcb_t *utcb, l4vcpu_event_hdl_t do_event_work_cb, l4vcpu_setup_ipc_t setup_ipc)** **throw ()**
Enable the vCPU for event delivery.
- **void irq_restore (Irq_state s, l4_utcb_t *utcb, l4vcpu_event_hdl_t do_event_work_cb, l4vcpu_setup_ipc_t setup_ipc)** **throw ()**
Restore a previously saved IRQ/event state.
- **void halt (l4_utcb_t *utcb, l4vcpu_event_hdl_t do_event_work_cb, l4vcpu_setup_ipc_t setup_ipc)** **throw ()**
Halt/block the vCPU.
- **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 ()**
Return whether the entry reason was a page fault.
- **int is_irq_entry ()**
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="") 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.

11.192.1 Detailed Description

C++ implementation of the vCPU save state area.

Definition at line 71 of file `vcpu`.

11.192.2 Member Function Documentation

11.192.2.1 `Irq_state 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 89 of file `vcpu`.

11.192.2.2 State* L4vcpu::Vcpu::state () throw () [inline]

Get state word.

Returns

Pointer to state word in the vCPU

Definition at line 96 of file [vcpu](#).

11.192.2.3 State L4vcpu::Vcpu::state () const throw () [inline]

Get state word.

Returns

Pointer to state word in the vCPU

Definition at line 103 of file [vcpu](#).

11.192.2.4 State* L4vcpu::Vcpu::saved_state () throw () [inline]

Get saved_state word.

Returns

Pointer to saved_state word in the vCPU

Definition at line 110 of file [vcpu](#).

11.192.2.5 State L4vcpu::Vcpu::saved_state () const throw () [inline]

Get saved_state word.

Returns

Pointer to saved_state word in the vCPU

Definition at line 117 of file [vcpu](#).

11.192.2.6 void L4vcpu::Vcpu::irq_enable (l4_utcb_t * *utcb*, l4vcpu_event_hdl_t *do_event_work_cb*, l4vcpu_setup_ipc_t *setup_ipc*) throw () [inline]

Enable the vCPU for event delivery.

Parameters

utcb The UTCB to use.

do_event_work_cb Call-back function that is called in case an event (such as an interrupt) is pending.

setup_ipc Call-back function that is called before an IPC operation is called.

Definition at line 129 of file [vcpu](#).

11.192.2.7 void L4vcpu::Vcpu::irq_restore (*Irq_state* *s*, *l4_utcb_t* * *utcb*, *l4vcpu_event_hdl_t* *do_event_work_cb*, *l4vcpu_setup_ipc_t* *setup_ipc*) throw () [inline]

Restore a previously saved IRQ/event state.

Parameters

s IRQ state to be restored.

utcb The UTCB to use.

do_event_work_cb Call-back function that is called in case an event (such as an interrupt) is pending.

setup_ipc Call-back function that is called before an IPC operation is called.

Definition at line 143 of file [vcpu](#).

11.192.2.8 void L4vcpu::Vcpu::halt (*l4_utcb_t* * *utcb*, *l4vcpu_event_hdl_t* *do_event_work_cb*, *l4vcpu_setup_ipc_t* *setup_ipc*) throw () [inline]

Halt/block the vCPU.

Parameters

utcb The UTCB to use.

do_event_work_cb Call-back function that is called in case an event (such as an interrupt) is pending.

setup_ipc Call-back function that is called before an IPC operation is called.

Definition at line 157 of file [vcpu](#).

11.192.2.9 void L4vcpu::Vcpu::task (*L4::Cap< L4::Task >* const *task* = *L4::Cap<L4::Task>::Invalid*) throw () [inline]

Set the task of the vCPU.

Parameters

task Task to set, defaults to invalid task.

Definition at line 165 of file [vcpu](#).

11.192.2.10 int L4vcpu::Vcpu::is_page_fault_entry () [inline]

Return whether the entry reason was a page fault.

return 0 if not, !=0 otherwise.

Definition at line 172 of file [vcpu](#).

11.192.2.11 int L4vcpu::Vcpu::is_irq_entry () [inline]

Return whether the entry reason was an IRQ/IPC message.

return 0 if not, !=0 otherwise.

Definition at line 179 of file [vcpu](#).

11.192.2.12 l4_vcpu_regs_t* L4vcpu::Vcpu::r () throw () [inline]

Return pointer to register state.

Returns

Pointer to register state.

Definition at line 186 of file [vcpu](#).

11.192.2.13 l4_vcpu_regs_t const* L4vcpu::Vcpu::r () const throw () [inline]

Return pointer to register state.

Returns

Pointer to register state.

Definition at line 193 of file [vcpu](#).

11.192.2.14 l4_vcpu_ipc_regs_t* L4vcpu::Vcpu::i () throw () [inline]

Return pointer to IPC state.

Returns

Pointer to IPC state.

Definition at line 200 of file [vcpu](#).

11.192.2.15 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 207 of file [vcpu](#).

11.192.2.16 void L4vcpu::Vcpu::entry_sp (l4_umword_t sp) [inline]

Set vCPU entry stack pointer.

Parameters

sp Stack pointer address to set.

Note

The value is only used when entering from a user-task.

Definition at line 216 of file [vcpu](#).

11.192.2.17 void L4vcpu::Vcpu::entry_ip(l4_umword_t ip) [inline]

Set vCPU entry instruction pointer.

Parameters

ip Instruction pointer address to set.

Definition at line 223 of file [vcpu](#).

11.192.2.18 static 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.

Return values

vcpu Allocated vcpu-state area.

ext_state Allocated extended vcpu-state area.

Parameters

task Task to use for allocation, defaults to own task.

rm Region manager to use for allocation defaults to standard region manager.

Returns

0 for success, error code otherwise

11.192.2.19 static Vcpu* L4vcpu::Vcpu::cast(void * x) throw() [inline, static]

Cast a void pointer to a class pointer.

Parameters

x Pointer.

Returns

Pointer to [Vcpu](#) class.

Definition at line 249 of file [vcpu](#).

11.192.2.20 static Vcpu* L4vcpu::Vcpu::cast(l4_addr_t x) throw() [inline, static]

Cast an address to a class pointer.

Parameters

x Pointer.

Returns

Pointer to [Vcpu](#) class.

Definition at line [259](#) of file [vcpu](#).

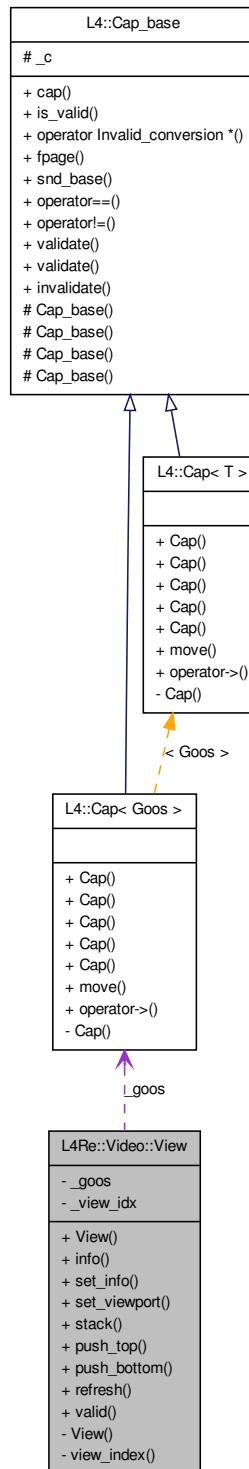
The documentation for this class was generated from the following file:

- [l4/vcpu/vcpu](#)

11.193 L4Re::Video::View Class Reference

[View](#).

Collaboration diagram for L4Re::Video::View:



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 throw ()

Return the view information of the view.
- int [set_info](#) ([Info](#) const &info) const throw ()

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 throw ()

Set the position of the view in the goos.
- int [stack](#) ([View](#) const &pivot, bool behind=true) const throw ()

Move this view in the view stack.
- int [push_top](#) () const throw ()

Make this view the top-most view.
- int [push_bottom](#) () const throw ()

Push this view the back.
- int [refresh](#) (int x, int y, int w, int h) const throw ()

Refresh/Redraw the view.
- bool [valid](#) () const

Return whether this view is valid.

Friends

- class [Goos](#)

ID for goos objects.

11.193.1 Detailed Description

[View](#).

Definition at line 34 of file [view](#).

11.193.2 Member Enumeration Documentation

11.193.2.1 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.

F_set_flags Set view flags (.

See also

[V_flags](#))

F_fully_dynamic Flags for a fully dynamic view.

Definition at line 53 of file [view](#).

11.193.2.2 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:

F_above Flag the view as stay on top.

F_flags_mask Mask containing all possible property flags.

Definition at line 76 of file [view](#).

11.193.3 Member Function Documentation

11.193.3.1 int L4Re::Video::View::info (Info * *info*) const throw()

Return the view information of the view.

Return values

info Information structure pointer.

Returns

0 on success, error otherwise

11.193.3.2 int L4Re::Video::View::set_info (Info const & *info*) const throw ()

Set the information structure for this view.

Parameters

info Information structure.

Returns

0 on success, error otherwise

The function will also set the view port according to the values given in the information structure.

11.193.3.3 int L4Re::Video::View::set_viewport (int *scr_x*, int *scr_y*, int *w*, int *h*, unsigned long *buf_offset*) const throw ()

Set the position of the view in the goos.

Parameters

scr_x X position

scr_y Y position

w Width

h Height

buf_offset Offset in the buffer in bytes

Returns

0 on success, error otherwise

11.193.3.4 int L4Re::Video::View::stack (View const & *pivot*, bool *behind = true*) const throw ()

Move this view in the view stack.

Parameters

pivot [View](#) to move relative to

behind When true move the view behind the pivot view, if false move the view before the pivot view.

Returns

0 on success, error otherwise

11.193.3.5 int L4Re::Video::View::refresh (int *x*, int *y*, int *w*, int *h*) const throw ()

Refresh/Redraw the view.

Parameters

x X position.

y Y position.

w Width.

h Height.

Returns

0 on success, error otherwise

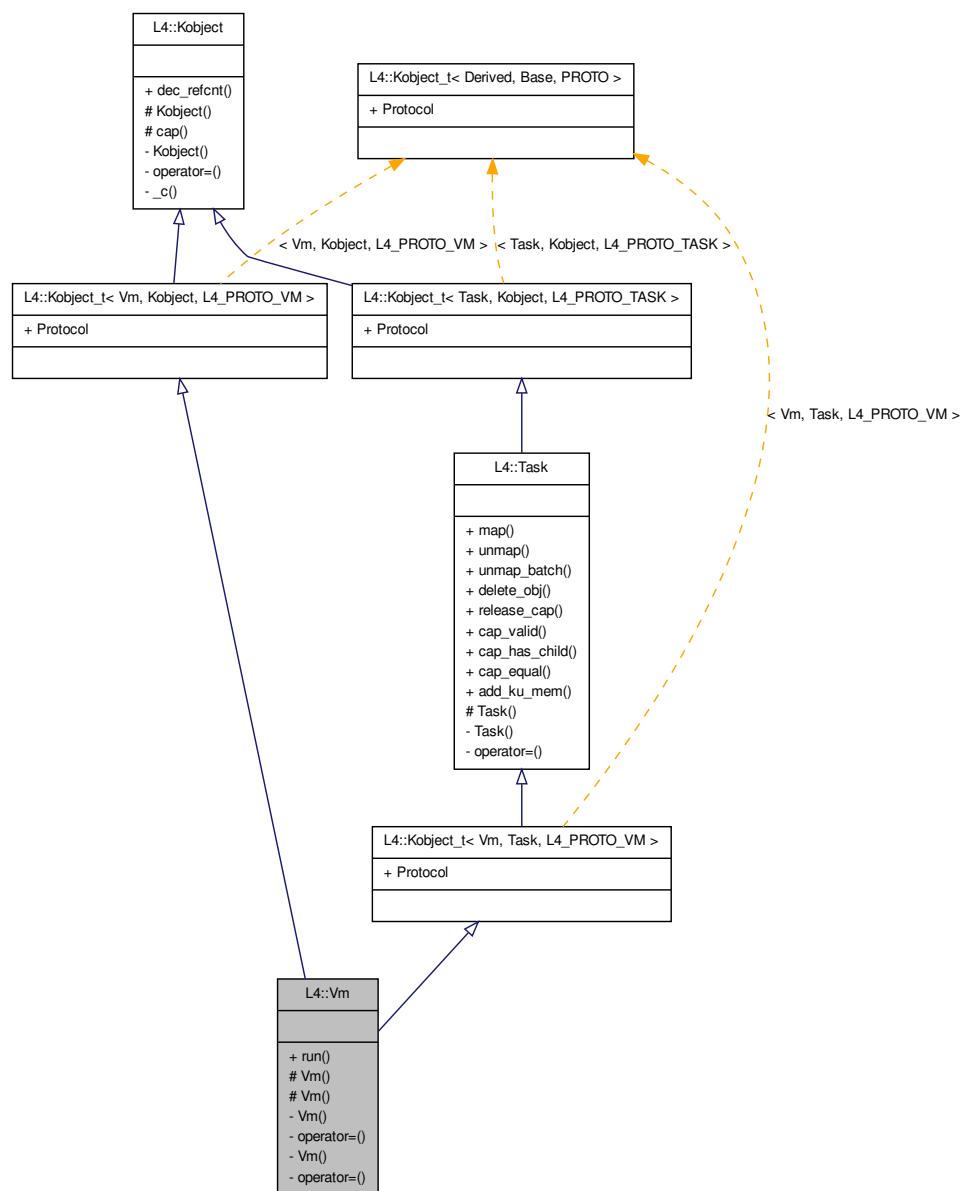
The documentation for this class was generated from the following file:

- l4/re/video/view

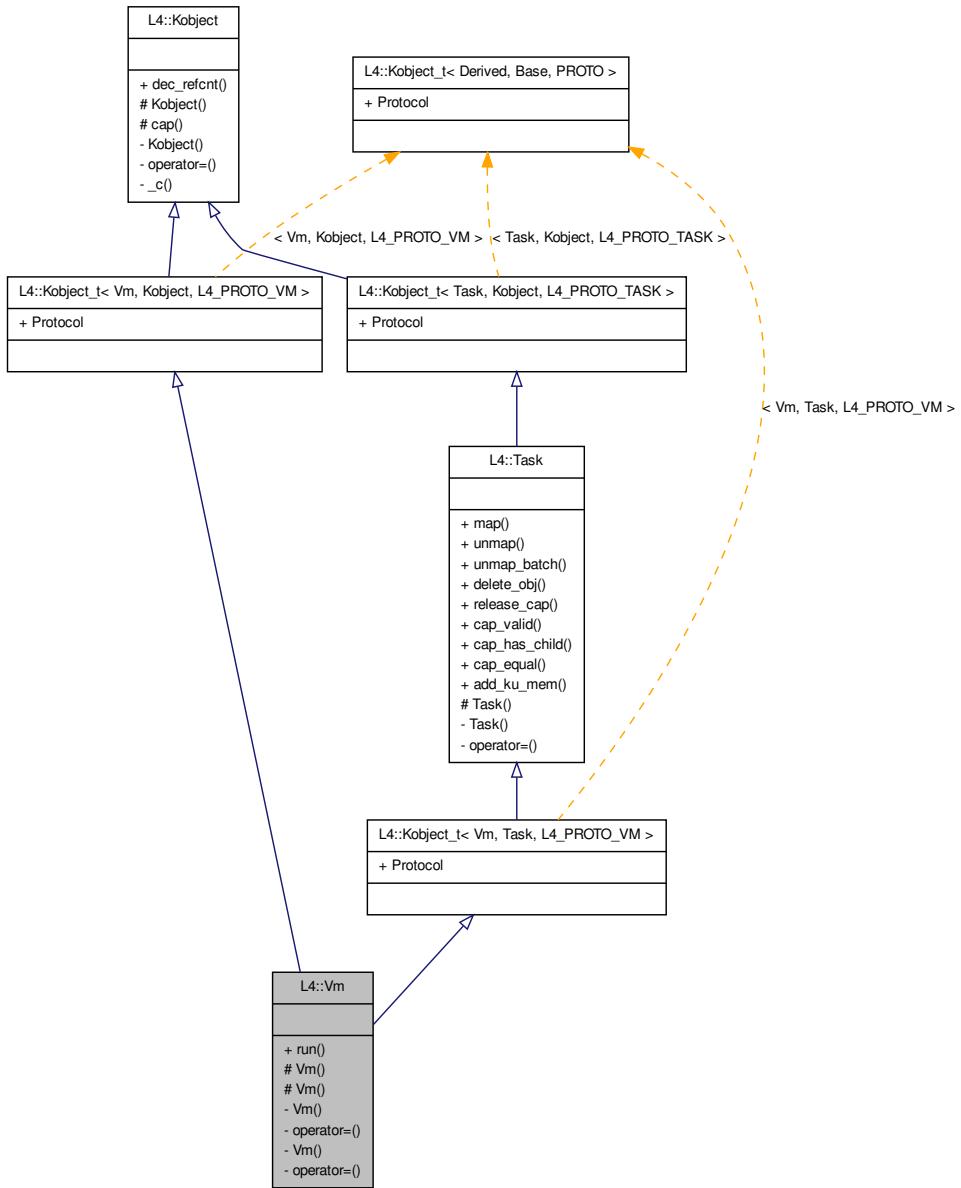
11.194 L4::Vm Class Reference

Virtual machine.

Inheritance diagram for L4::Vm:



Collaboration diagram for L4::Vm:



Public Member Functions

- `l4_mshtag_t run (l4_fpage_t const &fpage, l4_umword_t *label, l4_utcb_t *utcb=l4_utcb()) throw ()`

11.194.1 Detailed Description

Virtual machine. TZ Virtual machine.

Definition at line 36 of file [__vm](#).

11.194.2 Member Function Documentation

11.194.2.1 `l4_mshtag_t L4::Vm::run (l4_fpage_t const & fpage, l4_umword_t * label, l4_utcb_t * utcb = l4_utcb()) throw () [inline]`

Run a VM.

Parameters

`vm` Capability selector for VM

Note

`dst_task` is the implicit *this* pointer.

Definition at line 51 of file [vm](#).

References [L4::Kobject::cap\(\)](#).

Here is the call graph for this function:



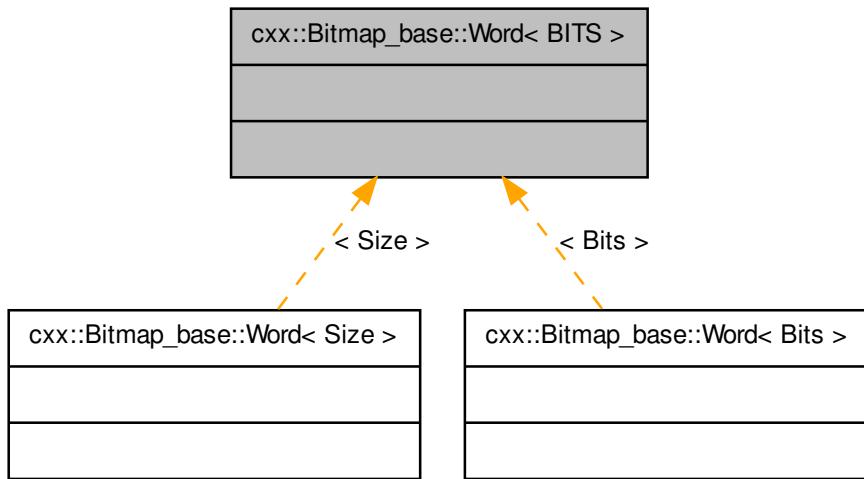
The documentation for this class was generated from the following files:

- [l4/sys/__vm](#)
- [arm/l4/sys/vm](#)

11.195 `cxx::Bitmap_base::Word< BITS > Class Template Reference`

Helper abstraction for a word contained in the bitmap.

Inheritance diagram for cxx::Bitmap_base::Word< BITS >:



11.195.1 Detailed Description

`template<long BITS> class cxx::Bitmap_base::Word< BITS >`

Helper abstraction for a word contained in the bitmap.

Definition at line 50 of file [bitmap](#).

The documentation for this class was generated from the following file:

- [l4/cxx\(bitmap](#)

Chapter 12

Example Documentation

12.1 examples/clntsrv/client.cc

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 Universitt 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 <14/sys/err.h>
#include <14/sys/types.h>
#include <14/re/env>
#include <14/re/util/cap_alloc>
#include <14/cxx/ipc_stream>

#include <stdio.h>

#include "shared.h"

static int
func_neg_call(L4::Cap<void> const &server, 14_uint32_t *result,
              14_uint32_t val)
{
    L4::Ipc::Iostream s(14_utcb());
    s << 14_umword_t(Opcode::func_neg) << val;
    int r = 14_error(s.call(server.cap()), Protocol::Calc));
    if (r)
        return r; // failure
    s >> *result;
    return 0; // ok
}

static int
func_sub_call(L4::Cap<void> const &server, 14_uint32_t *result,
              14_uint32_t val1, 14_uint32_t val2)
{
    L4::Ipc::Iostream s(14_utcb());
    s << 14_umword_t(Opcode::func_sub) << val1 << val2;
    int r = 14_error(s.call(server.cap()), Protocol::Calc));
    if (r)
        return r; // failure
```

```

    s >> *result;
    return 0; // ok
}

int
main()
{
    L4::Cap<void> server = L4Re::Env::env()->get_cap<void>("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 (func_sub_call(server, &val1, val1, val2))
    {
        printf("Error talking to server\n");
        return 1;
    }
    printf("Result of subtract call: %d\n", val1);
    printf("Asking for -%d\n", val1);
    if (func_neg_call(server, &val1, val1))
    {
        printf("Error talking to server\n");
        return 1;
    }
    printf("Result of negate call: %d\n", val1);

    return 0;
}

```

12.2 examples/clntsrv/clntsrv.cfg

Sample configuration file for the client/server example.

```

-- vim:set ft=lua:

-- Include L4 functionality
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");

```

12.3 examples/clntsrv/server.cc

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 Universitt 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 L4Re::Util::Registry_server<> server;

class Calculation_server : public L4::Server_object
{
public:
    int dispatch(l4_umword_t obj, L4::Ipc::Iostream &ios);
};

int
Calculation_server::dispatch(l4_umword_t, L4::Ipc::Iostream &ios)
{
    l4_mshtag_t t;
    ios >> t;

    // We're only talking the calculation protocol
    if (t.label() != Protocol::Calc)
        return -L4_EBADPROTO;

    L4::Opcode opcode;
    ios >> opcode;

    switch (opcode)
    {
        case Opcode::func_neg:
            l4_uint32_t val;
            ios >> val;
            val = -val;
            ios << val;
            return L4_EOK;
        case Opcode::func_sub:
            l4_uint32_t val1, val2;
            ios >> val1 >> val2;
            val1 -= val2;
            ios << val1;
            return L4_EOK;
        default:
            return -L4_ENOSYS;
    }
}

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;
}

```

12.4 examples/libs/l4re/c++/mem_alloc/ma+rm.cc

Coarse grained memory allocation, in C++.

```

/*
 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
 *      economic rights: Technische Universitt 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::Search_addr, 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;

    /* Free memory at our memory allocator, this is optional */
    if ((r = L4Re::Env::env()>mem_alloc()>free(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;
}

```

12.5 examples/libs/l4re/c++/shared_ds/ds_clnt.cc

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 Universitt 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 <14/re/util/cap_alloc> // L4::Cap
#include <14/re/dataspace> // L4Re::Dataspace
#include <14/re/rm> // L4::Rm
#include <14/re/env> // L4::Env
#include <14/sys/cache.h>

#include <cstring>
#include <cstdio>
#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::Search_addr, 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
sleep(3);

/*
 * Fill in new stuff
 */
memset(addr, 0, ds->size());
char const * 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");

L4Re::Util::cap_alloc.free(ds, L4Re::This_task);

return 0;
}

```

12.6 examples/libs/l4re/c++/shared_ds/ds_srv.cc

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 Universitt 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/namespace>
#include <l4/re/util/cap_alloc>
#include <l4/re/util/object_registry>
#include <l4/re/dataspace>
#include <l4/cxx/ipc_server>

#include <l4/sys/typeinfo_svr>

#include <cstring>
#include <cstdio>
#include <unistd.h>

#include "interface.h"

class My_server_obj : public L4::Server_object
{
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_mshtag_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::Server_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;

    l4_mshtag_t t;
    ios >> t; // extract the tag

    switch (t.label())
    {
        case L4::Irq::Protocol:
            // Got an IRQ so just print the new contents of the
            // shared memory.
            printf("Content: %s\n", _shm);
            return 0;
        default:
    }
}

```

```

        // every other protocol is not supported.
        return -L4_EBADPROTO;
    }

static L4Re::Util::Registry_server<> server;

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::Search_addr,
                                             *_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;
}

int main()
{
    L4::Cap<L4Re::Dataspace> ds;
    char *addr;

    if (!(addr = get_ds(&ds)))
        return 2;

    // 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();
return 0;
}

```

12.7 examples/libs/l4re/c++/shared_ds/shared_ds.lua

Sharing memory between applications, configuration file.

```

-- Include L4 functionality
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"
);

```

12.8 examples/libs/l4re/c/ma+rm.c

Coarse grained memory allocation, in C.

```

/*
 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
 *      economic rights: Technische Universitt 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/c/mem_alloc.h>
#include <l4/re/c/rm.h>
#include <l4/re/c/util/cap_alloc.h>
#include <l4/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_SEARCH_ADDR, ds, 0,
                           flags & L4RE_MA_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;
    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 */
    if ((r = l4re_ma_free(ds)))
        return r;

    l4re_util_cap_free(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;
}

```

12.9 examples/libs/l4re/streammap/client.cc

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 Universitt 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::Search_addr)))
    {
        printf("The reservation of one page within our virtual memory failed with %d\n", err);
        return 1;
    }

    s << l4_umword_t(Opcode::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(), Protocol::Map_example));
    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;
}

```

12.10 examples/libs/l4re/streammap/server.cc

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 Universitt 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
{
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() != Protocol::Map_example)
        return -L4_EBADPROTO;

    L4::Opcode opcode;
    ios >> opcode;

    switch (opcode)
    {
        case Opcode::Do_map:
            l4_addr_t snd_base;
            ios >> snd_base;
            // put something into the page to read it out at the other side
            sprintf(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;
}

```

12.11 examples/libs/l4re/streammap/streammap.cfg

Sample configuration file for the client/server map example.

```

-- vim:set ft=lua:

-- Include L4 functionality
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");

```

12.12 examples/libs/libirq/async_isr.c

libirq usage example using asynchronous ISR handler functionality.

```

/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
 *      economic rights: Technische Universitt 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;
}

```

```
}
```

12.13 examples/libs/libirq/loop.c

libirq usage example using a self-created thread.

```
/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
 *      economic rights: Technische Universitt 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;
}
```

12.14 examples/libs/shmc/prodcons.c

Simple shared memory example.

```

/*
 *  (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
 *      economic rights: Technische Universitt 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>

// a small helper
#define CHK(func) if (func) { printf("failure: %d\n", __LINE__); return (void *)-1; }

static const char some_data[] = "Hi consumer!";

static void *thread_producer(void *d)
{
    (void)d;
    l4shmc_chunk_t p_one;
    l4shmc_signal_t s_one, s_done;
    l4shmc_area_t shmarea;

    // 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, "prod", &s_one));

    CHK(l4shmc_attach_signal_to(&shmarea, "done",
                                pthread_getl4cap(pthread_self()), 10000, &s_done));

    // connect chunk and signal
    CHK(l4shmc_connect_chunk_signal(&p_one, &s_one));

    printf("PRODUCER: 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)));

        printf("PRODUCER: Sent data\n");

        CHK(l4shmc_wait_signal(&s_done));
    }
}

```

```

    }

    l4_sleep_forever();
    return NULL;
}

static void *thread_consume(void *d)
{
    (void)d;
    l4shmc_area_t shmarea;
    l4shmc_chunk_t p_one;
    l4shmc_signal_t s_one, s_done;

    // 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, "done", &s_done));

    // attach signal to this thread
    CHK(l4shmc_attach_signal_to(&shmarea, "prod",
                                pthread_getl4cap(pthread_self()), 10000, &s_one));

    // connect chunk and signal
    CHK(l4shmc_connect_chunk_signal(&p_one, &s_one));

    while (1)
    {
        CHK(l4shmc_wait_chunk(&p_one));

        printf("CONSUMER: 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;

    // create new shared memory area, 8K in size
    if (l4shmc_create("testshm", 8192))
        return 1;

    // create two threads, one for producer, one for consumer
    pthread_create(&one, 0, thread_producer, 0);
    pthread_create(&two, 0, thread_consume, 0);

    // now sleep, the two threads are doing the work
    l4_sleep_forever();

    return 0;
}

```

12.15 examples/sys/aliens/main.c

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>,
 *                  BjÄurn DÄubel <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.
 */
//#define MEASURE

#include <14/sys/ ipc.h>
#include <14/sys/ thread.h>
#include <14/sys/ factory.h>
#include <14/sys/ utcb.h>
#include <14/sys/ kdebug.h>
#include <14/util/ util.h>
#include <14/util/ rdtsc.h>
#include <14/re/ env.h>
#include <14/re/c/ util/ cap_alloc.h>
#include <14/sys/ debugger.h>

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

static char alien_thread_stack[8 << 10];
static 14_cap_idx_t alien;

static void alien_thread(void)
{
    volatile 14_mshtag_t x;
    while (1) {
        x = 14_ipc_call(0x1234 << L4_CAP_SHIFT, 14_utcb(), 14_mshtag(0, 0, 0, 0),
                        L4_IPC_NEVER);
    #ifdef MEASURE
        14_sleep(0);
    #else
        14_sleep(1000);
        outstring("An int3 -- you should see this\n");
        outnstring("345", 3);
    #endif
    }
}

int main(void)
{
    14_mshtag_t tag;
    #ifdef MEASURE
        14_cpu_time_t s, e;
    #endif
    14_utcb_t *u = 14_utcb();
    14_exc_regs_t exc;
    14_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_mshtag_has_error(tag))
    return 1;

l4_debugger_set_object_name(alien, "alienth");

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(alien);
if (l4_mshtag_has_error(tag))
    return 2;

tag = l4_thread_ex_regs(alien,
    (l4_umword_t)alien_thread,
    (l4_umword_t)alien_thread_stack + sizeof(alien_thread_s
    tack),
    0);
if (l4_mshtag_has_error(tag))
    return 3;

#ifndef MEASURE
    l4_calibrate_tsc(l4re_kip());
#endif

/* Pager/Exception loop */
if (l4_mshtag_has_error(tag = l4_ipc_receive(alien, u, L4_IPC_NEVER)))
{
    printf("l4_ipc_receive failed");
    return 1;
}

memcpy(&exc, l4_utcb_exc(), sizeof(exc));
mr0 = l4_utcb_mr()->mr[0];
mr1 = l4_utcb_mr()->mr[1];

for (i++)
{
#ifndef MEASURE
    s = l4_rdtsc();
#endif

    if (l4_mshtag_is_exception(tag))
    {
#ifndef MEASURE
        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, (exc.err & 4) ? " after" : "before",
        exc.err >> 3);
#endif
    }
    tag = l4_mshtag((exc.err & 4) ? 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(14_utcb_exc(), &exc, sizeof(exc));

/* Reply and wait */
if (14_mshtag_has_error(tag = 14_ipc_call(alien, u, tag, L4_IPC_NEVER)))
{
    printf("14_ipc_call failed\n");
    return 1;
}
memcpy(&exc, 14_utcb_exc(), sizeof(exc));
mr0 = 14_utcb_mr()->mr[0];
mr1 = 14_utcb_mr()->mr[1];
#endif
}

return 0;
}

```

12.16 examples/sys/ ipc/ ipc.cfg

Sample configuration file for the IPC example.

```

# vim:se ft=lua:

require("L4");

L4.default_loader:start({}, "rom/ex_ipc1");

```

12.17 examples/sys/ ipc/ ipc_example.c

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 Universitt 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 <14/sys/ ipc.h>

#include <pthread-14.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)
{
    14_mshtag_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. MRO). No timeout. */
    tag = l4_ipc_call(pthread_getl4cap(t2), l4_utcb(),
                      l4_mshtag(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_mshtag_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. MRO) */
        tag = l4_ipc_reply_and_wait(l4_utcb(), l4_mshtag(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;
}

```

12.18 examples/sys/isr/main.c

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>,
 *      BjÃrn DÃbel <doebel@os.inf.tu-dresden.de>
 *      economic rights: Technische UniversitÃ¤t 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. The interrupt number of the virtual key interrupt can be
 * found in the kernel info page.
 */

#include <14/re/c/util/cap_alloc.h>
#include <14/re/c/namespace.h>
#include <14/sys/utcb.h>
#include <14/sys/irq.h>
#include <14/sys/factory.h>
#include <14/sys/icu.h>

#include <stdio.h>

int main(void)
{
    int irqno = 1;
    l4_cap_idx_t irqcap, icucap;
    l4_msntag_t tag;
    int err;
    icucap = 14re_env_get_cap("icu");

    /* Get a free capability slot for the ICU capability */
    if (14_is_invalid_cap(icucap))
    {
        printf("Did not find an ICU\n");
        return 1;
    }

    /* Get another free capability slot for the corresponding IRQ object*/
    if (14_is_invalid_cap(irqcap = 14re_util_cap_alloc()))

```

```

        return 1;
/* Create IRQ object */
if (l4_error(tag = l4_factory_create_irq(l4re_global_env->factory, irqcap)))
{
    printf("Could not create IRQ object: %lx\n", l4_error(tag));
    return 1;
}

/*
 * Bind the recently allocated IRQ object to the IRQ number irqno
 * as provided by the ICU.
 */
if (l4_error(l4_icu_bind(icucap, irqno, irqcap)))
{
    printf("Binding IRQ%d to the ICU failed\n", irqno);
    return 1;
}

/* Attach ourselves to the IRQ */
tag = l4_irq_attach(irqcap, 0xDEAD, l4re_env()>main_thread);
if ((err = l4_error(tag)))
{
    printf("Error attaching to IRQ %d: %d\n", irqno, err);
    return 1;
}

printf("Attached to key IRQ %d\nPress keys now, Shift-Q to exit\n", irqno);

/* IRQ receive loop */
while (1)
{
    unsigned long label = 0;
    /* Wait for the interrupt to happen */
    tag = l4_irq_receive(irqcap, L4_IPC_NEVER);
    if ((err = l4_ipc_error(tag, l4_utcb())))
        printf("Error on IRQ receive: %d\n", err);
    else
    {
        /* Process the interrupt -- may do a 'break' */
        printf("Got IRQ with label 0x%lx\n", label);
    }
}

/* We're done, detach from the interrupt. */
tag = l4_irq_detach(irqcap);
if ((err = l4_error(tag)))
    printf("Error detach from IRQ: %d\n", err);

return 0;
}

```

12.19 examples/sys/migrate/thread_migrate.cc

Thread migration example.

```

/*
 * (c) 2008-2009 Author(s)
 *      economic rights: Technische Universitt 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-14.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("Found %d CPUs.\n", cpu_nrs);
    if (cpu_nrs >= L4_MWORD_BITS)
    {
        printf("Will only handle %d 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_getl4cap(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);
            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;
}

```

12.20 examples/sys/migrate/thread_migrate.cfg

Sample configuration file for the thread migration example.

```

-- vim:set ft=lua:

-- The log prefix will be 'migrate', colored green.
L4.default_loader:start({ log = { "migrate", "green" } },
                        "rom/ex_thread_migrate");

```

12.21 examples/sys/singlestep/main.c

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>,
 *      BjÄurn DÄbel <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_mshtag_t tag;
int ipc_stat = 0;
l4_cap_idx_t th = l4re_util_cap_alloc();
l4_exc_regs_t exc;
l4_umword_t mr0, mrl;
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_mshtag_has_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_mshtag_has_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_mshtag_has_error(tag))
    return 3;

/* Pager/Exception loop */
if (l4_mshtag_has_error(tag = l4_ipc_receive(th, u, L4_IPC_NEVER)))
{
    printf("l4_ipc_receive failed");
    return 4;
}
memcpy(&exc, l4_utcb_exc(), sizeof(exc));
mr0 = l4_utcb_mr()>mr[0];
mrl = l4_utcb_mr()>mr[1];

for (;;)
{
    if (l4_mshtag_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_mshtag(L4_PROTO_ALLOW_SYSCALL,
                    L4_UTCB_EXCEPTION_REGS_SIZE, 0, 0);
                if (ipc_stat)
                    enter_kdebug("Should not be 1");
            }
            else
            {
                tag = l4_mshtag(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_mshtag_label(tag), mr0, mr1);

memcpy(l4_utcb_exc(), &exc, sizeof(exc));

/* Reply and wait */
if (l4_mshtag_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;
}

```

12.22 examples/sys/start-with-exc/main.c

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>,
 *      BjÃ¶rn DÃ¶bel <doebel@os.inf.tu-dresden.de>,
 *      Frank Mehnert <fm3@os.inf.tu-dresden.de>
 *      economic rights: Technische UniversitÃ¤t 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 architecture.
 */

#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\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\t"
"thread:      \n\t"
"  pusha      \n\t"
"  push %esp      \n\t"
"  call thread_func  \n\t"
);
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_mshtag_t tag;
    int err;
    extern char _start[], _end[], _etext[];

    if (l4_is_invalid_cap(t1))
        return 1;

    /* Prevent pagefaults of our new thread because we do not want to
     * implement a pager as well. */
    l4_touch_ro(_start, _end - _start + 1);
    l4_touch_rw(_etext, _end - _etext);

    /* Create the thread using our default factory */
    tag = l4_factory_create_thread(l4re_env()>factory, t1);
    if (l4_mshtag_has_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_mshtag_has_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_mshtag_has_error(tag))
        return 3;

    /* 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_mshtag_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_mshtag_label(tag));
        return 1;
    }

    /* Fill out the complete register set of the new thread */
    e->ip = (l4_umword_t)thread;
    e->sp = (l4_umword_t)(thread_stack + sizeof(thread_stack));
    e->eax = 1;
    e->ebx = 4;
    e->ecx = 2;
    e->edx = 3;
    e->esi = 6;
    e->edi = 7;
    e->ebp = 5;
    /* Send a complete exception */
    tag = l4_mshtag(0, L4_UTCB_EXCEPTION_REGS_SIZE, 0, 0);

    /* Send reply and start the thread with the defined CPU register set */
    tag = l4_ipc_send(t1, 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;
}

```

12.23 examples/sys/utcb-ipc/main.c

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>,
 *      BjÃ¶rn DÃ¶bel <doebel@os.inf.tu-dresden.de>
 *      economic rights: Technische UniversitÃ¤t 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/re/ env.h>
#include <l4/re/ c/ util/ cap_alloc.h>

#include <stdio.h>
#include <string.h>

static unsigned char stack2[8 << 10];
static l4_cap_idx_t thread1_cap, thread2_cap;

static void thread1(void)

```

```

{
    14_msg_regs_t *mr = 14_utcb_mr();
    14_mshtag_t tag;
    int i, j;

    printf("Thread1 up (%p)\n", 14_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 = 14_mshtag(0, L4_UTCB_GENERIC_DATA_SIZE, 0, 0);
        if (14_mshtag_has_error(14_ipc_send(thread2_cap, 14_utcb(), tag,
            L4_IPC_NEVER)))
            printf("IPC-send error\n");
    }
}

static void thread2(void)
{
    14_mshtag_t tag;
    14_msg_regs_t mr;
    unsigned i;

    printf("Thread2 up (%p)\n", 14_utcb());

    while (1)
    {
        if (14_mshtag_has_error(tag = 14_ipc_receive(thread1_cap, 14_utcb(),
            L4_IPC_NEVER)))
            printf("IPC receive error\n");
        memcpy(&mr, 14_utcb_mr(), sizeof(mr));
        printf("Thread2 receive (%d): ", 14_mshtag_words(tag));
        for (i = 0; i < 14_mshtag_words(tag); i++)
            printf("%c", (char)mr.mr[i]);
        printf("\n");
    }
}

int main(void)
{
    14_mshtag_t tag;

    thread1_cap = 14re_env()->main_thread;
    thread2_cap = 14re_util_cap_alloc();

    if (14_is_invalid_cap(thread2_cap))
        return 1;

    tag = 14_factory_create_thread(14re_env()->factory, thread2_cap);
    if (14_mshtag_has_error(tag))
        return 1;

    14_thread_control_start();
    14_thread_control_pager(14re_env()->rm);
    14_thread_control_exc_handler(14re_env()->rm);
    14_thread_control_bind((14_utcb_t *)14re_env()->first_free_utcb,
        L4RE_THIS_TASK_CAP);
    tag = 14_thread_control_commit(thread2_cap);
    if (14_mshtag_has_error(tag))
        return 2;

    tag = 14_thread_ex_regs(thread2_cap,
        (14_umword_t)thread2,
        (14_umword_t)(stack2 + sizeof(stack2)), 0);
    if (14_mshtag_has_error(tag))
        return 3;
}

```

```

    thread1();

    return 0;
}

```

12.24 examples/sys/ux-vhw/main.c

This example shows how to iterate the virtual hardware descriptors under Fiasco-UX.

```

/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 *      Alexander Warg <warg@os.inf.tu-dresden.de>
 *      economic rights: Technische Universitt 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/vhw.h>
#include <l4/util/util.h>
#include <l4/util/kip.h>
#include <l4/re/env.h>

#include <stdlib.h>
#include <stdio.h>

static void print_entry(struct l4_vhw_entry *e)
{
    printf("type: %d  mem start: %08lx  end: %08lx\n"
           "irq: %d pid %d\n",
           e->type, e->mem_start, e->mem_size,
           e->irq_no, e->provider_pid);
}

int main(void)
{
    l4_kernel_info_t *kip = l4re_kip();
    struct l4_vhw_descriptor *vhw;
    int i;

    if (!kip)
    {
        printf("KIP not available!\n");
        return 1;
    }

    vhw = l4_vhw_get(kip);

    printf("kip at %p, vhw at %p\n", kip, vhw);
    printf("magic: %08x, version: %08x, count: %02d\n",
           vhw->magic, vhw->version, vhw->count);

    for (i = 0; i < vhw->count; i++)
        print_entry(l4_vhw_get_entry(vhw, i));

    return 0;
}

```

12.25 hello/server/src/main.c

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 GrÄijtzmacher <lg2@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 <unistd.h>

int
main(void)
{
    for (;;)
    {
        puts("Hello World!");
        sleep(1);
    }
}
```

12.26 tmpfs/lib/src/fs.cc

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 <14/l4re_vfs/backend>
#include <14/cxx/string>
#include <14/cxx/avl_tree>

#include <sys/stat.h>
#include <sys/ioctl.h>
#include <dirent.h>

#include <cstdio>

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() const { return _size; }

~File_data() throw() { 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;
        _buf = realloc(_buf, _size);
    }

    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;
}

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() throw() { ++_ref_cnt; }
    int remove_ref() throw() { 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) throw()
        : _dir(d), _getdents_state(false) {}

```

```

int get_entry(const char *, int, mode_t, Ref_ptr<File> *) throw();
ssize_t getdents(char *, size_t) throw();
int fstat64(struct stat64 *buf) const throw();
int utime(const struct utimbuf *) throw();
int fchmod(mode_t) throw();
int mkdir(const char *, mode_t) throw();
int unlink(const char *) throw();
int rename(const char *, const char *) throw();

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) throw()
        : Be_file_pos(), _file(f) {}

    off64_t size() const throw();
    int fstat64(struct stat64 *buf) const throw();
    int ioctl(unsigned long, va_list) throw();
    int utime(const struct utimbuf *) throw();
    int fchmod(mode_t) throw();

private:
    ssize_t preadv(const struct iovec *v, int iovcnt, off64_t p) throw();
    ssize_t pwritev(const struct iovec *v, int iovcnt, off64_t p) throw();
    Ref_ptr<Pers_file> _file;
};

ssize_t Tmpfs_file::preadv(const struct iovec *v, int iovcnt, off64_t p) throw()
{
    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) throw()
{
    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::fstat64(struct stat64 *buf) const throw()
{

```

```

_file->info()->st_size = _file->data().size();
memcpy(buf, _file->info(), sizeof(*buf));
return 0;
}

off64_t Tmpfs_file::size() const throw()
{ return _file->data().size(); }

int
Tmpfs_file::ioctl(unsigned long v, va_list args) throw()
{
    switch (v)
    {
        case FIONREAD: // return amount of data still available
            int *available = va_arg(args, int *);
            *available = _file->data().size() - pos();
            return 0;
        };
        return -EINVAL;
    }

int
Tmpfs_file::utime(const struct utimbuf *times) throw()
{
    _file->info()->st_atime = times->actime;
    _file->info()->st_mtime = times->modtime;
    return 0;
}

int
Tmpfs_file::fchmod(mode_t m) throw()
{
    _file->info()->st_mode = m;
    return 0;
}

int
Tmpfs_dir::get_entry(const char *name, int flags, mode_t mode,
                     Ref_ptr<File> *file) throw()
{
    Ref_ptr<Node> path;
    if (!*name)
    {
        *file = 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 = new Tmpfs_dir(cxx::ref_ptr_static_cast<Pers_dir>(path));
else
    *file = 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) throw()
{
    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) + l;
        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::fstat64(struct stat64 *buf) const throw()
{
    memcpy(buf, _dir->info(), sizeof(*buf));
    return 0;
}

int
Tmpfs_dir::utime(const struct utimbuf *times) throw()
{
    _dir->info()->st_atime = times->actime;
    _dir->info()->st_mtime = times->modtime;
}

```

```

        return 0;
    }

    int
    Tmpfs_dir::fchmod(mode_t m) throw()
    {
        _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) throw()
    {

```

```

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) throw()
{
    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) throw()
{
    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) throw()
    {
        (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;

}
```

Index

~Auto_ptr
 cxx::Auto_ptr, 437

~Be_file_system
 L4Re::Vfs::Be_file_system, 482

_c
 L4::Cap_base, 526

a
 L4Re::Video::Pixel_info, 828, 829

add
 L4::Thread::Modify_senders, 799
 L4vcpu::State, 881

add_ku_mem
 L4::Task, 894

alloc
 cxx::Base_slab_static, 473
 cxx::List_alloc, 774
 cxx::Slab, 868
 cxx::Slab_static, 871
 L4Re::Cap_alloc, 514, 515
 L4Re::Mem_alloc, 786

alloc_fd
 L4Re::Vfs::Fs, 630

allocate
 L4Re::Dataspace, 543

amd64 Virtual Registers (UTCB), 294

api_l4io
 L4IO_DEVICE_ANY, 371
 L4IO_DEVICE_INVALID, 371
 L4IO_DEVICE_OTHER, 371
 L4IO_DEVICE_PCI, 371
 L4IO_DEVICE_USB, 371
 L4IO_MEM_CACHED, 370
 L4IO_MEM_EAGER_MAP, 370
 L4IO_MEM_NONCACHED, 370
 L4IO_MEM_USE_MTRR, 370
 L4IO_MEM_USE_RESERVED_AREA, 370
 L4IO_RESOURCE_ANY, 371
 L4IO_RESOURCE_INVALID, 371
 L4IO_RESOURCE_IRQ, 371
 L4IO_RESOURCE_MEM, 371
 L4IO_RESOURCE_PORT, 371

api_l4re_c_rm
 L4RE_RM_ATTACH_FLAGS, 76
 L4RE_RM_EAGER_MAP, 76

 L4RE_RM_IN_AREA, 76
 L4RE_RM_NO_ALIAS, 76
 L4RE_RM_OVERMAP, 76
 L4RE_RM_PAGER, 76
 L4RE_RM_READ_ONLY, 76
 L4RE_RM_REGION_FLAGS, 76
 L4RE_RM_RESERVED, 76
 L4RE_RM_SEARCH_ADDR, 76

api_l4re_c_video
 F_l4re_video_goops_auto_refresh, 89
 F_l4re_video_goops_dynamic_buffers, 89
 F_l4re_video_goops_dynamic_views, 89
 F_l4re_video_goops_pointer, 89
 F_l4re_video_view_above, 90
 F_l4re_video_view_dyn_allocated, 90
 F_l4re_video_view_flags_mask, 90
 F_l4re_video_view_none, 89
 F_l4re_video_view_set_background, 90
 F_l4re_video_view_set_buffer, 89
 F_l4re_video_view_set_buffer_offset, 90
 F_l4re_video_view_set_bytes_per_line, 90
 F_l4re_video_view_set_flags, 90
 F_l4re_video_view_set_pixel, 90
 F_l4re_video_view_set_position, 90

api_l4re_elf_aux
 L4RE_ELF_AUX_T_KIP_ADDR, 98
 L4RE_ELF_AUX_T_NONE, 98
 L4RE_ELF_AUX_T_STACK_ADDR, 98
 L4RE_ELF_AUX_T_STACK_SIZE, 98
 L4RE_ELF_AUX_T_VMA, 98

api_l4re_protocols
 Dataspace, 108
 Debug, 108
 Default, 108
 Event, 108
 Goos, 108
 Mem_alloc, 108
 Namespace, 108
 Parent, 108
 Rm, 108

api_libvcpu
 L4VCPU_IRQ_STATE_DISABLED, 391
 L4VCPU_IRQ_STATE_ENABLED, 391

api_calls_fiasco
 fiasco_gdt_get_entry_offset, 132

fiasco_gdt_set, 132
 fiasco_ldt_set, 131
 fiasco_tbuf_clear, 130
 fiasco_tbuf_dump, 130
 fiasco_tbuf_get_status, 128
 fiasco_tbuf_get_status_phys, 128
 fiasco_tbuf_log, 128
 fiasco_tbuf_log_3val, 129
 fiasco_tbuf_log_binary, 130
 fiasco_watchdog_takeover, 131
 fiasco_watchdog_touch, 131
 api_calls_rt_sched
 l4_next_period_id, 140
 l4_preemption_id, 140
 l4_rt_begin_minimal_periodic, 136
 l4_rt_begin_strictly_periodic, 135
 l4_rt_end_periodic, 137
 l4_rt_generic, 141
 l4_rt_next_period, 139
 l4_rt_next_reservation, 139
 l4_rt_remove, 137
 l4_rt_set_period, 138
 slice, 134, 135
 api_gfxbitmap_bitmap
 gfxbitmap_bmap, 364
 gfxbitmap_color_pix_t, 363
 gfxbitmap_color_t, 363
 gfxbitmap_convert_color, 363
 gfxbitmap_copy, 365
 gfxbitmap_fill, 363
 gfxbitmap_set, 364
 api_gfxbitmap_font
 gfxbitmap_font_data, 367
 gfxbitmap_font_get, 367
 gfxbitmap_font_height, 367
 gfxbitmap_font_init, 367
 gfxbitmap_font_text, 368
 gfxbitmap_font_text_scale, 368
 gfxbitmap_font_width, 367
 api_l4io
 l4io_device_types_t, 370
 l4io_has_resource, 374
 l4io_iomem_flags_t, 370
 l4io_lookup_device, 373
 l4io_lookup_resource, 373
 l4io_release_iomem, 372
 l4io_release_ioport, 373
 l4io_request_iomem, 371
 l4io_request_iomem_region, 371
 l4io_request_ioport, 372
 l4io_request_resource_iomem, 374
 l4io_resource_t, 370
 l4io_resource_types_t, 371
 l4io_search_iomem_region, 372
 api_l4re_c_debug
 l4re_debug_obj_debug, 63
 api_l4re_c_ds
 l4re_ds_allocate, 61
 l4re_ds_clear, 61
 l4re_ds_copy_in, 61
 l4re_ds_flags, 61
 l4re_ds_info, 61
 l4re_ds_phys, 62
 l4re_ds_size, 61
 api_l4re_c_event
 l4re_event_get_axis_info, 65
 l4re_event_get_buffer, 64
 l4re_event_get_num_streams, 64
 l4re_event_get_stream_info, 64
 l4re_event_get_stream_info_for_id, 65
 api_l4re_c_log
 l4re_log_print, 66
 l4re_log_print_srv, 67
 l4re_log_printn, 67
 l4re_log_printn_srv, 68
 api_l4re_c_mem_alloc
 l4re_ma_alloc, 70
 l4re_ma_alloc_srv, 71
 l4re_ma_flags, 70
 l4re_ma_free, 71
 l4re_ma_free_srv, 72
 api_l4re_c_ns
 l4re_ns_query_to_srv, 74
 l4re_ns_register_flags, 74
 l4re_ns_register_obj_srv, 74
 api_l4re_c_rm
 l4re_rm_attach, 77
 l4re_rm_attach_srv, 83
 l4re_rm_detach, 78
 l4re_rm_detach_ds, 79
 l4re_rm_detach_ds_unmap, 80
 l4re_rm_detach_srv, 84
 l4re_rm_detach_unmap, 80
 l4re_rm_find, 81
 l4re_rm_find_srv, 84
 l4re_rm_flags_t, 76
 l4re_rm_free_area, 76
 l4re_rm_free_area_srv, 83
 l4re_rm_reserve_area, 76
 l4re_rm_reserve_area_srv, 82
 l4re_rm_show_lists, 82
 api_l4re_c_util_cap
 l4re_util_cap_last, 85
 api_l4re_c_util_kumem_alloc
 l4re_util_kumem_alloc, 86
 api_l4re_c_video
 l4re_video_goops_create_buffer, 90
 l4re_video_goops_create_view, 91

l4re_video_goos_delete_buffer, 91
l4re_video_goos_delete_view, 91
l4re_video_goos_get_static_buffer, 91
l4re_video_goos_get_view, 92
l4re_video_goos_info, 90
l4re_video_goos_info_flags_t, 89
l4re_video_goos_refresh, 90
l4re_video_view_get_info, 92
l4re_video_view_info_flags_t, 89
l4re_video_view_refresh, 92
l4re_video_view_set_info, 92
l4re_video_view_set_viewport, 93
l4re_video_view_stack, 93
l4re_video_view_t, 89

api_l4re_elf_aux
 L4RE_ELF_AUX_ELEM, 97
 L4RE_ELF_AUX_ELEM_T, 97

api_l4re_env
 l4re_env, 100
 l4re_env_get_cap, 101
 l4re_env_get_cap_e, 101
 l4re_env_get_cap_l, 102
 l4re_env_t, 100
 l4re_kip, 100

api_l4re_protocols
 Protocols, 108

api_l4shm
 l4shmc_area_overhead, 400
 l4shmc_area_size, 400
 l4shmc_area_size_free, 400
 l4shmc_attach, 399
 l4shmc_attach_to, 399
 l4shmc_chunk_overhead, 400
 l4shmc_connect_chunk_signal, 400
 l4shmc_create, 399

api_l4shmc_chunk
 l4shmc_add_chunk, 402
 l4shmc_chunk_capacity, 403
 l4shmc_chunk_ptr, 403
 l4shmc_chunk_signal, 404
 l4shmc_get_chunk, 402
 l4shmc_get_chunk_to, 402
 l4shmc_iterate_chunk, 403

api_l4shmc_chunk_cons
 l4shmc_chunk_consumed, 408
 l4shmc_chunk_size, 408
 l4shmc_enable_chunk, 407
 l4shmc_is_chunk_ready, 408
 l4shmc_wait_chunk, 407
 l4shmc_wait_chunk_to, 407
 l4shmc_wait_chunk_try, 407

api_l4shmc_chunk_prod
 l4shmc_chunk_ready, 405
 l4shmc_chunk_ready_sig, 405

l4shmc_chunk_try_to_take, 405
l4shmc_is_chunk_clear, 405

api_l4shmc_signal
 l4shmc_add_signal, 410
 l4shmc_attach_signal, 410
 l4shmc_attach_signal_to, 410
 l4shmc_check_magic, 411
 l4shmc_get_signal_to, 411
 l4shmc_signal_cap, 411

api_l4shmc_signal_cons
 l4shmc_enable_signal, 413
 l4shmc_wait_any, 413
 l4shmc_wait_any_to, 414
 l4shmc_wait_any_try, 413
 l4shmc_wait_signal, 414
 l4shmc_wait_signal_to, 414
 l4shmc_wait_signal_try, 415

api_l4shmc_signal_prod
 l4shmc_trigger, 412

api_libvcpu
 l4vcpu_halt, 395
 l4vcpu_irq_disable, 392
 l4vcpu_irq_disable_save, 393
 l4vcpu_irq_enable, 393
 l4vcpu_irq_restore, 394
 l4vcpu_irq_state_t, 391
 l4vcpu_is_irq_entry, 396
 l4vcpu_is_page_fault_entry, 396
 l4vcpu_print_state, 395
 l4vcpu_state, 391

api_libvcpu_ext
 l4vcpu_ext_alloc, 397

ARM Virtual Registers (UTCB), 292

asm_enter_kdebug
 l4_debugger_api, 173, 174

Atomic Instructions, 306

attach
 L4:::Irq, 679
 L4Re:::Rm, 843, 844
 L4Re:::Util::Event_buffer_t, 608

Attach_flags
 L4Re:::Rm, 841

Attach_flags
 L4Re:::Rm, 840

Attr
 L4:::Thread::Attr, 430

Auto_ptr
 cxx:::Auto_ptr, 437

auto_refresh
 L4Re:::Video:::Goos:::Info, 657

Auxiliary data, 104

avail
 cxx:::List_alloc, 774

Avl_map

cxx::Avl_map, 443
 Avl_set
 cxx::Avl_set, 450, 451
 ax
 l4_vcpu_regs_t, 736
 b
 L4Re::Video::Pixel_info, 828, 829
 Base API, 115
 Basic Macros, 122
 Be_file_system
 L4Re::Vfs::Be_file_system, 482
 begin
 cxx::Avl_set, 453, 454
 cxx::Bits::Bst, 498, 499
 bind
 L4::Icu, 649
 L4::Thread::Attr, 432
 bind_thread
 L4::Ipc_gate, 676
 bit
 cxx::Bitmap_base, 487
 Bit Manipulation, 313
 Bitmap
 cxx::Bitmap, 485
 Bitmap graphics and fonts, 361
 bits_per_pixel
 L4Re::Video::Pixel_info, 828
 bp
 l4_vcpu_regs_t, 736
 buf
 L4::Ipc::Buf_cp_out, 508
 L4Re::Util::Event_buffer_t, 608
 Buf_cp_in
 L4::Ipc::Buf_cp_in, 506
 buf_cp_in
 ipc_fw, 54
 Buf_cp_out
 L4::Ipc::Buf_cp_out, 507
 buf_cp_out
 ipc_fw, 54
 Buf_in
 L4::Ipc::Buf_in, 509
 buf_in
 ipc_fw, 55
 buffer
 L4Re::Util::Event_t, 611
 Buffer Registers (BRs), 277
 bx
 l4_vcpu_regs_t, 736
 bytes_per_pixel
 L4Re::Video::Pixel_info, 828, 830
 C++ Exceptions, 43
 Cache Consistency, 163
 call
 L4::Ipc::Iostream, 671
 Cap
 L4::Cap, 512, 513
 cap
 L4::Cap_base, 522
 L4::Invalid_capability, 666
 L4::Kobject, 702
 cap_alloc
 l4re_cap_api, 110
 Cap_base
 L4::Cap_base, 521
 cap_cast
 l4_cap_api, 270
 cap_dynamic_cast
 l4_cap_api, 271
 cap_equal
 L4::Task, 894
 cap_has_child
 L4::Task, 893
 cap_reinterpret_cast
 l4_cap_api, 270
 Cap_type
 L4::Cap_base, 521
 cap_valid
 L4::Task, 892
 Capabilities, 267
 Capability allocator, 85
 cast
 L4vcpu::Vcpu, 925
 chain
 L4::Irq, 680
 chars
 cxx::Bitmap_base, 487
 Chunks, 401
 clear
 L4Re::Dataspace, 543
 L4Re::Util::Dataspace_svr, 550
 L4vcpu::State, 881
 clear_all
 cxx::Bitmap, 485
 clear_bit
 cxx::Bitmap_base, 488
 Client/Server IPC Framework, 46
 cnt_iobmap_tlb_flush
 l4_tracebuffer_status_t, 730
 Color_component
 L4Re::Video::Color_component, 528
 Com_error
 L4::Com_error, 533
 Comfortable Command Line Parsing, 347
 Console API, 93
 Consumer, 406, 412

Continuous
 L4Re::Mem_alloc, 786

control
 L4::Thread, 900

copy
 L4Re::Util::Dataspace_svr, 550

copy_in
 L4Re::Dataspace, 544

count
 L4::Kip::Mem_desc, 789
 l4_vhw_descriptor, 742

CPU related functions, 296

create
 L4::Factory, 617

create_buffer
 L4Re::Video::Goos, 640

create_factory
 L4::Factory, 620

create_gate
 L4::Factory, 620

create_irq
 L4::Factory, 621

create_task
 L4::Factory, 618

create_thread
 L4::Factory, 619

create_view
 L4Re::Video::Goos, 641

create_vm
 L4::Factory, 622

cx
 l4_vcpu_regs_t, 736

cxx, 419

cxx::Auto_ptr, 436
 ~Auto_ptr, 437
 Auto_ptr, 437
 get, 438
 operator Priv_type *, 439
 operator*, 438
 operator->, 438
 operator=, 438
 Ref_type, 437
 release, 438

cxx::Avl_map, 439
 Avl_map, 443
 erase, 445
 find, 444
 find_node, 444
 insert, 443
 lower_bound_node, 444
 remove, 444

cxx::Avl_set, 446
 Avl_set, 450, 451
 begin, 453, 454
 end, 453, 454
 find_node, 452
 insert, 451
 lower_bound_node, 452
 rbegin, 454, 455
 remove, 452
 rend, 454, 455

cxx::Avl_set::Node, 806
 valid, 807

cxx::Avl_tree, 455
 insert, 459
 Iterator, 459
 remove, 460

cxx::Avl_tree_node, 461

cxx::Base_slab
 max_free_slabs, 468
 object_size, 468
 objects_per_slab, 468
 slab_size, 468

cxx::Base_slab_static
 max_free_slabs, 472
 object_size, 472
 objects_per_slab, 472
 slab_size, 472

cxx::Base_slab, 466
 free_objects, 469
 total_objects, 469

cxx::Base_slab_static, 470
 alloc, 473
 free, 473
 free_objects, 473
 total_objects, 473

cxx::Bitmap, 482
 Bitmap, 485
 clear_all, 485

cxx::Bitmap_base, 485
 bit, 487
 chars, 487
 clear_bit, 488
 scan_zero, 489
 set_bit, 488
 words, 487

cxx::Bitmap_base::Char, 527

cxx::Bitmap_base::Word, 934

cxx::Bits, 421

cxx::Bits::Bst, 493
 begin, 498, 499
 dir, 497
 end, 498, 499
 find, 502
 find_node, 500
 lower_bound_node, 501
 rbegin, 499, 500
 rend, 500

cxx::Bits::Bst_node, 503
 cxx::Bits::Direction, 561
 Direction_e, 562
 L, 562
 N, 562
 R, 562
 cxx::List, 770
 items, 773
 push_back, 771
 push_front, 771
 remove, 772
 size, 772
 cxx::List::Iter, 699
 cxx::List_alloc, 773
 alloc, 774
 avail, 774
 free, 774
 List_alloc, 773
 cxx::List_item, 775
 get_next_item, 776
 get_prev_item, 776
 insert_next_item, 776
 insert_prev_item, 776
 push_back, 777
 push_front, 777
 remove, 778
 remove_me, 777
 cxx::List_item::Iter, 695
 remove_me, 698
 cxx::List_item::T_iter, 882
 remove_me, 885
 cxx::Lt_functor, 783
 cxx::New_allocator, 806
 cxx::Nothrow, 807
 cxx::Pair, 819
 Pair, 820
 cxx::Pair_first_compare, 820
 operator(), 821
 Pair_first_compare, 821
 cxx::Slab, 865
 alloc, 868
 free, 868
 cxx::Slab_static, 869
 alloc, 871
 cxx_api
 max, 46
 min, 46
 operator new, 46

 d_val
 Elf32_Dyn, 574
 Elf64_Dyn, 579
 Data-Space API, 94
 data_space

L4Re::Vfs::Regular_file, 834
 Dataspace
 api_l4re_protocols, 108
 Dataspace interface, 60
 Debug
 api_l4re_protocols, 108
 debug
 L4Re::Debug_obj, 554
 Debug interface, 62
 Debugging API, 95
 dec_refcnt
 L4::Kobject, 702
 Default
 api_l4re_protocols, 108
 delete_buffer
 L4Re::Video::Goos, 640
 delete_obj
 L4::Task, 891
 delete_view
 L4Re::Video::Goos, 641
 descs
 l4_vhw_descriptor, 742
 detach
 L4::Irq, 681
 L4Re::Rm, 844, 845
 L4Re::Util::Event_buffer_t, 609
 Detach_again
 L4Re::Rm, 840
 Detach_exact
 L4Re::Rm, 841
 Detach_free
 L4Re::Rm, 840
 Detach_keep
 L4Re::Rm, 841
 Detach_overlap
 L4Re::Rm, 841
 Detach_flags
 L4Re::Rm, 841
 Detach_result
 L4Re::Rm, 840
 Detached_ds
 L4Re::Rm, 840
 DF_1_CONFALT
 l4util_elf, 344
 DF_1_DIRECT
 l4util_elf, 343
 DF_1_DISPRELDNE
 l4util_elf, 344
 DF_1_DISPRELPPND
 l4util_elf, 344
 DF_1_ENDFILTEE
 l4util_elf, 344
 DF_1_GLOBAL
 l4util_elf, 342

DF_1_GROUP
 l4util_elf, 343
DF_1_INTERPOSE
 l4util_elf, 343
DF_1_LOADFLTR
 l4util_elf, 343
DF_1_NODEFLIB
 l4util_elf, 343
DF_1_NODELETE
 l4util_elf, 343
DF_1_NODUMP
 l4util_elf, 343
DF_1_NOOPEN
 l4util_elf, 343
DF_1_NOW
 l4util_elf, 342
DF_1_ORIGIN
 l4util_elf, 343
DF_P1_GROUPPERM
 l4util_elf, 344
DF_P1_LAZYLOAD
 l4util_elf, 344
di
 l4_vcpu_regs_t, 736
dir
 cxx::Bits::Bst, 497
Direction_e
 cxx::Bits::Direction, 562
dispatch
 L4::Basic_registry, 475
 L4::Server_object, 865
 L4Re::Util::Vcon_svr, 916
 L4Re::Util::Video::Goos_svr, 645
drive_cylinders
 l4util_mb_drive_t, 765
drive_mode
 l4util_mb_drive_t, 765
drive_number
 l4util_mb_drive_t, 765
DT_HIPROC
 l4util_elf, 342
DT_LOPROC
 l4util_elf, 342
DT_NULL
 l4util_elf, 342
dump
 L4Re::Video::Color_component, 529
 L4Re::Video::Pixel_info, 830
dx
 l4_vcpu_regs_t, 736
e_phnum
 Elf32_Ehdr, 576
 Elf64_Ehdr, 580
e_shnum
 Elf32_Ehdr, 576
 Elf64_Ehdr, 580
Eager_map
 L4Re::Rm, 841
EI_CLASS
 l4util_elf, 336
EI_DATA
 l4util_elf, 336, 337
EI_OSABI
 l4util_elf, 338
EI_PAD
 l4util_elf, 340
EI_VERSION
 l4util_elf, 338
ELF binary format, 319
Elf32_Dyn, 574
 d_val, 574
Elf32_Ehdr, 574
 e_phnum, 576
 e_shnum, 576
Elf32_Phdr, 576
Elf32_Shdr, 577
Elf32_Sym, 578
Elf64_Dyn, 578
 d_val, 579
Elf64_Ehdr, 579
 e_phnum, 580
 e_shnum, 580
Elf64_Phdr, 581
Elf64_Shdr, 581
Elf64_Sym, 582
ELFCLASSNONE
 l4util_elf, 336
ELFDATA2LSB
 l4util_elf, 337
ELFDATA2MSB
 l4util_elf, 337
ELFDATANONE
 l4util_elf, 337
ELFOSABI_AIX
 l4util_elf, 339
ELFOSABI_FREEBSD
 l4util_elf, 339
ELFOSABI_HPUX
 l4util_elf, 339
ELFOSABI_IRIX
 l4util_elf, 339
ELFOSABI_LINUX
 l4util_elf, 339
ELFOSABI_MODESTO
 l4util_elf, 340
ELFOSABI_NETBSD
 l4util_elf, 339

ELFOSABI_OPENBSD
 l4util_elf, 340
 ELFOSABI_SOLARIS
 l4util_elf, 339
 ELFOSABI_SYSV
 l4util_elf, 338
 ELFOSABI_TRU64
 l4util_elf, 339
 EM_ARC
 l4util_elf, 340
 end
 cxx::Avl_set, 453, 454
 cxx::Bits::Bst, 498, 499
 L4::Kip::Mem_desc, 790
 enter_kdebug
 l4_debugger_api, 173, 174
 entry_ip
 L4vcpu::Vcpu, 924
 entry_sp
 L4vcpu::Vcpu, 924
 env
 L4Re::Env, 587
 erase
 cxx::Avl_map, 445
 Error codes, 179
 Error Handling, 204
 Event
 api_l4re_protocols, 108
 Event API, 103
 Event interface, 63
 Event_buffer_t
 L4Re::Event_buffer_t, 604
 ex_regs
 L4::Thread, 898, 899
 exc_handler
 L4::Thread::Attr, 431, 432
 Exception registers, 279
 ext_alloc
 L4vcpu::Vcpu, 925
 Extended vCPU support, 396

 F_above
 L4Re::Video::View, 929
 F_auto_refresh
 L4Re::Video::Goos, 639
 F_dyn_allocated
 L4Re::Video::View, 929
 F_dynamic_buffers
 L4Re::Video::Goos, 640
 F_dynamic_views
 L4Re::Video::Goos, 639
 F_flags_mask
 L4Re::Video::View, 929
 F_fully_dynamic
 L4Re::Video::View, 929

 L4Re::Video::View, 929
 F_l4re_video_goops_auto_refresh
 api_l4re_c_video, 89
 F_l4re_video_goops_dynamic_buffers
 api_l4re_c_video, 89
 F_l4re_video_goops_dynamic_views
 api_l4re_c_video, 89
 F_l4re_video_goops_pointer
 api_l4re_c_video, 89
 F_l4re_video_view_above
 api_l4re_c_video, 90
 F_l4re_video_view_dyn_allocated
 api_l4re_c_video, 90
 F_l4re_video_view_flags_mask
 api_l4re_c_video, 90
 F_l4re_video_view_none
 api_l4re_c_video, 89
 F_l4re_video_view_set_background
 api_l4re_c_video, 90
 F_l4re_video_view_set_buffer
 api_l4re_c_video, 89
 F_l4re_video_view_set_buffer_offset
 api_l4re_c_video, 90
 F_l4re_video_view_set_bytes_per_line
 api_l4re_c_video, 90
 F_l4re_video_view_set_flags
 api_l4re_c_video, 90
 F_l4re_video_view_set_pixel
 api_l4re_c_video, 90
 F_l4re_video_view_set_position
 api_l4re_c_video, 90
 F_none
 L4Re::Video::View, 929
 F_pointer
 L4Re::Video::Goos, 639
 F_set_background
 L4Re::Video::View, 929
 F_set_buffer
 L4Re::Video::View, 929
 F_set_buffer_offset
 L4Re::Video::View, 929
 F_set_bytes_per_line
 L4Re::Video::View, 929
 F_set_flags
 L4Re::Video::View, 929
 F_set_pixel
 L4Re::Video::View, 929
 F_set_position
 L4Re::Video::View, 929
 faccessat
 L4Re::Vfs::Directory, 565
 Factory, 180
 factory
 L4Re::Env, 588, 591

fchmod
 L4Re::Vfs::Generic_file, 635

fd
 l4_vhw_entry, 744

fdatasync
 L4Re::Vfs::Regular_file, 835

features
 l4_icu_info_t, 712

Fiasco extensions, 125

Fiasco real time scheduling extensions, 133

Fiasco-UX Virtual devices, 289

fiasco_gdt_get_entry_offset
 api_calls_fiasco, 132

fiasco_gdt_set
 api_calls_fiasco, 132

fiasco_ldt_set
 api_calls_fiasco, 131

fiasco_tbuf_clear
 api_calls_fiasco, 130

fiasco_tbuf_dump
 api_calls_fiasco, 130

fiasco_tbuf_get_status
 api_calls_fiasco, 128

fiasco_tbuf_get_status_phys
 api_calls_fiasco, 128

fiasco_tbuf_log
 api_calls_fiasco, 128

fiasco_tbuf_log_3val
 api_calls_fiasco, 129

fiasco_tbuf_log_binary
 api_calls_fiasco, 130

fiasco_watchdog_takeover
 api_calls_fiasco, 131

fiasco_watchdog_touch
 api_calls_fiasco, 131

find
 cxx::Avl_map, 444
 cxx::Bits::Bst, 502
 L4Re::Rm, 846

find_node
 cxx::Avl_map, 444
 cxx::Avl_set, 452
 cxx::Bits::Bst, 500

first
 L4::Kip::Mem_desc, 789

first_free_cap
 L4Re::Env, 588, 592

first_free_utcb
 L4Re::Env, 589, 592

Flags
 L4Re::Video::Goos, 639
 L4Re::Video::View, 929

flags
 l4_exc_regs_t, 710

l4_mshtag_t, 717

L4Re::Dataspace, 546

L4Re::Video::View::Info, 661

l4re_env_cap_entry_t, 752

Flex pages, 142

foreach_available_event
 L4Re::Util::Event_buffer_consumer_t, 600

fpage
 L4::Cap_base, 524

free
 cxx::Base_slab_static, 473
 cxx::List_alloc, 774
 cxx::Slab, 868
 L4Re::Cap_alloc, 515
 L4Re::Mem_alloc, 787

free_area
 L4Re::Rm, 843

free_fd
 L4Re::Vfs::Fs, 631

free_objects
 cxx::Base_slab, 469
 cxx::Base_slab_static, 473

fstat64
 L4Re::Vfs::Generic_file, 634

fsync
 L4Re::Vfs::Regular_file, 835

ftruncate64
 L4Re::Vfs::Regular_file, 835

Functions for rendering bitmap data in frame buffers, 362

Functions for rendering bitmap fonts to frame buffers, 365

Functions to manipulate the local IDT, 298

g
 L4Re::Video::Pixel_info, 828, 829

get
 cxx::Auto_ptr, 438
 L4::Ipc::Istream, 688, 689
 L4Re::Env, 589
 L4Re::Video::Color_component, 529

get_attr
 L4::Vcon, 915

get_buffer
 L4Re::Event, 596

get_cap
 L4Re::Env, 590

get_cap_alloc
 L4Re::Cap_alloc, 516

get_fb
 L4Re::Util::Video::Goos_svr, 644

get_file
 L4Re::Vfs::Fs, 630

get_infos

L4::Ipc_gate, 676
 get_lock
 L4Re::Vfs::Regular_file, 836
 get_next_item
 cxx::List_item, 776
 get_object_name
 L4::Debugger, 558
 get_prev_item
 cxx::List_item, 776
 get_static_buffer
 L4Re::Video::Goos, 640
 get_status_flags
 L4Re::Vfs::Generic_file, 635
 gfxbitmap_bmap
 api_gfxbitmap_bitmap, 364
 gfxbitmap_color_pix_t
 api_gfxbitmap_bitmap, 363
 gfxbitmap_color_t
 api_gfxbitmap_bitmap, 363
 gfxbitmap_convert_color
 api_gfxbitmap_bitmap, 363
 gfxbitmap_copy
 api_gfxbitmap_bitmap, 365
 gfxbitmap_fill
 api_gfxbitmap_bitmap, 363
 gfxbitmap_font_data
 api_gfxbitmap_font, 367
 gfxbitmap_font_get
 api_gfxbitmap_font, 367
 gfxbitmap_font_height
 api_gfxbitmap_font, 367
 gfxbitmap_font_init
 api_gfxbitmap_font, 367
 gfxbitmap_font_text
 api_gfxbitmap_font, 368
 gfxbitmap_font_text_scale
 api_gfxbitmap_font, 368
 gfxbitmap_font_width
 api_gfxbitmap_font, 367
 gfxbitmap_offset, 636
 gfxbitmap_set
 api_gfxbitmap_bitmap, 364
 global_id
 L4::Debugger, 557
 Goos
 api_l4re_protocols, 108
 Goos video API, 112

 halt
 L4vcpu::Vcpu, 923
 has_alpha
 L4Re::Video::Pixel_info, 829

 i

L4vcpu::Vcpu, 924
 IA32 Port I/O API, 357
 idle_time
 L4::Scheduler, 858
 In_area
 L4Re::Rm, 841
 info
 L4::Icu, 650
 L4::Scheduler, 857
 L4Re::Dataspace, 546
 L4Re::Video::Goos, 640
 L4Re::Video::View, 929
 init
 L4Re::Util::Event_t, 611
 init_infos
 L4Re::Util::Video::Goos_svr, 646
 Initial Environment, 98
 initial_caps
 L4Re::Env, 589, 593
 insert
 cxx::Avl_map, 443
 cxx::Avl_set, 451
 cxx::Avl_tree, 459
 insert_next_item
 cxx::List_item, 776
 insert_prev_item
 cxx::List_item, 776
 Integer Types, 415
 interface
 L4::Meta, 795
 Interface for asynchronous ISR handlers with a given IRQ capability., 382
 Interface for asynchronous ISR handlers., 379
 Interface using direct functionality., 375, 380
 Internal constants, 388
 Internal functions, 313
 internal_loop
 L4::Server, 862
 Interrupt controller, 186
 Invalid
 L4::Cap_base, 521
 Invalid_capability
 L4::Invalid_capability, 666
 IO interface, 369
 ioctl
 L4Re::Vfs::Special_file, 880
 Iostream
 L4::Ipc::Iostream, 670
 IPC Messaging Framework, 53
 IPC Streams, 47
 IPC-Gate API, 119
 ipc_fw
 buf_cp_in, 54
 buf_cp_out, 54

buf_in, 55
msg_ptr, 55
ipc_stream
 operator<<, 50–52
 operator>>, 47–50
irq
 L4Re::Util::Event_t, 612
IRQ handling library, 375
irq_disable_save
 L4vcpu::Vcpu, 921
irq_enable
 L4vcpu::Vcpu, 922
irq_no
 l4_vhw_entry, 744
irq_restore
 L4vcpu::Vcpu, 922
IRQs, 209
is_irq_entry
 L4vcpu::Vcpu, 923
is_online
 L4::Scheduler, 859
is_page_fault_entry
 L4vcpu::Vcpu, 923
is_valid
 L4::Cap_base, 523
is_virtual
 L4::Kip::Mem_desc, 791
Istream
 L4::Ipc::Istream, 688
items
 cxx::List, 773
Iterator
 cxx::Avl_tree, 459
kd_display
 l4_debugger_api, 173, 175
Kept_ds
 L4Re::Rm, 840
Kernel Debugger, 171
Kernel Interface Page, 219
Kernel Interface Page API, 345
Kernel Objects, 215
ko
 l4_debugger_api, 174, 175
kobj_to_id
 L4::Debugger, 557
kobject_typeid
 L4, 424
 L4::Kobject, 703
 L4::Kobject_2t, 705
 L4::Kobject_t, 706
Kumem allocator utility, 85
Kumem utilities, 111
kumem_alloc
 l4re_util_kumem, 111
L
 cxx::Bits::Direction, 562
L4, 421
 kobject_typeid, 424
L4::Alloc_list, 429
L4::Base_exception, 464
L4::Basic_registry, 474
 dispatch, 475
 Value, 474
L4::Bounds_error, 490
L4::Cap, 510
 Cap, 512, 513
 move, 513
L4::Cap_base
 Invalid, 521
 No_init, 521
L4::Cap_base, 519
 _c, 526
 cap, 522
 Cap_base, 521
 Cap_type, 521
 fpage, 524
 is_valid, 523
 No_init_type, 521
 snd_base, 525
 validate, 526
L4::Com_error, 530
 Com_error, 533
L4::Debugger, 554
 get_object_name, 558
 global_id, 557
 kobj_to_id, 557
 query_log_name, 558
 query_log_typeid, 558
 set_object_name, 557
 switch_log, 558
L4::Element_already_exists, 568
L4::Element_not_found, 571
L4::Exception_tracer, 612
L4::Factory, 614
 create, 617
 create_factory, 620
 create_gate, 620
 create_irq, 621
 create_task, 618
 create_thread, 619
 create_vm, 622
L4::Factory::Lstr, 782
L4::Factory::Nil, 806
L4::Factory::S, 850
 operator l4_mshtag_t, 853
 operator<<, 853, 854

S, 852
 L4::Icu, 646
 bind, 649
 info, 650
 mask, 652
 msi_info, 651
 set_mode, 653
 unbind, 650
 unmask, 652
 L4::Icu::Info, 661
 L4::Invalid_capability, 663
 cap, 666
 Invalid_capability, 666
 L4::IOModifier, 666
 L4::Ipc::Buf_cp_in, 506
 Buf_cp_in, 506
 L4::Ipc::Buf_cp_out, 507
 buf, 508
 Buf_cp_out, 507
 size, 508
 L4::Ipc::Buf_in, 509
 Buf_in, 509
 L4::Ipc::Iostream, 667
 call, 671
 Iostream, 670
 reply_and_wait, 672, 673
 reset, 671
 L4::Ipc::Istream, 684
 get, 688, 689
 Istream, 688
 receive, 692
 reset, 688
 skip, 689
 tag, 690
 wait, 690, 691
 L4::Ipc::Msg_ptr, 800
 Msg_ptr, 800
 L4::Ipc::Ostream, 810
 put, 814
 send, 815
 tag, 814, 815
 L4::Ipc::Small_buf, 871
 L4::Ipc_svr
 Reply_compound, 426
 Reply_separate, 426
 L4::Ipc_gate, 674
 bind_thread, 676
 get_infos, 676
 L4::Ipc_svr, 425
 Reply_mode, 425
 L4::Ipc_svr::Compound_reply, 533
 L4::Ipc_svr::Default_loop_hooks, 559
 L4::Ipc_svr::Default_setup_wait, 560
 L4::Ipc_svr::Default_timeout, 560
 L4::Ipc_svr::Ignore_errors, 654
 L4::Irq, 676
 attach, 679
 chain, 680
 detach, 681
 receive, 681
 trigger, 683
 unmask, 683
 wait, 682
 L4::Kip::Mem_desc, 787
 count, 789
 end, 790
 first, 789
 is_virtual, 791
 Mem_desc, 788
 set, 792
 size, 790
 start, 789
 sub_type, 791
 type, 791
 L4::Kobject, 701
 cap, 702
 dec_refcnt, 702
 kobject_typeid, 703
 L4::Kobject_2t, 704
 kobject_typeid, 705
 L4::Kobject_t, 705
 kobject_typeid, 706
 L4::Meta, 792
 interface, 795
 num_interfaces, 795
 supports, 796
 L4::Out_of_memory, 816
 L4::Runtime_error, 847
 L4::Scheduler, 854
 idle_time, 858
 info, 857
 is_online, 859
 run_thread, 857
 L4::Server, 860
 internal_loop, 862
 Server, 862
 L4::Server_object, 863
 dispatch, 865
 L4::Smart_cap, 872
 Smart_cap, 875
 L4::String, 882
 L4::Task, 885
 add_ku_mem, 894
 cap_equal, 894
 cap_has_child, 893
 cap_valid, 892
 delete_obj, 891
 map, 888

release_cap, 891
unmap, 889
unmap_batch, 890
L4::Thread, 895
 control, 900
 ex_regs, 898, 899
 modify_senders, 904
 register_del_irq, 903
 stats_time, 901
 switch_to, 900
 vcpu_control, 902
 vcpu_control_ext, 903
 vcpu_resume_commit, 902
 vcpu_resume_start, 901
L4::Thread::Attr, 429
 Attr, 430
 bind, 432
 exc_handler, 431, 432
 pager, 430, 431
 ux_host_syscall, 433
L4::Thread::Modify_senders, 799
 add, 799
L4::Type_info, 905
L4::Unknown_error, 906
L4::Vcon, 909
 get_attr, 915
 read, 913
 send, 912
 set_attr, 914
 write, 913
L4::Vm, 931
 run, 934
L4_BASE_FACTORY_CAP
 l4_cap_api, 269
L4_BASE_ICU_CAP
 l4_cap_api, 270
L4_BASE_LOG_CAP
 l4_cap_api, 269
L4_BASE_PAGER_CAP
 l4_cap_api, 269
L4_BASE_SCHEDULER_CAP
 l4_cap_api, 270
L4_BASE_TASK_CAP
 l4_cap_api, 269
L4_BASE_THREAD_CAP
 l4_cap_api, 269
L4_BDR_IO_SHIFT
 l4_utcb_br_api, 278
L4_BDR_MEM_SHIFT
 l4_utcb_br_api, 278
L4_BDR_OBJ_SHIFT
 l4_utcb_br_api, 278
l4_cap_api
 L4_BASE_FACTORY_CAP, 269
L4_BASE_ICU_CAP, 270
L4_BASE_LOG_CAP, 269
L4_BASE_PAGER_CAP, 269
L4_BASE_SCHEDULER_CAP, 270
L4_BASE_TASK_CAP, 269
L4_BASE_THREAD_CAP, 269
L4_CAP_MASK, 269
L4_CAP_SHIFT, 269
L4_CAP_SIZE, 269
L4_INVALID_CAP, 269
L4_CAP_FPAGE_R
 l4_fpage_api, 145
L4_CAP_FPAGE_RO
 l4_fpage_api, 145
L4_CAP_FPAGE_RW
 l4_fpage_api, 145
L4_CAP_MASK
 l4_cap_api, 269
L4_CAP_SHIFT
 l4_cap_api, 269
L4_CAP_SIZE
 l4_cap_api, 269
L4_EACCESS
 l4_error_api, 180
L4_EADDRNOTAVAIL
 l4_error_api, 180
L4_EAGAIN
 l4_error_api, 180
L4_EBADPROTO
 l4_error_api, 180
L4_EBUSY
 l4_error_api, 180
L4_EEXIST
 l4_error_api, 180
L4_EINVAL
 l4_error_api, 180
L4_EIO
 l4_error_api, 180
L4_EIPC_HI
 l4_error_api, 180
L4_EIPC_LO
 l4_error_api, 180
L4_ENAMETOOLONG
 l4_error_api, 180
L4_ENODEV
 l4_error_api, 180
L4_ENOENT
 l4_error_api, 180
L4_ENOMEM
 l4_error_api, 180
L4_ENOREPLY
 l4_error_api, 180
L4_ENOSYS
 l4_error_api, 180

L4_EOK
 l4_error_api, 180
 L4_EPERM
 l4_error_api, 180
 L4_ERANGE
 l4_error_api, 180
 L4_ERRNOMAX
 l4_error_api, 180
 l4_error_api
 L4_EACCESS, 180
 L4_EADDRNOTAVAIL, 180
 L4_EAGAIN, 180
 L4_EBADPROTO, 180
 L4_EBUSY, 180
 L4_EEXIST, 180
 L4_EINVAL, 180
 L4_EIO, 180
 L4_EIPC_HI, 180
 L4_EIPC_LO, 180
 L4_ENAMETOOLONG, 180
 L4_ENODEV, 180
 L4_ENOENT, 180
 L4_ENOMEM, 180
 L4_ENOREPLY, 180
 L4_ENOSYS, 180
 L4_EOK, 180
 L4_EPERM, 180
 L4_ERANGE, 180
 L4_ERRNOMAX, 180
 L4_FP_ALL_SPACES
 l4_task_api, 232
 L4_FP_DELETE_OBJ
 l4_task_api, 232
 L4_FP_OTHER_SPACES
 l4_task_api, 232
 L4_FPAGE_ADDR_BITS
 l4_fpage_api, 145
 L4_FPAGE_ADDR_SHIFT
 l4_fpage_api, 145
 l4_fpage_api
 L4_CAP_FPAGE_R, 145
 L4_CAP_FPAGE_RO, 145
 L4_CAP_FPAGE_RW, 145
 L4_FPAGE_ADDR_BITS, 145
 L4_FPAGE_ADDR_SHIFT, 145
 L4_FPAGE_BUFFERABLE, 146
 L4_FPAGE_CACHE_OPT, 146
 L4_FPAGE_CACHEABLE, 146
 L4_FPAGE_RIGHTS_BITS, 145
 L4_FPAGE_RIGHTS_SHIFT, 145
 L4_FPAGE_RO, 145
 L4_FPAGE_RW, 145
 L4_FPAGE_SIZE_BITS, 145
 L4_FPAGE_SIZE_SHIFT, 145
 L4_FPAGE_TYPE_BITS, 145
 L4_FPAGE_TYPE_SHIFT, 145
 L4_FPAGE_UNCACHEABLE, 146
 L4_IOPORT_MAX, 146
 L4_WHOLE_ADDRESS_SPACE, 145
 L4_WHOLE_IOADDRESS_SPACE, 146
 L4_FPAGE_BUFFERABLE
 l4_fpage_api, 146
 L4_FPAGE_CACHE_OPT
 l4_fpage_api, 146
 L4_FPAGE_CACHEABLE
 l4_fpage_api, 146
 L4_FPAGE_RIGHTS_BITS
 l4_fpage_api, 145
 L4_FPAGE_RIGHTS_SHIFT
 l4_fpage_api, 145
 L4_FPAGE_RO
 l4_fpage_api, 145
 L4_FPAGE_RW
 l4_fpage_api, 145
 L4_FPAGE_SIZE_BITS
 l4_fpage_api, 145
 L4_FPAGE_SIZE_SHIFT
 l4_fpage_api, 145
 L4_FPAGE_TYPE_BITS
 l4_fpage_api, 145
 L4_FPAGE_TYPE_SHIFT
 l4_fpage_api, 145
 L4_FPAGE_UNCACHEABLE
 l4_fpage_api, 146
 l4_icu_api
 L4_ICU_FLAG_MSI, 188
 L4_ICU_FLAG_MSI
 l4_icu_api, 188
 L4_INVALID_ADDR
 l4_memory_api, 168
 L4_INVALID_CAP
 l4_cap_api, 269
 L4_IOPORT_MAX
 l4_fpage_api, 146
 l4_ipc_api
 L4_SYSF_CALL, 195
 L4_SYSF_NONE, 195
 L4_SYSF_OPEN_WAIT, 195
 L4_SYSF_RECV, 195
 L4_SYSF_REPLY, 195
 L4_SYSF_REPLY_AND_WAIT, 196
 L4_SYSF_SEND, 195
 L4_SYSF_SEND_AND_WAIT, 196
 L4_SYSF_WAIT, 196
 L4_IPC_ENOT_EXISTENT
 l4_ipc_err_api, 205
 l4_ipc_err_api
 L4_IPC_ENOT_EXISTENT, 205

L4_IPC_ERROR_MASK, 205
L4_IPC_REABORTED, 205
L4_IPC_RECANCELED, 205
L4_IPC_REMAPFAILED, 205
L4_IPC_REMSGCUT, 205
L4_IPC_RERCVPFTO, 205
L4_IPC_RESNDPFTO, 205
L4_IPC_RETIMEOUT, 205
L4_IPC_SEABORTED, 205
L4_IPC_SECANCELED, 205
L4_IPC_SEMAPFAILED, 205
L4_IPC_SEMSGCUT, 205
L4_IPC_SERCVPFTO, 205
L4_IPC_SESNDPFTO, 205
L4_IPC_SETIMEOUT, 205
L4_IPC SND_ERR_MASK, 205
L4_IPC_ERROR_MASK
 l4_ipc_err_api, 205
L4_IPC_GATE_BIND_OP
 l4_kernel_object_gate_api, 120
L4_IPC_GATE_GET_INFO_OP
 l4_kernel_object_gate_api, 120
L4_IPC_REABORTED
 l4_ipc_err_api, 205
L4_IPC_RECANCELED
 l4_ipc_err_api, 205
L4_IPC_REMAPFAILED
 l4_ipc_err_api, 205
L4_IPC_REMSGCUT
 l4_ipc_err_api, 205
L4_IPC_RERCVPFTO
 l4_ipc_err_api, 205
L4_IPC_RESNDPFTO
 l4_ipc_err_api, 205
L4_IPC_RETIMEOUT
 l4_ipc_err_api, 205
L4_IPC_SEABORTED
 l4_ipc_err_api, 205
L4_IPC_SECANCELED
 l4_ipc_err_api, 205
L4_IPC_SEMAPFAILED
 l4_ipc_err_api, 205
L4_IPC_SEMSGCUT
 l4_ipc_err_api, 205
L4_IPC_SERCVPFTO
 l4_ipc_err_api, 205
L4_IPC_SESNDPFTO
 l4_ipc_err_api, 205
L4_IPC_SETIMEOUT
 l4_ipc_err_api, 205
L4_IPC SND_ERR_MASK
 l4_ipc_err_api, 205
l4_irq_api
 L4 IRQ_F_BOTH, 210
L4 IRQ_F_BOTH_EDGE, 211
L4 IRQ_F_EDGE, 210
L4 IRQ_F_LEVEL, 210
L4 IRQ_F_LEVEL_HIGH, 210
L4 IRQ_F_LEVEL_LOW, 210
L4 IRQ_F_MASK, 211
L4 IRQ_F_NEG, 210
L4 IRQ_F_NEG_EDGE, 211
L4 IRQ_F_NONE, 210
L4 IRQ_F_POS, 210
L4 IRQ_F_POS_EDGE, 211
L4 IRQ_F_BOTH
 l4_irq_api, 210
L4 IRQ_F_BOTH_EDGE
 l4_irq_api, 211
L4 IRQ_F_EDGE
 l4_irq_api, 210
L4 IRQ_F_LEVEL
 l4_irq_api, 210
L4 IRQ_F_LEVEL_HIGH
 l4_irq_api, 210
L4 IRQ_F_LEVEL_LOW
 l4_irq_api, 210
L4 IRQ_F_MASK
 l4_irq_api, 211
L4 IRQ_F_NEG
 l4_irq_api, 210
L4 IRQ_F_NEG_EDGE
 l4_irq_api, 211
L4 IRQ_F_NONE
 l4_irq_api, 210
L4 IRQ_F_POS
 l4_irq_api, 210
L4 IRQ_F_POS_EDGE
 l4_irq_api, 211
L4 ITEM_CONT
 l4_msgitem_api, 152
L4 ITEM_MAP
 l4_msgitem_api, 152
l4_kernel_object_gate_api
 L4 IPC_GATE_BIND_OP, 120
 L4 IPC_GATE_GET_INFO_OP, 120
l4_kip_memdesc_api
 l4_mem_type_archspecific, 224
 l4_mem_type_bootloader, 224
 l4_mem_type_conventional, 223
 l4_mem_type_dedicated, 223
 l4_mem_type_reserved, 223
 l4_mem_type_shared, 223
 l4_mem_type_undefined, 223
l4_kip_vhw_api
 L4 TYPE_VHW_FRAMEBUFFER, 290
 L4 TYPE_VHW_INPUT, 290
 L4 TYPE_VHW_NET, 290

L4_TYPE_VHW_NONE, 290
 L4_MAP_ITEM_GRANT
 l4_msgitem_api, 152
 L4_MAP_ITEM_MAP
 l4_msgitem_api, 152
 l4_mem_op_api
 L4_MEM_WIDTH_1BYTE, 291
 L4_MEM_WIDTH_2BYTE, 291
 L4_MEM_WIDTH_4BYTE, 291
 l4_mem_type_archspecific
 l4_kip_memdesc_api, 224
 l4_mem_type_bootloader
 l4_kip_memdesc_api, 224
 l4_mem_type_conventional
 l4_kip_memdesc_api, 223
 l4_mem_type_dedicated
 l4_kip_memdesc_api, 223
 l4_mem_type_reserved
 l4_kip_memdesc_api, 223
 l4_mem_type_shared
 l4_kip_memdesc_api, 223
 l4_mem_type_undefined
 l4_kip_memdesc_api, 223
 L4_MEM_WIDTH_1BYTE
 l4_mem_op_api, 291
 L4_MEM_WIDTH_2BYTE
 l4_mem_op_api, 291
 L4_MEM_WIDTH_4BYTE
 l4_mem_op_api, 291
 l4_memory_api
 L4_INVALID_ADDR, 168
 l4_msgitem_api
 L4_ITEM_CONT, 152
 L4_ITEM_MAP, 152
 L4_MAP_ITEM_GRANT, 152
 L4_MAP_ITEM_MAP, 152
 L4_RCV_ITEM_LOCAL_ID, 152
 L4_RCV_ITEM_SINGLE_CAP, 152
 l4_mshtag_api
 L4_MSGTAG_ERROR, 260
 L4_MSGTAG_FLAGS, 260
 L4_MSGTAG_PROPAGATE, 260
 L4_MSGTAG_SCHEDULE, 260
 L4_MSGTAG_TRANSFER_FPU, 260
 L4_MSGTAG_XCPU, 260
 L4_PROTO_ALLOW_SYSCALL, 259
 L4_PROTO_EXCEPTION, 259
 L4_PROTO_FACTORY, 259
 L4_PROTO_IO_PAGE_FAULT, 259
 L4_PROTO_IRQ, 259
 L4_PROTO_KOBJECT, 259
 L4_PROTO_LOG, 259
 L4_PROTO_META, 260
 L4_PROTO_NONE, 259
 L4_PROTO_PAGE_FAULT, 259
 L4_PROTO_PF_EXCEPTION, 259
 L4_PROTO_PREEMPTION, 259
 L4_PROTO_SCHEDULER, 259
 L4_PROTO_SIGMA0, 259
 L4_PROTO_SYS_EXCEPTION, 259
 L4_PROTO_TASK, 259
 L4_PROTO_THREAD, 259
 L4_PROTO_VM, 260
 L4_MSGTAG_ERROR
 l4_mshtag_api, 260
 L4_MSGTAG_FLAGS
 l4_mshtag_api, 260
 L4_MSGTAG_PROPAGATE
 l4_mshtag_api, 260
 L4_MSGTAG_SCHEDULE
 l4_mshtag_api, 260
 L4_MSGTAG_TRANSFER_FPU
 l4_mshtag_api, 260
 L4_MSGTAG_XCPU
 l4_mshtag_api, 260
 L4_PROTO_ALLOW_SYSCALL
 l4_mshtag_api, 259
 L4_PROTO_EXCEPTION
 l4_mshtag_api, 259
 L4_PROTO_FACTORY
 l4_mshtag_api, 259
 L4_PROTO_IO_PAGE_FAULT
 l4_mshtag_api, 259
 L4_PROTO_IRQ
 l4_mshtag_api, 259
 L4_PROTO_KOBJECT
 l4_mshtag_api, 259
 L4_PROTO_LOG
 l4_mshtag_api, 259
 L4_PROTO_META
 l4_mshtag_api, 260
 L4_PROTO_NONE
 l4_mshtag_api, 259
 L4_PROTO_PAGE_FAULT
 l4_mshtag_api, 259
 L4_PROTO_PF_EXCEPTION
 l4_mshtag_api, 259
 L4_PROTO_PREEMPTION
 l4_mshtag_api, 259
 L4_PROTO_SCHEDULER
 l4_mshtag_api, 259
 L4_PROTO_SIGMA0
 l4_mshtag_api, 259
 L4_PROTO_SYS_EXCEPTION
 l4_mshtag_api, 259
 L4_PROTO_TASK
 l4_mshtag_api, 259
 L4_PROTO_THREAD

l4_mshtag_api, 259
L4_PROTO_VM
 l4_mshtag_api, 260
L4_RCV_ITEM_LOCAL_ID
 l4_msgetitem_api, 152
L4_RCV_ITEM_SINGLE_CAP
 l4_msgetitem_api, 152
l4_scheduler_api
 L4_SCHEDULER_IDLE_TIME_OP, 227
 L4_SCHEDULER_INFO_OP, 227
 L4_SCHEDULER_RUN_THREAD_OP, 227
L4_SCHEDULER_IDLE_TIME_OP
 l4_scheduler_api, 227
L4_SCHEDULER_INFO_OP
 l4_scheduler_api, 227
L4_SCHEDULER_RUN_THREAD_OP
 l4_scheduler_api, 227
L4_SYSF_CALL
 l4_ipc_api, 195
L4_SYSF_NONE
 l4_ipc_api, 195
L4_SYSF_OPEN_WAIT
 l4_ipc_api, 195
L4_SYSF_RECV
 l4_ipc_api, 195
L4_SYSF_REPLY
 l4_ipc_api, 195
L4_SYSF_REPLY_AND_WAIT
 l4_ipc_api, 196
L4_SYSF_SEND
 l4_ipc_api, 195
L4_SYSF_SEND_AND_WAIT
 l4_ipc_api, 196
L4_SYSF_WAIT
 l4_ipc_api, 196
l4_task_api
 L4_FP_ALL_SPACES, 232
 L4_FP_DELETE_OBJ, 232
 L4_FP_OTHER_SPACES, 232
l4_thread_api
 L4_THREAD_CONTROL_ALIEN, 242
 L4_THREAD_CONTROL_BIND_TASK,
 242
 L4_THREAD_CONTROL_MR_IDX_-
 BIND_TASK, 242
 L4_THREAD_CONTROL_MR_IDX_-
 BIND_UTCB, 242
 L4_THREAD_CONTROL_MR_IDX_EXC_-
 HANDLER, 242
 L4_THREAD_CONTROL_MR_IDX_-
 FLAG_VALS, 242
 L4_THREAD_CONTROL_MR_IDX_-
 FLAGS, 242
L4_THREAD_CONTROL_MR_IDX_-
 PAGER, 242
L4_THREAD_CONTROL_OP, 241
L4_THREAD_CONTROL_SET_EXC_-
 HANDLER, 242
L4_THREAD_CONTROL_SET_PAGER, 242
L4_THREAD_CONTROL_UX_NATIVE,
 242
L4_THREAD_EX_REGS_CANCEL, 242
L4_THREAD_EX_REGS_OP, 241
L4_THREAD_EX_REGS_TRIGGER_-
 EXCEPTION, 242
L4_THREAD_GDT_X86_OP, 241
L4_THREAD MODIFY_SENDER_OP, 241
L4_THREAD_OPCODE_MASK, 241
L4_THREAD_REGISTER_DELETE_IRQ_-
 OP, 241
L4_THREAD_SET_FS_AMD64_OP, 241
L4_THREAD_STATS_OP, 241
L4_THREAD_SWITCH_OP, 241
L4_THREAD_VCPU_CONTROL_OP, 241
L4_THREAD_VCPU_RESUME_OP, 241
L4_THREAD_CONTROL_ALIEN
 l4_thread_api, 242
L4_THREAD_CONTROL_BIND_TASK
 l4_thread_api, 242
L4_THREAD_CONTROL_MR_IDX_BIND_-
 TASK
 l4_thread_api, 242
L4_THREAD_CONTROL_MR_IDX_BIND_-
 UTCB
 l4_thread_api, 242
L4_THREAD_CONTROL_MR_IDX_EXC_-
 HANDLER
 l4_thread_api, 242
L4_THREAD_CONTROL_MR_IDX_FLAG_-
 VALS
 l4_thread_api, 242
L4_THREAD_CONTROL_MR_IDX_FLAGS
 l4_thread_api, 242
L4_THREAD_CONTROL_MR_IDX_PAGER
 l4_thread_api, 242
L4_THREAD_CONTROL_OP
 l4_thread_api, 241
L4_THREAD_CONTROL_SET_EXC_-
 HANDLER
 l4_thread_api, 242
L4_THREAD_CONTROL_SET_PAGER
 l4_thread_api, 242
L4_THREAD_CONTROL_UX_NATIVE
 l4_thread_api, 242
L4_THREAD_EX_REGS_CANCEL
 l4_thread_api, 242
L4_THREAD_EX_REGS_OP

l4_thread_api, 241
 L4_THREAD_EX_REGS_TRIGGER_-
 EXCEPTION
 l4_thread_api, 242
 L4_THREAD_GDT_X86_OP
 l4_thread_api, 241
 L4_THREAD MODIFY_SENDER_OP
 l4_thread_api, 241
 L4_THREAD_OPCODE_MASK
 l4_thread_api, 241
 L4_THREAD_REGISTER_DELETE_IRQ_OP
 l4_thread_api, 241
 L4_THREAD_SET_FS_AMD64_OP
 l4_thread_api, 241
 L4_THREAD_STATS_OP
 l4_thread_api, 241
 L4_THREAD_SWITCH_OP
 l4_thread_api, 241
 L4_THREAD_VCPU_CONTROL_OP
 l4_thread_api, 241
 L4_THREAD_VCPU_RESUME_OP
 l4_thread_api, 241
 L4_TYPE_VHW_FRAMEBUFFER
 l4_kip_vhw_api, 290
 L4_TYPE_VHW_INPUT
 l4_kip_vhw_api, 290
 L4_TYPE_VHW_NET
 l4_kip_vhw_api, 290
 L4_TYPE_VHW_NONE
 l4_kip_vhw_api, 290
 l4_utcb_api_x86
 L4_UTCB_BUF_REGS_OFFSET, 295
 L4_UTCB_EXCEPTION_REGS_SIZE, 295
 L4_UTCB_GENERIC_BUFFERS_SIZE, 295
 L4_UTCB_GENERIC_DATA_SIZE, 295
 L4_UTCB_INHERIT_FPU, 296
 L4_UTCB_MSG_REGS_OFFSET, 295
 L4_UTCB_OFFSET, 296
 L4_UTCB_THREAD_REGS_OFFSET, 296
 l4_utcb_br_api
 L4_BDR_IO_SHIFT, 278
 L4_BDR_MEM_SHIFT, 278
 L4_BDR_OBJ_SHIFT, 278
 L4_UTCB_BUF_REGS_OFFSET
 l4_utcb_api_x86, 295
 L4_UTCB_EXCEPTION_REGS_SIZE
 l4_utcb_api_x86, 295
 L4_UTCB_GENERIC_BUFFERS_SIZE
 l4_utcb_api_x86, 295
 L4_UTCB_GENERIC_DATA_SIZE
 l4_utcb_api_x86, 295
 L4_UTCB_INHERIT_FPU
 l4_utcb_api_x86, 296
 L4_UTCB_MSG_REGS_OFFSET

l4_utcb_api_x86, 295
 L4_UTCB_OFFSET
 l4_utcb_api_x86, 296
 L4_UTCB_THREAD_REGS_OFFSET
 l4_utcb_api_x86, 296
 l4_vcon_api
 L4_VCON_ECHO, 283
 L4_VCON_GET_ATTR_OP, 284
 L4_VCON_ICANON, 283
 L4_VCON_ICRNL, 283
 L4_VCON_IGNCR, 283
 L4_VCON_INLCR, 283
 L4_VCON_OCRNL, 283
 L4_VCON_ONLCR, 283
 L4_VCON_ONLRET, 283
 L4_VCON_SET_ATTR_OP, 284
 L4_VCON_WRITE_OP, 284
 L4_VCON_WRITE_SIZE, 283
 L4_VCON_ECHO
 l4_vcon_api, 283
 L4_VCON_GET_ATTR_OP
 l4_vcon_api, 284
 L4_VCON_ICANON
 l4_vcon_api, 283
 L4_VCON_ICRNL
 l4_vcon_api, 283
 L4_VCON_IGNCR
 l4_vcon_api, 283
 L4_VCON_INLCR
 l4_vcon_api, 283
 L4_VCON_OCRNL
 l4_vcon_api, 283
 L4_VCON_ONLCR
 l4_vcon_api, 283
 L4_VCON_ONLRET
 l4_vcon_api, 283
 L4_VCON_SET_ATTR_OP
 l4_vcon_api, 284
 L4_VCON_WRITE_OP
 l4_vcon_api, 284
 L4_VCON_WRITE_SIZE
 l4_vcon_api, 283
 l4_vcpu_api
 L4_VCPU_F_DEBUG_EXC, 288
 L4_VCPU_F_EXCEPTIONS, 288
 L4_VCPU_F_FPU_ENABLED, 288
 L4_VCPU_F_IRQ, 288
 L4_VCPU_F_PAGE_FAULTS, 288
 L4_VCPU_F_USER_MODE, 288
 L4_VCPU_OFFSET_EXT_STATE, 289
 L4_VCPU_SF_IRQ_PENDING, 289
 L4_VCPU_F_DEBUG_EXC
 l4_vcpu_api, 288
 L4_VCPU_F_EXCEPTIONS

l4_vcpu_api, 288
L4_VCPU_F_FPU_ENABLED
 l4_vcpu_api, 288
L4_VCPU_F_IRQ
 l4_vcpu_api, 288
L4_VCPU_F_PAGEFAULTS
 l4_vcpu_api, 288
L4_VCPU_F_USER_MODE
 l4_vcpu_api, 288
L4_VCPU_OFFSET_EXT_STATE
 l4_vcpu_api, 289
L4_VCPU_SF_IRQ_PENDING
 l4_vcpu_api, 289
L4_WHOLE_ADDRESS_SPACE
 l4_fpage_api, 145
L4_WHOLE_IOADDRESS_SPACE
 l4_fpage_api, 146
l4_addr_consts_t
 l4_memory_api, 168
l4_basic_types
 l4_int16_t, 417
 l4_int32_t, 418
 l4_int64_t, 418
 l4_int8_t, 417
 l4_uint16_t, 418
 l4_uint32_t, 418
 l4_uint64_t, 418
 l4_uint8_t, 417
l4_buf_regs_t, 707
l4_buffer_desc_consts_t
 l4_utcb_br_api, 278
l4_busy_wait_ns
 l4util_tsc, 303
l4_busy_wait_us
 l4util_tsc, 303
l4_cache_api
 l4_cache_clean_data, 164
 l4_cache_coherent, 165
 l4_cache_dma_coherent, 165
 l4_cache_flush_data, 165
 l4_cache_inv_data, 165
l4_cache_clean_data
 l4_cache_api, 164
l4_cache_coherent
 l4_cache_api, 165
l4_cache_dma_coherent
 l4_cache_api, 165
l4_cache_flush_data
 l4_cache_api, 165
l4_cache_inv_data
 l4_cache_api, 165
l4_calibrate_tsc
 l4util_tsc, 304
l4_cap_api
 cap_cast, 270
 cap_dynamic_cast, 271
 cap_reinterpret_cast, 270
 l4_cap_consts_t, 269
 l4_cap_idx_t, 269
 l4_capability_equal, 272
 l4_default_caps_t, 269
 l4_is_invalid_cap, 272
 l4_is_valid_cap, 272
 l4_cap_consts_t
 l4_cap_api, 269
 L4_cap_fpage_rights
 l4_fpage_api, 145
 l4_cap_idx_t
 l4_cap_api, 269
 l4_capability_equal
 l4_cap_api, 272
 l4_debugger_api
 asm_enter_kdebug, 173, 174
 enter_kdebug, 173, 174
 kd_display, 173, 175
 ko, 174, 175
 l4_debugger_global_id, 176
 l4_debugger_kobj_to_id, 176
 l4_debugger_set_object_name, 175
 l4kd_inchar, 178
 outchar, 177
 outdec, 178
 outhex12, 178
 outhex16, 178
 outhex20, 178
 outhex32, 177
 outhex8, 178
 outnstring, 177
 outstring, 177
 l4_debugger_global_id
 l4_debugger_api, 176
 l4_debugger_kobj_to_id
 l4_debugger_api, 176
 l4_debugger_set_object_name
 l4_debugger_api, 175
L4_DECLARE_CONSTRUCTOR
 l4sys_defines, 123
l4_default_caps_t
 l4_cap_api, 269
L4_DISABLE_COPY
 l4_kernel_object_api, 217
l4_error
 l4_ipc_err_api, 206
l4_error_api
 l4_error_code_t, 179
l4_error_code_t
 l4_error_api, 179
l4_exc_regs_t, 707

flags, 710
 L4_EXPORT
 l4sysDefines, 124
 l4_factory_api
 l4_factory_create_factory, 183
 l4_factory_create_gate, 183
 l4_factory_create_irq, 184
 l4_factory_create_task, 181
 l4_factory_create_thread, 182
 l4_factory_create_vm, 185
 l4_factory_create_factory
 l4_factory_api, 183
 l4_factory_create_gate
 l4_factory_api, 183
 l4_factory_create_irq
 l4_factory_api, 184
 l4_factory_create_task
 l4_factory_api, 181
 l4_factory_create_thread
 l4_factory_api, 182
 l4_factory_create_vm
 l4_factory_api, 185
 l4_fpage
 l4_fpage_api, 146
 l4_fpage_all
 l4_fpage_api, 146
 l4_fpage_api
 L4_cap_fpage_rights, 145
 l4_fpage, 146
 l4_fpage_all, 146
 l4_fpage_cacheability_opt_t, 145
 l4_fpage_consts, 144
 l4_fpage_contains, 150
 l4_fpage_invalid, 146
 l4_fpage_max_order, 150
 l4_fpage_page, 149
 L4_fpage_rights, 145
 l4_fpage_rights, 148
 l4_fpage_set_rights, 149
 l4_fpage_size, 149
 l4_fpage_type, 148
 l4_iofpage, 147
 l4_is_fpage_writable, 148
 l4_obj_fpage, 147
 l4_fpage_cacheability_opt_t
 l4_fpage_api, 145
 l4_fpage_consts
 l4_fpage_api, 144
 l4_fpage_contains
 l4_fpage_api, 150
 l4_fpage_invalid
 l4_fpage_api, 146
 l4_fpage_max_order
 l4_fpage_api, 150
 l4_fpage_page
 l4_fpage_api, 149
 l4_fpage_rights
 l4_fpage_api, 145
 l4_fpage_set_rights
 l4_fpage_api, 148
 l4_fpage_size
 l4_fpage_api, 149
 l4_fpage_t, 710
 l4_fpage_type
 l4_fpage_api, 148
 l4_get_hz
 l4util_tsc, 306
 L4_HIDDEN
 l4sysDefines, 124
 l4_icu_api
 l4_icu_bind, 188
 L4_icu_flags, 188
 l4_icu_info, 190
 l4_icu_info_t, 188
 l4_icu_mask, 192
 l4_icu_msi_info, 191
 l4_icu_set_mode, 190
 l4_icu_unbind, 189
 l4_icu_unmask, 191
 l4_icu_bind
 l4_icu_api, 188
 L4_icu_flags
 l4_icu_api, 188
 l4_icu_info
 l4_icu_api, 190
 l4_icu_info_t, 711
 features, 712
 l4_icu_api, 188
 l4_icu_mask
 l4_icu_api, 192
 l4_icu_msi_info
 l4_icu_api, 191
 l4_icu_set_mode
 l4_icu_api, 190
 l4_icu_unbind
 l4_icu_api, 189
 l4_icu_unmask
 l4_icu_api, 191
 l4_int16_t
 l4_basic_types, 417
 l4_int32_t
 l4_basic_types, 418
 l4_int64_t
 l4_basic_types, 418
 l4_int8_t
 l4_basic_types, 417

l4_iofpage
 l4_fpage_api, 147

l4_ipc
 l4_ipc_api, 201

l4_ipc_api
 l4_ipc, 201

l4_ipc_call, 199

l4_ipc_receive, 198

l4_ipc_reply_and_wait, 200

l4_ipc_send, 196

l4_ipc_send_and_wait, 201

l4_ipc_sleep, 202

l4_ipc_wait, 197

l4_sndfpage_add, 203

l4_syscall_flags_t, 195

l4_ipc_call
 l4_ipc_api, 199

l4_ipc_err_api
 l4_error, 206

l4_ipc_error, 206

l4_ipc_error_code, 208

l4_ipc_is_rcv_error, 208

l4_ipc_is_snd_error, 207

l4_ipc_tcr_error_t, 205

l4_ipc_error
 l4_ipc_err_api, 206

l4_ipc_error_code
 l4_ipc_err_api, 208

l4_ipc_gate_bind_thread
 l4_kernel_object_gate_api, 120

l4_ipc_gate_get_infos
 l4_kernel_object_gate_api, 121

l4_ipc_gate_ops
 l4_kernel_object_gate_api, 120

l4_ipc_is_rcv_error
 l4_ipc_err_api, 208

l4_ipc_is_snd_error
 l4_ipc_err_api, 207

l4_ipc_receive
 l4_ipc_api, 198

l4_ipc_reply_and_wait
 l4_ipc_api, 200

l4_ipc_send
 l4_ipc_api, 196

l4_ipc_send_and_wait
 l4_ipc_api, 201

l4_ipc_sleep
 l4_ipc_api, 202

l4_ipc_tcr_error_t
 l4_ipc_err_api, 205

l4_ipc_timeout
 l4_timeout_api, 157

L4_IPC_TIMEOUT_0
 l4_timeout_api, 156

l4_ipc_wait
 l4_ipc_api, 197

l4_irq_api
 l4_irq_attach, 211

l4_irq_chain, 211

l4_irq_detach, 212

L4_irq_flow_type, 210

l4_irq_receive, 213

l4_irq_trigger, 213

l4_irq_unmask, 215

l4_irq_wait, 214

l4_irq_attach
 l4_irq_api, 211

l4_irq_chain
 l4_irq_api, 211

l4_irq_detach
 l4_irq_api, 212

L4_irq_flow_type
 l4_irq_api, 210

l4_irq_receive
 l4_irq_api, 213

l4_irq_trigger
 l4_irq_api, 213

l4_irq_unmask
 l4_irq_api, 215

l4_irq_wait
 l4_irq_api, 214

l4_is_fpage_writable
 l4_fpage_api, 148

l4_is_invalid_cap
 l4_cap_api, 272

l4_is_valid_cap
 l4_cap_api, 272

l4_kernel_info_get_mem_desc_end
 l4_kip_memdesc_api, 224

l4_kernel_info_get_mem_desc_is_virtual
 l4_kip_memdesc_api, 225

l4_kernel_info_get_mem_desc_start
 l4_kip_memdesc_api, 224

l4_kernel_info_get_mem_desc_subtype
 l4_kip_memdesc_api, 225

l4_kernel_info_get_mem_desc_type
 l4_kip_memdesc_api, 225

l4_kernel_info_get_num_mem_descs
 l4_kip_memdesc_api, 224

l4_kernel_info_mem_desc_t, 712
 l4_kip_memdesc_api, 223

l4_kernel_info_set_mem_desc
 l4_kip_memdesc_api, 224

l4_kernel_info_t, 712
 l4_kip_api, 221

l4_kernel_info_version_offset
 l4_kip_api, 221

l4_kernel_object_api
 L4_DISABLE_COPY, 217

L4_KOBJECT, 218
 L4_KOBJECT_DISABLE_COPY, 218
 l4_kernel_object_gate_api
 l4_ipc_gate_bind_thread, 120
 l4_ipc_gate_get_infos, 121
 L4_ipc_gate_ops, 120
 l4_kip_api
 l4_kernel_info_version_offset, 221
 l4_kip_version, 220
 l4_kip_version_string, 220
 l4_kip_memdesc_api
 l4_kernel_info_get_mem_desc_end, 224
 l4_kernel_info_get_mem_desc_is_virtual, 225
 l4_kernel_info_get_mem_desc_start, 224
 l4_kernel_info_get_mem_desc_subtype, 225
 l4_kernel_info_get_mem_desc_type, 225
 l4_kernel_info_get_num_mem_descs, 224
 l4_kernel_info_mem_desc_t, 223
 l4_kernel_info_set_mem_desc, 224
 l4_mem_type_t, 223
 l4_kip_version
 l4_kip_api, 220
 l4_kip_version_string
 l4_kip_api, 220
 l4_kip_vhw_api
 l4_vhw_entry_type, 290
 L4_KOBJECT
 l4_kernel_object_api, 218
 L4_KOBJECT_DISABLE_COPY
 l4_kernel_object_api, 218
 L4_LOG2_PAGESIZE
 l4_memory_api, 167
 L4_LOG2_SUPERPAGESIZE
 l4_memory_api, 168
 l4_map_control
 l4_msgitem_api, 153
 l4_map_obj_control
 l4_msgitem_api, 153
 l4_mem_op_api
 L4_mem_op_widths, 291
 l4_mem_read, 291
 l4_mem_write, 291
 L4_mem_op_widths
 l4_mem_op_api, 291
 l4_mem_read
 l4_mem_op_api, 291
 l4_mem_type_t
 l4_kip_memdesc_api, 223
 l4_mem_write
 l4_mem_op_api, 291
 l4_memory_api
 l4_addr_consts_t, 168
 L4_LOG2_PAGESIZE, 167
 L4_LOG2_SUPERPAGESIZE, 168
 L4_PAGEMASK, 167
 l4_round_page, 169
 l4_round_size, 170
 L4_SUPERPAGEMASK, 168
 L4_SUPERPAGESIZE, 168
 l4_trunc_page, 168
 l4_trunc_size, 169
 l4_msgitem_consts_t
 l4_msgitem_api, 152
 l4_msgitem_regs_t, 715
 l4_msgitem_api
 l4_map_control, 153
 l4_map_obj_control, 153
 l4_msgitem_consts_t, 152
 l4_mshtag
 l4_mshtag_api, 260
 l4_mshtag_api
 l4_mshtag, 260
 l4_mshtag_flags, 260, 263
 l4_mshtag_has_error, 263
 l4_mshtag_is_exception, 265
 l4_mshtag_is_io_page_fault, 266
 l4_mshtag_is_page_fault, 264
 l4_mshtag_is_preemption, 264
 l4_mshtag_is_sigma0, 266
 l4_mshtag_is_sys_exception, 265
 l4_mshtag_items, 263
 l4_mshtag_label, 261
 l4_mshtag_protocol, 259
 l4_mshtag_t, 259
 l4_mshtag_words, 262
 l4_mshtag_flags
 l4_mshtag_api, 260, 263
 l4_mshtag_has_error
 l4_mshtag_api, 263
 l4_mshtag_is_exception
 l4_mshtag_api, 265
 l4_mshtag_is_io_page_fault
 l4_mshtag_api, 266
 l4_mshtag_is_page_fault
 l4_mshtag_api, 264
 l4_mshtag_is_preemption
 l4_mshtag_api, 264
 l4_mshtag_is_sigma0
 l4_mshtag_api, 266
 l4_mshtag_is_sys_exception
 l4_mshtag_api, 265
 l4_mshtag_items
 l4_mshtag_api, 263
 l4_mshtag_label
 l4_mshtag_api, 261
 l4_mshtag_protocol
 l4_mshtag_api, 259
 l4_mshtag_t, 715

flags, 717
l4_msntag_api, 259
l4_msntag_words
 l4_msntag_api, 262
l4_next_period_id
 api_calls_rt_sched, 140
L4_NOTHROW
 l4sys_defines, 124
l4_ns_to_tsc
 l4util_tsc, 302
l4_obj_fpage
 l4_fpage_api, 147
L4_PAGESIZE
 l4_memory_api, 167
l4_preemption_id
 api_calls_rt_sched, 140
l4_rcv_timeout
 l4_timeout_api, 158
l4_rdpmc
 l4util_tsc, 301
l4_rdpmc_32
 l4util_tsc, 301
l4_rdtsc
 l4util_tsc, 300
l4_rdtsc_32
 l4util_tsc, 301
l4_round_page
 l4_memory_api, 169
l4_round_size
 l4_memory_api, 170
l4_rt_begin_minimal_periodic
 api_calls_rt_sched, 136
l4_rt_begin_strictly_periodic
 api_calls_rt_sched, 135
l4_rt_end_periodic
 api_calls_rt_sched, 137
l4_rt_generic
 api_calls_rt_sched, 141
l4_rt_next_period
 api_calls_rt_sched, 139
l4_rt_next_reservation
 api_calls_rt_sched, 139
l4_rt_preemption_t, 717
l4_rt_preemption_val32_t, 718
l4_rt_preemption_val_t, 719
l4_rt_remove
 api_calls_rt_sched, 137
l4_rt_set_period
 api_calls_rt_sched, 138
l4_sched_cpu_set
 l4_scheduler_api, 227
l4_sched_cpu_set_t, 720
l4_sched_param_t, 720
l4_scheduler_api
 l4_sched_cpu_set, 227
 l4_scheduler_idle_time, 229
 l4_scheduler_info, 228
 l4_scheduler_is_online, 230
 L4_scheduler_ops, 227
 l4_scheduler_run_thread, 229
 l4_scheduler_idle_time
 l4_scheduler_api, 229
 l4_scheduler_info
 l4_scheduler_api, 228
 l4_scheduler_is_online
 l4_scheduler_api, 230
 L4_scheduler_ops
 l4_scheduler_api, 227
 l4_scheduler_run_thread
 l4_scheduler_api, 229
 l4_sleep_forever
 l4util_api, 355
 l4_snd_fpage_t, 722
 l4_snd_timeout
 l4_timeout_api, 158
 l4_sndfpage_add
 l4_ipc_api, 203
L4_SUPERPAGESIZE
 l4_memory_api, 168
L4_SUPERPAGESIZE
 l4_memory_api, 168
l4_syscall_flags_t
 l4_ipc_api, 195
l4_task_add_ku_mem
 l4_task_api, 237
l4_task_api
 l4_task_add_ku_mem, 237
 l4_task_cap_equal, 237
 l4_task_cap_has_child, 236
 l4_task_cap_valid, 236
 l4_task_delete_obj, 234
 l4_task_map, 233
 l4_task_release_cap, 235
 l4_task_unmap, 233
 l4_task_unmap_batch, 234
 l4_unmap_flags_t, 232
l4_task_cap_equal
 l4_task_api, 237
l4_task_cap_has_child
 l4_task_api, 236
l4_task_cap_valid
 l4_task_api, 236
l4_task_delete_obj
 l4_task_api, 234
l4_task_map
 l4_task_api, 233
l4_task_release_cap
 l4_task_api, 235

l4_task_unmap
 l4_task_api, 233
 l4_task_unmap_batch
 l4_task_api, 234
 l4_thread_api
 L4_thread_control_flags, 241
 L4_thread_control_mr_indices, 242
 l4_thread_ex_regs, 242
 L4_thread_ex_regs_flags, 242
 l4_thread_ex_regs_ret, 243
 l4_thread_modify_sender_add, 249
 l4_thread_modify_sender_commit, 250
 l4_thread_modify_sender_start, 249
 L4_thread_ops, 241
 l4_thread_register_del_irq, 248
 l4_thread_stats_time, 245
 l4_thread_switch, 244
 l4_thread_vcpu_control, 247
 l4_thread_vcpu_control_ext, 247
 l4_thread_vcpu_resume_commit, 246
 l4_thread_vcpu_resume_start, 245
 l4_thread_yield, 244
 l4_thread_control_alien
 l4_thread_control_api, 255
 l4_thread_control_api
 l4_thread_control_alien, 255
 l4_thread_control_bind, 254
 l4_thread_control_commit, 256
 l4_thread_control_exc_handler, 253
 l4_thread_control_pager, 253
 l4_thread_control_start, 252
 l4_thread_control_ux_host_syscall, 255
 l4_thread_control_bind
 l4_thread_control_api, 254
 l4_thread_control_commit
 l4_thread_control_api, 256
 l4_thread_control_exc_handler
 l4_thread_control_api, 253
 L4_thread_control_flags
 l4_thread_api, 241
 L4_thread_control_mr_indices
 l4_thread_api, 242
 l4_thread_control_pager
 l4_thread_control_api, 253
 l4_thread_control_start
 l4_thread_control_api, 252
 l4_thread_control_ux_host_syscall
 l4_thread_control_api, 255
 l4_thread_ex_regs
 l4_thread_api, 242
 L4_thread_ex_regs_flags
 l4_thread_api, 242
 l4_thread_ex_regs_ret
 l4_thread_api, 243
 l4_thread_modify_sender_add
 l4_thread_api, 249
 l4_thread_modify_sender_commit
 l4_thread_api, 250
 l4_thread_modify_sender_start
 l4_thread_api, 249
 L4_thread_ops
 l4_thread_api, 241
 l4_thread_register_del_irq
 l4_thread_api, 248
 l4_thread_regs_t, 723
 l4_thread_stats_time
 l4_thread_api, 245
 l4_thread_switch
 l4_thread_api, 244
 l4_thread_vcpu_control
 l4_thread_api, 247
 l4_thread_vcpu_control_ext
 l4_thread_api, 247
 l4_thread_vcpu_resume_commit
 l4_thread_api, 246
 l4_thread_vcpu_resume_start
 l4_thread_api, 245
 l4_thread_yield
 l4_thread_api, 244
 l4_timeout
 l4_timeout_api, 157
 l4_timeout_abs
 l4_timeout_api, 160
 l4_timeout_abs_validity
 l4_timeout_api, 157
 l4_timeout_api
 l4_ipc_timeout, 157
 L4_IPC_TIMEOUT_0, 156
 l4_rcv_timeout, 158
 l4_snd_timeout, 158
 l4_timeout, 157
 l4_timeout_abs, 160
 l4_timeout_abs_validity, 157
 l4_timeout_get, 159
 l4_timeout_is_absolute, 159
 l4_timeout_rel, 157
 l4_timeout_rel_get, 158
 l4_timeout_s, 156
 l4_timeout_t, 156
 l4_timeout_get
 l4_timeout_api, 159
 l4_timeout_is_absolute
 l4_timeout_api, 159
 l4_timeout_rel
 l4_timeout_api, 157
 l4_timeout_rel_get
 l4_timeout_api, 158
 l4_timeout_s, 724

l4_timeout_api, 156
l4_timeout_t, 724
 l4_timeout_api, 156
l4_tracebuffer_status_t, 726
 cnt_jobmap_tlb_flush, 730
 size0, 730
 size1, 730
 tracebuffer0, 730
 tracebuffer1, 730
 version0, 730
 version1, 730
l4_tracebuffer_status_window_t, 731
l4_trunc_page
 l4_memory_api, 168
l4_trunc_size
 l4_memory_api, 169
l4_tsc_init
 l4util_tsc, 304
l4_tsc_to_ns
 l4util_tsc, 301
l4_tsc_to_s_and_ns
 l4util_tsc, 302
l4_tsc_to_us
 l4util_tsc, 302
l4_uint16_t
 l4_basic_types, 418
l4_uint32_t
 l4_basic_types, 418
l4_uint64_t
 l4_basic_types, 418
l4_uint8_t
 l4_basic_types, 417
l4_unmap_flags_t
 l4_task_api, 232
l4_utcb_api
 l4_utcb_br, 275
 l4_utcb_mr, 275
 l4_utcb_t, 275
 l4_utcb_tcr, 276
l4_utcb_api_x86
 L4_utcb_consts_x86, 295
l4_utcb_br
 l4_utcb_api, 275
l4_utcb_br_api
 l4_buffer_desc_consts_t, 278
L4_utcb_consts_x86
 l4_utcb_api_x86, 295
l4_utcb_exc
 l4_utcb_exc_api, 280
l4_utcb_exc_api
 l4_utcb_exc, 280
 l4_utcb_exc_is_pf, 281
 l4_utcb_exc_pc, 280
 l4_utcb_exc_pc_set, 280
l4_utcb_exc_is_pf
 l4_utcb_exc_api, 281
l4_utcb_exc_pc
 l4_utcb_exc_api, 280
l4_utcb_exc_pc_set
 l4_utcb_exc_api, 280
l4_utcb_mr
 l4_utcb_api, 275
l4_utcb_t
 l4_utcb_api, 275
l4_utcb_tcr
 l4_utcb_api, 276
l4_vcon_api
 l4_vcon_get_attr, 286
 L4_vcon_i_flags, 283
 L4_vcon_l_flags, 283
 L4_vcon_o_flags, 283
 L4_vcon_ops, 283
 l4_vcon_read, 285
 l4_vcon_send, 284
 l4_vcon_set_attr, 285
 l4_vcon_write, 284
 L4_vcon_write_consts, 283
l4_vcon_attr_t, 731
l4_vcon_get_attr
 l4_vcon_api, 286
 L4_vcon_i_flags
 l4_vcon_api, 283
 L4_vcon_l_flags
 l4_vcon_api, 283
 L4_vcon_o_flags
 l4_vcon_api, 283
 L4_vcon_ops
 l4_vcon_api, 283
 l4_vcon_read
 l4_vcon_api, 285
 l4_vcon_send
 l4_vcon_api, 284
 l4_vcon_set_attr
 l4_vcon_api, 285
 l4_vcon_write
 l4_vcon_api, 284
 L4_vcon_write_consts
 l4_vcon_api, 283
l4_vcpu_api
 L4_vcpu_state_flags, 288
 L4_vcpu_state_offset, 289
 L4_vcpu_sticky_flags, 288
l4_vcpu_ipc_regs_t, 732
l4_vcpu_regs_t, 734
 ax, 736
 bp, 736
 bx, 736
 cx, 736

di, [736](#)
 dx, [736](#)
 si, [736](#)
L4_vcpu_state_flags
 l4_vcpu_api, [288](#)
L4_vcpu_state_offset
 l4_vcpu_api, [289](#)
l4_vcpu_state_t, [737](#)
L4_vcpu_sticky_flags
 l4_vcpu_api, [288](#)
l4_vhw_descriptor, [740](#)
 count, [742](#)
 descs, [742](#)
 magic, [742](#)
 version, [742](#)
l4_vhw_entry, [743](#)
 fd, [744](#)
 irq_no, [744](#)
 mem_size, [744](#)
 mem_start, [744](#)
 provider_pid, [744](#)
 type, [744](#)
l4_vhw_entry_type
 l4_kip_vhw_api, [290](#)
l4_vm_run
 l4_vm_tz_api, [293](#)
l4_vm_state, [745](#)
l4_vm_svm_vmcb_control_area, [745](#)
l4_vm_svm_vmcb_state_save_area, [745](#)
l4_vm_svm_vmcb_state_save_area_seg, [747](#)
l4_vm_svm_vmcb_t, [747](#)
l4_vm_tz_api
 l4_vm_run, [293](#)
l4_vm_vmx_api
 l4_vm_vmx_field_len, [162](#)
 l4_vm_vmx_field_ptr, [163](#)
l4_vm_vmx_field_len
 l4_vm_vmx_api, [162](#)
l4_vm_vmx_field_ptr
 l4_vm_vmx_api, [163](#)
L4IO_DEVICE_ANY
 api_l4io, [371](#)
L4IO_DEVICE_INVALID
 api_l4io, [371](#)
L4IO_DEVICE_OTHER
 api_l4io, [371](#)
L4IO_DEVICE_PCI
 api_l4io, [371](#)
L4IO_DEVICE_USB
 api_l4io, [371](#)
L4IO_MEM_CACHED
 api_l4io, [370](#)
L4IO_MEM_EAGER_MAP
 api_l4io, [370](#)
L4IO_MEM_NONCACHED
 api_l4io, [370](#)
L4IO_MEM_USE_MTRR
 api_l4io, [370](#)
L4IO_MEM_USE_RESERVED_AREA
 api_l4io, [370](#)
L4IO_RESOURCE_ANY
 api_l4io, [371](#)
L4IO_RESOURCE_INVALID
 api_l4io, [371](#)
L4IO_RESOURCE_IRQ
 api_l4io, [371](#)
L4IO_RESOURCE_MEM
 api_l4io, [371](#)
L4IO_RESOURCE_PORT
 api_l4io, [371](#)
l4io_device_types_t
 api_l4io, [370](#)
l4io_has_resource
 api_l4io, [374](#)
l4io_iomem_flags_t
 api_l4io, [370](#)
l4io_lookup_device
 api_l4io, [373](#)
l4io_lookup_resource
 api_l4io, [373](#)
l4io_release_iomem
 api_l4io, [372](#)
l4io_release_ioport
 api_l4io, [373](#)
l4io_request_iomem
 api_l4io, [371](#)
l4io_request_iomem_region
 api_l4io, [371](#)
l4io_request_ioport
 api_l4io, [372](#)
l4io_request_resource_iomem
 api_l4io, [374](#)
l4io_resource_t
 api_l4io, [370](#)
l4io_resource_types_t
 api_l4io, [371](#)
l4io_search_iomem_region
 api_l4io, [372](#)
l4irq_api_async
 l4irq_release, [380](#)
 l4irq_request, [380](#)
l4irq_api_async_cap
 l4irq_request_cap, [383](#)
l4irq_api_irq
 l4irq_attach, [376](#)
 l4irq_attach_ft, [376](#)
 l4irq_attach_thread, [377](#)
 l4irq_attach_thread_ft, [377](#)

l4irq_detach, 378
l4irq_unmask, 378
l4irq_unmask_and_wait_any, 378
l4irq_wait, 377
l4irq_wait_any, 378
l4irq_api_irq_cap
 l4irq_attach_cap, 381
 l4irq_attach_cap_ft, 381
 l4irq_attach_thread_cap, 381
 l4irq_attach_thread_cap_ft, 382
l4irq_attach
 l4irq_api_irq, 376
l4irq_attach_cap
 l4irq_api_irq_cap, 381
l4irq_attach_cap_ft
 l4irq_api_irq_cap, 381
l4irq_attach_ft
 l4irq_api_irq, 376
l4irq_attach_thread
 l4irq_api_irq, 377
l4irq_attach_thread_cap
 l4irq_api_irq_cap, 381
l4irq_attach_thread_cap_ft
 l4irq_api_irq_cap, 382
l4irq_attach_thread_ft
 l4irq_api_irq, 377
l4irq_detach
 l4irq_api_irq, 378
l4irq_release
 l4irq_api_async, 380
l4irq_request
 l4irq_api_async, 380
l4irq_request_cap
 l4irq_api_async_cap, 383
l4irq_unmask
 l4irq_api_irq, 378
l4irq_unmask_and_wait_any
 l4irq_api_irq, 378
l4irq_wait
 l4irq_api_irq, 377
l4irq_wait_any
 l4irq_api_irq, 378
l4kd_inchar
 l4_debugger_api, 178
L4Re, 426
L4Re C Interface, 112
L4Re C++ Interface, 56
L4Re Capability API, 109
L4Re ELF Auxiliary Information, 95
L4Re Protocol identifiers, 107
L4Re Util C Interface, 114
L4Re Util C++ Interface, 59
L4Re::Cap_alloc, 513
 alloc, 514, 515
 free, 515
 get_cap_alloc, 516
L4Re::Console, 534
L4Re::Dataspace, 537
 allocate, 543
 clear, 543
 copy_in, 544
 flags, 546
 info, 546
 map, 541
 Map_ro, 541
 Map_rw, 541
 Map_flags, 541
 map_region, 542
 phys, 544
 size, 545
L4Re::Dataspace::Stats, 882
L4Re::Debug_obj, 551
 debug, 554
L4Re::Env, 583
 env, 587
 factory, 588, 591
 first_free_cap, 588, 592
 first_free_utcb, 589, 592
 get, 589
 get_cap, 590
 initial_caps, 589, 593
 log, 588, 591
 main_thread, 588, 591
 mem_alloc, 587, 591
 parent, 587, 590
 rm, 587, 591
 scheduler, 592
 task, 588
 utcb_area, 589, 592
L4Re::Event, 593
 get_buffer, 596
L4Re::Event_buffer_t, 601
 Event_buffer_t, 604
 next, 604
 put, 604
L4Re::Event_buffer_t::Event, 596
L4Re::Log, 779
 print, 782
 printfn, 782
L4Re::Mem_alloc
 Continuous, 786
 Pinned, 786
 Super_pages, 786
L4Re::Mem_alloc, 783
 alloc, 786
 free, 787
 Mem_alloc_flags, 786
L4Re::Namespace, 801

query, 804, 805
 Register_flags, 804
 register_obj, 805
 Ro, 804
 Rw, 804
 Strong, 804
 L4Re::Parent, 821
 signal, 824
 L4Re::Rm, 836
 attach, 843, 844
 Attach_flags, 841
 Attach_flags, 840
 detach, 844, 845
 Detach_again, 840
 Detach_exact, 841
 Detach_free, 840
 Detach_keep, 841
 Detach_overlap, 841
 Detach_flags, 841
 Detach_result, 840
 Detached_ds, 840
 Eager_map, 841
 find, 846
 free_area, 843
 In_area, 841
 Kept_ds, 840
 Pager, 840
 Read_only, 840
 Region_flags, 840
 Region_flags, 840
 reserve_area, 841, 842
 Reserved, 840
 Search_addr, 841
 Split_ds, 840
 L4Re::Smart_cap_auto, 875
 L4Re::Util::Auto_cap, 433
 L4Re::Util::Auto_del_cap, 434
 L4Re::Util::Cap_alloc_base, 517
 L4Re::Util::Counting_cap_alloc, 536
 L4Re::Util::Dataspace_svr, 548
 clear, 550
 copy, 550
 map, 548
 map_hook, 549
 page_shift, 550
 phys, 549
 release, 550
 take, 549
 L4Re::Util::Event_t
 Mode_irq, 611
 Mode_polling, 611
 L4Re::Util::Event_buffer_consumer_t, 597
 foreach_available_event, 600
 process, 601
 L4Re::Util::Event_buffer_t, 605
 attach, 608
 buf, 608
 detach, 609
 L4Re::Util::Event_t, 609
 buffer, 611
 init, 611
 irq, 612
 Mode, 611
 L4Re::Util::Item_alloc_base, 693
 L4Re::Util::Names::Name, 801
 L4Re::Util::Ref_cap, 831
 L4Re::Util::Ref_del_cap, 831
 L4Re::Util::Smart_cap_auto, 877
 L4Re::Util::Smart_count_cap, 877
 L4Re::Util::Vcon_svr, 916
 dispatch, 916
 L4Re::Util::Video::Goos_svr, 642
 dispatch, 645
 get_fb, 644
 init_infos, 646
 refresh, 645
 screen_info, 645
 view_info, 645
 L4Re::Vfs, 427
 L4Re::Vfs::Be_file, 475
 L4Re::Vfs::Be_file_system, 479
 ~Be_file_system, 482
 Be_file_system, 482
 type, 482
 L4Re::Vfs::Directory, 563
 faccessat, 565
 link, 566
 mkdir, 565
 rename, 566
 rmdir, 567
 symlink, 567
 unlink, 566
 L4Re::Vfs::File, 623
 L4Re::Vfs::File_system, 625
 mount, 628
 type, 628
 L4Re::Vfs::Fs, 629
 alloc_fd, 630
 free_fd, 631
 get_file, 630
 mount, 631
 set_fd, 631
 L4Re::Vfs::Generic_file, 632
 fchmod, 635
 fstat64, 634
 get_status_flags, 635
 set_status_flags, 635
 unlock_all_locks, 634

L4Re::Vfs::Mman, 797
L4Re::Vfs::Ops, 808
L4Re::Vfs::Regular_file, 832
 data_space, 834
 fdatasync, 835
 fsync, 835
 ftruncate64, 835
 get_lock, 836
 lseek64, 835
 readv, 834
 set_lock, 836
 writev, 835
L4Re::Vfs::Special_file, 878
 ioctl, 880
L4Re::Video::Color_component, 527
 Color_component, 528
 dump, 529
 get, 529
 operator==, 528
 set, 529
 shift, 528
 size, 528
L4Re::Video::Goos, 636
 create_buffer, 640
 create_view, 641
 delete_buffer, 640
 delete_view, 641
 F_auto_refresh, 639
 F_dynamic_buffers, 640
 F_dynamic_views, 639
 F_pointer, 639
 Flags, 639
 get_static_buffer, 640
 info, 640
 view, 641
L4Re::Video::Goos::Info, 655
 auto_refresh, 657
L4Re::Video::Pixel_info, 824
 a, 828, 829
 b, 828, 829
 bits_per_pixel, 828
 bytes_per_pixel, 828, 830
 dump, 830
 g, 828, 829
 has_alpha, 829
 operator==, 830
 Pixel_info, 827
 r, 828, 829
L4Re::Video::View, 926
 F_above, 929
 F_dyn_allocated, 929
 F_flags_mask, 929
 F_fully_dynamic, 929
 F_none, 929
 F_set_background, 929
 F_set_buffer, 929
 F_set_buffer_offset, 929
 F_set_bytes_per_line, 929
 F_set_flags, 929
 F_set_pixel, 929
 F_set_position, 929
 Flags, 929
 info, 929
 refresh, 930
 set_info, 930
 set_viewport, 930
 stack, 930
 V_flags, 929
L4Re::Video::View::Info, 658
 flags, 661
L4RE_ELF_AUX_T_KIP_ADDR
 api_l4re_elf_aux, 98
L4RE_ELF_AUX_T_NONE
 api_l4re_elf_aux, 98
L4RE_ELF_AUX_T_STACK_ADDR
 api_l4re_elf_aux, 98
L4RE_ELF_AUX_T_STACK_SIZE
 api_l4re_elf_aux, 98
L4RE_ELF_AUX_T_VMA
 api_l4re_elf_aux, 98
L4RE_RM_ATTACH_FLAGS
 api_l4re_c_rm, 76
L4RE_RM_EAGER_MAP
 api_l4re_c_rm, 76
L4RE_RM_IN_AREA
 api_l4re_c_rm, 76
L4RE_RM_NO_ALIAS
 api_l4re_c_rm, 76
L4RE_RM_OVERMAP
 api_l4re_c_rm, 76
L4RE_RM_PAGER
 api_l4re_c_rm, 76
L4RE_RM_READ_ONLY
 api_l4re_c_rm, 76
L4RE_RM_REGION_FLAGS
 api_l4re_c_rm, 76
L4RE_RM_RESERVED
 api_l4re_c_rm, 76
L4RE_RM_SEARCH_ADDR
 api_l4re_c_rm, 76
l4re_aux_t, 749
l4re_cap_api
 cap_alloc, 110
l4re_debug_obj_debug
 api_l4re_c_debug, 63
l4re_ds_allocate
 api_l4re_c_ds, 61
l4re_ds_clear

l4re_ma_alloc
 api_l4re_c_mem_alloc, 70
 l4re_ma_alloc_srv
 api_l4re_c_mem_alloc, 71
 l4re_ma_flags
 api_l4re_c_mem_alloc, 70
 l4re_ma_free
 api_l4re_c_mem_alloc, 71
 l4re_ma_free_srv
 api_l4re_c_mem_alloc, 72
 l4re_ns_query_to_srv
 api_l4re_c_ns, 74
 l4re_ns_register_flags
 api_l4re_c_ns, 74
 l4re_ns_register_obj_srv
 api_l4re_c_ns, 74
 l4re_rm_attach
 api_l4re_c_rm, 77
 l4re_rm_attach_srv
 api_l4re_c_rm, 83
 l4re_rm_detach
 api_l4re_c_rm, 78
 l4re_rm_detach_ds
 api_l4re_c_rm, 79
 l4re_rm_detach_ds_unmap
 api_l4re_c_rm, 80
 l4re_rm_detach_srv
 api_l4re_c_rm, 84
 l4re_rm_detach_unmap
 api_l4re_c_rm, 80
 l4re_rm_find
 api_l4re_c_rm, 81
 l4re_rm_find_srv
 api_l4re_c_rm, 84
 l4re_rm_flags_t
 api_l4re_c_rm, 76
 l4re_rm_free_area
 api_l4re_c_rm, 76
 l4re_rm_free_area_srv
 api_l4re_c_rm, 83
 l4re_rm_reserve_area
 api_l4re_c_rm, 76
 l4re_rm_reserve_area_srv
 api_l4re_c_rm, 82
 l4re_rm_show_lists
 api_l4re_c_rm, 82
 l4re_util_cap_last
 api_l4re_c_util_cap, 85
 l4re_util_kumem
 kumem_alloc, 111
 l4re_util_kumem_alloc
 api_l4re_c_util_kumem_alloc, 86
 l4re_video_color_component_t, 755
 l4re_video_goops_create_buffer

api_l4re_c_video, 90
l4re_video_goops_create_view
 api_l4re_c_video, 91
l4re_video_goops_delete_buffer
 api_l4re_c_video, 91
l4re_video_goops_delete_view
 api_l4re_c_video, 91
l4re_video_goops_get_static_buffer
 api_l4re_c_video, 91
l4re_video_goops_get_view
 api_l4re_c_video, 92
l4re_video_goops_info
 api_l4re_c_video, 90
l4re_video_goops_info_flags_t
 api_l4re_c_video, 89
l4re_video_goops_info_t, 755
l4re_video_goops_refresh
 api_l4re_c_video, 90
l4re_video_pixel_info_t, 757
l4re_video_view_get_info
 api_l4re_c_video, 92
l4re_video_view_info_flags_t
 api_l4re_c_video, 89
l4re_video_view_info_t, 759
l4re_video_view_refresh
 api_l4re_c_video, 92
l4re_video_view_set_info
 api_l4re_c_video, 92
l4re_video_view_set_viewport
 api_l4re_c_video, 93
l4re_video_view_stack
 api_l4re_c_video, 93
l4re_video_view_t, 761
 api_l4re_c_video, 89
l4shmc_add_chunk
 api_l4shmc_chunk, 402
l4shmc_add_signal
 api_l4shmc_signal, 410
l4shmc_area_overhead
 api_l4shm, 400
l4shmc_area_size
 api_l4shm, 400
l4shmc_area_size_free
 api_l4shm, 400
l4shmc_attach
 api_l4shm, 399
l4shmc_attach_signal
 api_l4shmc_signal, 410
l4shmc_attach_signal_to
 api_l4shmc_signal, 410
l4shmc_attach_to
 api_l4shm, 399
l4shmc_check_magic
 api_l4shmc_signal, 411
l4shmc_chunk_capacity
 api_l4shmc_chunk, 403
l4shmc_chunk_consumed
 api_l4shmc_chunk_cons, 408
l4shmc_chunk_overhead
 api_l4shm, 400
l4shmc_chunk_ptr
 api_l4shmc_chunk, 403
l4shmc_chunk_ready
 api_l4shmc_chunk_prod, 405
l4shmc_chunk_ready_sig
 api_l4shmc_chunk_prod, 405
l4shmc_chunk_signal
 api_l4shmc_chunk, 404
l4shmc_chunk_size
 api_l4shmc_chunk_cons, 408
l4shmc_chunk_try_to_take
 api_l4shmc_chunk_prod, 405
l4shmc_connect_chunk_signal
 api_l4shm, 400
l4shmc_create
 api_l4shm, 399
l4shmc_enable_chunk
 api_l4shmc_chunk_cons, 407
l4shmc_enable_signal
 api_l4shmc_signal_cons, 413
l4shmc_get_chunk
 api_l4shmc_chunk, 402
l4shmc_get_chunk_to
 api_l4shmc_chunk, 402
l4shmc_get_signal_to
 api_l4shmc_signal, 411
l4shmc_is_chunk_clear
 api_l4shmc_chunk_prod, 405
l4shmc_is_chunk_ready
 api_l4shmc_chunk_cons, 408
l4shmc_iterate_chunk
 api_l4shmc_chunk, 403
l4shmc_signal_cap
 api_l4shmc_signal, 411
l4shmc_trigger
 api_l4shmc_signal_prod, 412
l4shmc_wait_any
 api_l4shmc_signal_cons, 413
l4shmc_wait_any_to
 api_l4shmc_signal_cons, 414
l4shmc_wait_any_try
 api_l4shmc_signal_cons, 413
l4shmc_wait_chunk
 api_l4shmc_chunk_cons, 407
l4shmc_wait_chunk_to
 api_l4shmc_chunk_cons, 407
l4shmc_wait_chunk_try
 api_l4shmc_chunk_cons, 407

l4shmc_wait_signal
 api_l4shmc_signal_cons, 414
 l4shmc_wait_signal_to
 api_l4shmc_signal_cons, 414
 l4shmc_wait_signal_try
 api_l4shmc_signal_cons, 415
 l4sigma0_api
 L4SIGMA0_IPCERROR, 385
 L4SIGMA0_NOFPAGE, 385
 L4SIGMA0_NOTALIGNED, 385
 L4SIGMA0_OK, 385
 L4SIGMA0_SMALLERFPAGE, 385
 L4SIGMA0_IPCERROR
 l4sigma0_api, 385
 L4SIGMA0_NOFPAGE
 l4sigma0_api, 385
 L4SIGMA0_NOTALIGNED
 l4sigma0_api, 385
 L4SIGMA0_OK
 l4sigma0_api, 385
 L4SIGMA0_SMALLERFPAGE
 l4sigma0_api, 385
 l4sigma0_api
 l4sigma0_debug_dump, 387
 l4sigma0_map_anypage, 386
 l4sigma0_map_errstr, 387
 l4sigma0_map_iomem, 385
 l4sigma0_map_kip, 385
 l4sigma0_map_mem, 385
 l4sigma0_map_tbuf, 386
 l4sigma0_new_client, 387
 l4sigma0_return_flags_t, 385
 l4sigma0_debug_dump
 l4sigma0_api, 387
 l4sigma0_map_anypage
 l4sigma0_api, 386
 l4sigma0_map_errstr
 l4sigma0_api, 387
 l4sigma0_map_iomem
 l4sigma0_api, 385
 l4sigma0_map_kip
 l4sigma0_api, 385
 l4sigma0_map_mem
 l4sigma0_api, 385
 l4sigma0_map_tbuf
 l4sigma0_api, 386
 l4sigma0_new_client
 l4sigma0_api, 387
 l4sigma0_return_flags_t
 l4sigma0_api, 385
 l4sysDefines
 L4_DECLARE_CONSTRUCTOR, 123
 L4_EXPORT, 124
 L4_HIDDEN, 124
 l4util_add8
 l4util_atomic, 311
 l4util_add8_res
 l4util_atomic, 311
 l4util_api
 l4_sleep_forever, 355
 l4util_micros2l4to, 357
 l4util_splitlog2_hdl, 355
 l4util_splitlog2_size, 356
 l4util_atomic
 l4util_add8, 311
 l4util_add8_res, 311
 l4util_atomic_add, 312
 l4util_atomic_inc, 312
 l4util_cmpxchg, 309
 l4util_cmpxchg16, 309
 l4util_cmpxchg32, 308
 l4util_cmpxchg64, 308
 l4util_cmpxchg8, 309
 l4util_inc8, 312
 l4util_inc8_res, 312
 l4util_xchg, 311
 l4util_xchg16, 311
 l4util_xchg32, 310
 l4util_xchg8, 311
 l4util_atomic_add
 l4util_atomic, 312
 l4util_atomic_inc
 l4util_atomic, 312
 l4util_bitops
 l4util_bsf, 318
 l4util_bsr, 317
 l4util_btc, 317
 l4util_btr, 316
 l4util_bts, 316
 l4util_clear_bit, 315
 l4util_complement_bit, 315
 l4util_find_first_set_bit, 318
 l4util_find_first_zero_bit, 319
 l4util_next_power2, 319
 l4util_set_bit, 314
 l4util_test_bit, 315
 l4util_bsf
 l4util_bitops, 318
 l4util_bsr
 l4util_bitops, 317
 l4util_btc
 l4util_bitops, 317
 l4util_btr
 l4util_bitops, 316
 l4util_bts
 l4util_bitops, 316
 l4util_clear_bit

l4util_bitops, 315
l4util_cmpxchg
 l4util_atomic, 309
l4util_cmpxchg16
 l4util_atomic, 309
l4util_cmpxchg32
 l4util_atomic, 308
l4util_cmpxchg64
 l4util_atomic, 308
l4util_cmpxchg8
 l4util_atomic, 309
l4util_complement_bit
 l4util_bitops, 315
l4util_cpu
 l4util_cpu_capabilities, 297
 l4util_cpu_capabilities_nocheck, 297
 l4util_cpu_has_cpuid, 296
l4util_cpu_capabilities
 l4util_cpu, 297
l4util_cpu_capabilities_nocheck
 l4util_cpu, 297
l4util_cpu_has_cpuid
 l4util_cpu, 296
l4util_elf
 DF_1_CONFALT, 344
 DF_1_DIRECT, 343
 DF_1_DISPRLDNE, 344
 DF_1_DISPRLPND, 344
 DF_1_ENDFILTEE, 344
 DF_1_GLOBAL, 342
 DF_1_GROUP, 343
 DF_1_INTERPOSE, 343
 DF_1_LOADFLTR, 343
 DF_1_NODEFLIB, 343
 DF_1_NODEDELETE, 343
 DF_1_NODUMP, 343
 DF_1_NOOPEN, 343
 DF_1_NOW, 342
 DF_1_ORIGIN, 343
 DF_P1_GROUPPERM, 344
 DF_P1_LAZYLOAD, 344
 DT_HIPROC, 342
 DT_LOPROC, 342
 DT_NULL, 342
 EI_CLASS, 336
 EI_DATA, 336, 337
 EI_OSABI, 338
 EI_PAD, 340
 EI_VERSION, 338
 ELFCIASSNONE, 336
 ELFDATA2LSB, 337
 ELFDATA2MSB, 337
 ELFDATANONE, 337
 ELFOSABI_AIX, 339
 ELFOSABI_FREEBSD, 339
 ELFOSABI_HPUX, 339
 ELFOSABI_IRIX, 339
 ELFOSABI_LINUX, 339
 ELFOSABI_MODESTO, 340
 ELFOSABI_NETBSD, 339
 ELFOSABI_OPENBSD, 340
 ELFOSABI_SOLARIS, 339
 ELFOSABI_SYSV, 338
 ELFOSABI_TRU64, 339
 EM_ARC, 340
 NT_VERSION, 342
 PT_GNU_EH_FRAME, 341
 PT_GNU_RELRO, 341
 PT_GNU_STACK, 341
 PT_HIOS, 341
 PT_HIPROC, 341
 PT_L4_AUX, 342
 PT_L4_KIP, 342
 PT_L4_STACK, 341
 PT_LOOS, 341
 PT_LOPROC, 341
 SHF_GROUP, 340
 SHF_MASKOS, 341
 SHF_TLS, 340
 SHT_NUM, 340
l4util_find_first_set_bit
 l4util_bitops, 318
l4util_find_first_zero_bit
 l4util_bitops, 319
l4util_idt_desc_t, 762
l4util_idt_header_t, 762
l4util_in16
 l4util_portio, 358
l4util_in32
 l4util_portio, 359
l4util_in8
 l4util_portio, 358
l4util_inc8
 l4util_atomic, 312
l4util_inc8_res
 l4util_atomic, 312
l4util_ins16
 l4util_portio, 359
l4util_ins32
 l4util_portio, 359
l4util_ins8
 l4util_portio, 359
l4util_kip_api
 l4util_kip_for_each_feature, 345
 l4util_kip_kernel_abi_version, 346
 l4util_kip_kernel_has_feature, 346
 l4util_kip_kernel_is_ux, 346
 l4util_memdesc_vm_high, 346

l4util_kip_for_each_feature
 l4util_kip_api, 345
 l4util_kip_kernel_abi_version
 l4util_kip_api, 346
 l4util_kip_kernel_has_feature
 l4util_kip_api, 346
 l4util_kip_kernel_is_ux
 l4util_kip_api, 346
 l4util_mb_addr_range_t, 764
 l4util_mb_apm_t, 764
 l4util_mb_drive_t, 765
 drive_cylinders, 765
 drive_mode, 765
 drive_number, 765
 l4util_mb_info_t, 766
 l4util_mb_mod_t, 767
 mod_end, 768
 mod_start, 768
 l4util_mb_vbe_ctrl_t, 768
 l4util_mb_vbe_mode_t, 769
 l4util_memdesc_vm_high
 l4util_kip_api, 346
 l4util_micros2l4to
 l4util_api, 357
 l4util_next_power2
 l4util_bitops, 319
 l4util_out16
 l4util_portio, 360
 l4util_out32
 l4util_portio, 360
 l4util_out8
 l4util_portio, 360
 l4util_outs16
 l4util_portio, 361
 l4util_outs32
 l4util_portio, 361
 l4util_outs8
 l4util_portio, 360
 l4util_parse_cmd
 parse_cmdline, 347
 l4util_portio
 l4util_in16, 358
 l4util_in32, 359
 l4util_in8, 358
 l4util_ins16, 359
 l4util_ins32, 359
 l4util_ins8, 359
 l4util_out16, 360
 l4util_out32, 360
 l4util_out8, 360
 l4util_outs16, 361
 l4util_outs32, 361
 l4util_outs8, 360
 l4util_rand

l4util_random, 350
 l4util_random
 l4util_rand, 350
 l4util_srand, 350
 l4util_set_bit
 l4util_bitops, 314
 l4util_splitlog2_hdl
 l4util_api, 355
 l4util_splitlog2_size
 l4util_api, 356
 l4util_srand
 l4util_random, 350
 l4util_test_bit
 l4util_bitops, 315
 l4util_tsc
 l4_busy_wait_ns, 303
 l4_busy_wait_us, 303
 l4_calibrate_tsc, 304
 l4_get_hz, 306
 l4_ns_to_tsc, 302
 l4_rdpmc, 301
 l4_rdpmc_32, 301
 l4_rdtsc, 300
 l4_rdtsc_32, 301
 l4_tsc_init, 304
 l4_tsc_to_ns, 301
 l4_tsc_to_s_and_ns, 302
 l4_tsc_to_us, 302
 l4util_xchg
 l4util_atomic, 311
 l4util_xchg16
 l4util_atomic, 311
 l4util_xchg32
 l4util_atomic, 310
 l4util_xchg8
 l4util_atomic, 311
 L4vcpu::State, 880
 add, 881
 clear, 881
 set, 881
 State, 881
 L4vcpu::Vcpu, 917
 cast, 925
 entry_ip, 924
 entry_sp, 924
 ext_alloc, 925
 halt, 923
 i, 924
 irq_disable_save, 921
 irq_enable, 922
 irq_restore, 922
 is_irq_entry, 923
 is_page_fault_entry, 923
 r, 923, 924

saved_state, 922
 state, 921, 922
 task, 923
L4VCPU_IRQ_STATE_DISABLED
 api_libvcpu, 391
L4VCPU_IRQ_STATE_ENABLED
 api_libvcpu, 391
l4vcpu_ext_alloc
 api_libvcpu_ext, 397
l4vcpu_halt
 api_libvcpu, 395
l4vcpu_irq_disable
 api_libvcpu, 392
l4vcpu_irq_disable_save
 api_libvcpu, 393
l4vcpu_irq_enable
 api_libvcpu, 393
l4vcpu_irq_restore
 api_libvcpu, 394
l4vcpu_irq_state_t
 api_libvcpu, 391
l4vcpu_is_irq_entry
 api_libvcpu, 396
l4vcpu_is_page_fault_entry
 api_libvcpu, 396
l4vcpu_print_state
 api_libvcpu, 395
l4vcpu_state
 api_libvcpu, 391
link
 L4Re::Vfs::Directory, 566
List_alloc
 cxx::List_alloc, 773
log
 L4Re::Env, 588, 591
Log interface, 66
Logging interface, 104
Low-Level Thread Functions, 351
lower_bound_node
 cxx::Avl_map, 444
 cxx::Avl_set, 452
 cxx::Bits::Bst, 501
lseek64
 L4Re::Vfs::Regular_file, 835
Machine Restarting Function, 350
magic
 l4_vhw_descriptor, 742
main_thread
 L4Re::Env, 588, 591
map
 L4::Task, 888
 L4Re::Dataspace, 541
 L4Re::Util::Dataspace_svr, 548
 Map_ro
 L4Re::Dataspace, 541
 Map_rw
 L4Re::Dataspace, 541
 Map_flags
 L4Re::Dataspace, 541
 map_hook
 L4Re::Util::Dataspace_svr, 549
 map_region
 L4Re::Dataspace, 542
 mask
 L4::Icu, 652
 max
 cxx_api, 46
 max_free_slabs
 cxx::Base_slab, 468
 cxx::Base_slab_static, 472
 Mem_alloc
 api_l4re_protocols, 108
 mem_alloc
 L4Re::Env, 587, 591
 Mem_alloc_flags
 L4Re::Mem_alloc, 786
 Mem_desc
 L4::Kip::Mem_desc, 788
 mem_size
 l4_vhw_entry, 744
 mem_start
 l4_vhw_entry, 744
 Memory allocator, 69
 Memory allocator API, 105
 Memory descriptors (C version), 222
 Memory operations., 290
 Memory related, 166
 Message Items, 151
 Message Registers (MRs), 277
 Message Tag, 257
 min
 cxx_api, 46
 mkdir
 L4Re::Vfs::Directory, 565
 mod_end
 l4util_mb_mod_t, 768
 mod_start
 l4util_mb_mod_t, 768
 Mode
 L4Re::Util::Event_t, 611
 Mode_irq
 L4Re::Util::Event_t, 611
 Mode_polling
 L4Re::Util::Event_t, 611
 modify_senders
 L4::Thread, 904
 mount

L4Re::Vfs::File_system, 628
 L4Re::Vfs::Fs, 631
 move
 L4::Cap, 513
 Msg_ptr
 L4::Ipc::Msg_ptr, 800
 msg_ptr
 ipc_fw, 55
 msi_info
 L4::Icu, 651

N
 cxx::Bits::Direction, 562
 Name-space API, 106
 Namespace
 api_l4re_protocols, 108
 Namespace interface, 73
 next
 L4Re::Event_buffer_t, 604
 No_init
 L4::Cap_base, 521
 No_init_type
 L4::Cap_base, 521
 NT_VERSION
 l4util_elf, 342
 num_interfaces
 L4::Meta, 795

Object Invocation, 193
 object_size
 cxx::Base_slab, 468
 cxx::Base_slab_static, 472
 objects_per_slab
 cxx::Base_slab, 468
 cxx::Base_slab_static, 472
 operator l4_msgtag_t
 L4::Factory::S, 853
 operator new
 cxx_api, 46
 operator Priv_type *
 cxx::Auto_ptr, 439
 operator<<
 ipc_stream, 50–52
 L4::Factory::S, 853, 854
 operator>>
 ipc_stream, 47–50
 operator*
 cxx::Auto_ptr, 438
 operator()
 cxx::Pair_first_compare, 821
 operator->
 cxx::Auto_ptr, 438
 operator=
 cxx::Auto_ptr, 438

operator==
 L4Re::Video::Color_component, 528
 L4Re::Video::Pixel_info, 830
 outchar
 l4_debugger_api, 177
 outdec
 l4_debugger_api, 178
 outhex12
 l4_debugger_api, 178
 outhex16
 l4_debugger_api, 178
 outhex20
 l4_debugger_api, 178
 outhex32
 l4_debugger_api, 177
 outhex8
 l4_debugger_api, 178
 outnstring
 l4_debugger_api, 177
 outstring
 l4_debugger_api, 177

page_shift
 L4Re::Util::Dataspace_svr, 550
 Pager
 L4Re::Rm, 840
 pager
 L4::Thread::Attr, 430, 431
 Pair
 cxx::Pair, 820
 Pair_first_compare
 cxx::Pair_first_compare, 821
 Parent
 api_l4re_protocols, 108
 parent
 L4Re::Env, 587, 590
 Parent API, 106
 parse_cmdline
 l4util_parse_cmd, 347
 phys
 L4Re::Dataspace, 544
 L4Re::Util::Dataspace_svr, 549
 Pinned
 L4Re::Mem_alloc, 786
 Pixel_info
 L4Re::Video::Pixel_info, 827
 print
 L4Re::Log, 782
 printf
 L4Re::Log, 782
 Priority related functions, 349
 process
 L4Re::Util::Event_buffer_consumer_t, 601
 Producer, 404, 412

Protocols
 api_l4re_protocols, 108
provider_pid
 l4_vhw_entry, 744
PT_GNU_EH_FRAME
 l4util_elf, 341
PT_GNU_RELRO
 l4util_elf, 341
PT_GNU_STACK
 l4util_elf, 341
PT_HIOS
 l4util_elf, 341
PT_HIPROC
 l4util_elf, 341
PT_L4_AUX
 l4util_elf, 342
PT_L4_KIP
 l4util_elf, 342
PT_L4_STACK
 l4util_elf, 341
PT_LOOS
 l4util_elf, 341
PT_LOPROC
 l4util_elf, 341
push_back
 cxx::List, 771
 cxx::List_item, 777
push_front
 cxx::List, 771
 cxx::List_item, 777
put
 L4::Ipc::Ostream, 814
 L4Re::Event_buffer_t, 604
query
 L4Re::Namespace, 804, 805
query_log_name
 L4::Debugger, 558
query_log_typeid
 L4::Debugger, 558

R
 cxx::Bits::Direction, 562

r
 L4Re::Video::Pixel_info, 828, 829
 L4vcpu::Vcpu, 923, 924

Random number support, 349

rbegin
 cxx::Avl_set, 454, 455
 cxx::Bits::Bst, 499, 500

read
 L4::Vcon, 913

Read_only
 L4Re::Rm, 840

readyv
 L4Re::Vfs::Regular_file, 834

Realtime API, 209

receive
 L4::Ipc::Istream, 692
 L4::Irq, 681

Ref_type
 cxx::Auto_ptr, 437

refresh
 L4Re::Util::Video::Goos_svr, 645
 L4Re::Video::View, 930

Region map API, 108

Region map interface, 74

Region_flags
 L4Re::Rm, 840

Region_flags
 L4Re::Rm, 840

register_del_irq
 L4::Thread, 903

Register_flags
 L4Re::Namespace, 804

register_obj
 L4Re::Namespace, 805

release
 cxx::Auto_ptr, 438
 L4Re::Util::Dataspace_svr, 550

release_cap
 L4::Task, 891

remove
 cxx::Avl_map, 444
 cxx::Avl_set, 452
 cxx::Avl_tree, 460
 cxx::List, 772
 cxx::List_item, 778

remove_me
 cxx::List_item, 777
 cxx::List_item::Iter, 698
 cxx::List_item::T_iter, 885

rename
 L4Re::Vfs::Directory, 566

rend
 cxx::Avl_set, 454, 455
 cxx::Bits::Bst, 500

Reply_compound
 L4::Ipc_svr, 426

Reply_separate
 L4::Ipc_svr, 426

reply_and_wait
 L4::Ipc::Iostream, 672, 673

Reply_mode
 L4::Ipc_svr, 425

reserve_area
 L4Re::Rm, 841, 842

Reserved

L4Re::Rm, 840
 reset
 L4::Ipc::Iostream, 671
 L4::Ipc::Istream, 688
 Rm
 api_l4re_protocols, 108
 rm
 L4Re::Env, 587, 591
 rmdir
 L4Re::Vfs::Directory, 567
 Ro
 L4Re::Namespace, 804
 run
 L4::Vm, 934
 run_thread
 L4::Scheduler, 857
 Rw
 L4Re::Namespace, 804

 S
 L4::Factory::S, 852
 saved_state
 L4Vcpu::Vcpu, 922
 scan_zero
 cxx::Bitmap_base, 489
 Scheduler, 225
 scheduler
 L4Re::Env, 592
 screen_info
 L4Re::Util::Video::Goos_svr, 645
 Search_addr
 L4Re::Rm, 841
 send
 L4::Ipc::Ostream, 815
 L4::Vcon, 912
 Server
 L4::Server, 862
 set
 L4::Kip::Mem_desc, 792
 L4Re::Video::Color_component, 529
 L4Vcpu::State, 881
 set_attr
 L4::Vcon, 914
 set_bit
 cxx::Bitmap_base, 488
 set_fd
 L4Re::Vfs::Fs, 631
 set_info
 L4Re::Video::View, 930
 set_lock
 L4Re::Vfs::Regular_file, 836
 set_mode
 L4::Icu, 653
 set_object_name

 L4::Debugger, 557
 set_status_flags
 L4Re::Vfs::Generic_file, 635
 set_viewport
 L4Re::Video::View, 930
 Shared Memory Library, 397
 SHF_GROUP
 l4util_elf, 340
 SHF_MASKOS
 l4util_elf, 341
 SHF_TLS
 l4util_elf, 340
 shift
 L4Re::Video::Color_component, 528
 SHT_NUM
 l4util_elf, 340
 si
 l4_vcpu_regs_t, 736
 Sigma0 API, 383
 signal
 L4Re::Parent, 824
 Signals, 409
 size
 cxx::List, 772
 L4::Ipc::Buf_cp_out, 508
 L4::Kip::Mem_desc, 790
 L4Re::Dataspace, 545
 L4Re::Video::Color_component, 528
 size0
 l4_tracebuffer_status_t, 730
 size1
 l4_tracebuffer_status_t, 730
 skip
 L4::Ipc::Istream, 689
 slab_size
 cxx::Base_slab, 468
 cxx::Base_slab_static, 472
 slice
 api_calls_rt_sched, 134, 135
 Small C++ Template Library, 44
 Smart_cap
 L4::Smart_cap, 875
 snd_base
 L4::Cap_base, 525
 Split_ds
 L4Re::Rm, 840
 stack
 L4Re::Video::View, 930
 start
 L4::Kip::Mem_desc, 789
 State
 L4Vcpu::State, 881
 state
 L4Vcpu::Vcpu, 921, 922

stats_time
 L4::Thread, 901
Strong
 L4Re::Namespace, 804
sub_type
 L4::Kip::Mem_desc, 791
Super_pages
 L4Re::Mem_alloc, 786
supports
 L4::Meta, 796
switch_log
 L4::Debugger, 558
switch_to
 L4::Thread, 900
symlink
 L4Re::Vfs::Directory, 567
tag
 L4::Ipc::Istream, 690
 L4::Ipc::Ostream, 814, 815
take
 L4Re::Util::Dataspace_svr, 549
Task, 230
task
 L4Re::Env, 588
 L4Vcpu::Vcpu, 923
Thread, 238
Thread control, 251
Thread Control Registers (TCRs), 278
Timeouts, 154
Timestamp Counter, 299
total_objects
 cxx::Base_slab, 469
 cxx::Base_slab_static, 473
tracebuffer0
 l4_tracebuffer_status_t, 730
tracebuffer1
 l4_tracebuffer_status_t, 730
trigger
 L4::Irq, 683
type
 L4::Kip::Mem_desc, 791
 l4_vhw_entry, 744
 L4Re::Vfs::Be_file_system, 482
 L4Re::Vfs::File_system, 628
unbind
 L4::Icu, 650
unlink
 L4Re::Vfs::Directory, 566
unlock_all_locks
 L4Re::Vfs::Generic_file, 634
unmap
 L4::Task, 889
 unmap_batch
 L4::Task, 890
 unmask
 L4::Icu, 652
 L4::Irq, 683
 utcb_area
 L4Re::Env, 589, 592
 Utility Functions, 353
 ux_host_syscall
 L4::Thread::Attr, 433
 V_flags
 L4Re::Video::View, 929
 valid
 cxx::Avl_set::Node, 807
 validate
 L4::Cap_base, 526
 Value
 L4::Basic_registry, 474
 vCPU API, 287
 vCPU Support Library, 389
 vcpu_control
 L4::Thread, 902
 vcpu_control_ext
 L4::Thread, 903
 vcpu_resume_commit
 L4::Thread, 902
 vcpu_resume_start
 L4::Thread, 901
 version
 l4_vhw_descriptor, 742
 version0
 l4_tracebuffer_status_t, 730
 version1
 l4_tracebuffer_status_t, 730
 Video API, 87
 view
 L4Re::Video::Goos, 641
 view_info
 L4Re::Util::Video::Goos_svr, 645
 Virtual Console, 281
 Virtual Machines, 185
 Virtual Registers (UTCBs), 273
 VM API for SVM, 161
 VM API for TZ, 292
 VM API for VMX, 162
 wait
 L4::Ipc::Istream, 690, 691
 L4::Irq, 682
 words
 cxx::Bitmap_base, 487
 write
 L4::Vcon, 913

writev
 L4Re::Vfs::Regular_file, [835](#)
x86 Virtual Registers (UTCB), [295](#)